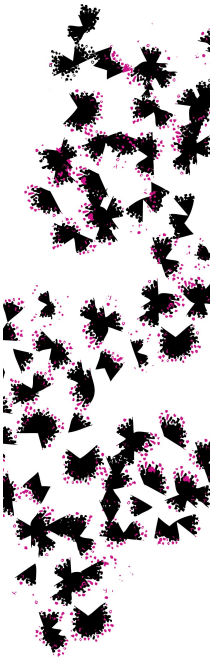


Towards an Interactive Drone,
A Bayesian Optimization Approach

A. (Asem) Khattab

MSc Report

Committee:
Prof.dr.ir. S. Stramigioli
Dr.ir. J.B.C. Engelen
R. Hashem, MSc
Dr. M. Poel

August 2018

030RAM2018
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE.

MIRA CTIT
BIOMEDICAL TECHNOLOGY
AND TECHNICAL MEDICINE

Summary

For a drone, equipped with an impedance controller, to correctly deal with physical contact and interaction with the environment, the impedance parameters should be adjusted correctly depending on the interaction task.

Concentrating on interaction tasks requiring a constant set of impedance parameters values throughout operation. A model-free learning framework is proposed to automatically find the suitable parameters values. The framework relies on Bayesian optimization and episodic reward calculation requiring the drone to repeatedly perform a predetermined task in the environment actively searching in the impedance parameters space.

The sample-efficiency and safety of learning were improved by adding two novel modifications to Bayesian optimization. The first one is local optimization of acquisition functions allowing learning to get early warnings during exploration. The second one is conditioning the reward signal with a logistic function exploiting the initial knowledge and enabling generalizing learning settings to different situations.

The proposed technique was validated by applying it to learn the suitable impedance parameters values for a simulated drone performing sliding tasks. The results show that the proposed framework is able to automatically find suitable impedance parameters values in different situations given the same initial knowledge and that the learned parameters values can be generalized to similar interaction tasks.

Contents

1 Introduction

With their high mobility, drones can be employed in various industrial tasks significantly decreasing their cost. Examples include inspection of high structures like wind turbines, cell towers and power lines. There is a mature research body in position control of drones with commercial products already available in the market. Drones are now well capable of handling all tasks requiring mere free flight. Exploiting advances in other fields like machine learning, computer vision and path planning, drones can perform a wide range of tasks like object identifying and tracking as well as surveying and imaging.

However, there is still much research work to be done to allow drones to handle interaction and contact tasks with the environment. In these situations, the force applied by the drone on the environment is of great influence to the success of the task. Motion control alone turns out to be unsuitable as the unavoidable modeling errors and uncertainties can cause the contact force to rise, leading to an unstable behavior during the interaction, especially in the presence of rigid environments [31]. To solve this issue, several control schemes have been proposed in the literature known as interaction (or force) control schemes. While these schemes have been extensively studied and experimentally applied for robotic manipulators, there is a lack in literature in their application to drones.

Interaction control is an essential requirement for a successful employment of drones in industrial applications as it will give them the ability to perform direct contact tasks like parts insertion and replacement as well as surface polishing and writing. With these capabilities, a drone can be easily used to handle, for instance, the maintenance of wind turbines.

One of the most widely used interaction control schemes is impedance control [31]. The robot (the drone in our case) then behaves as an impedance in the environment modeled as a mass-spring-damper system with adjustable parameters. Adjusting these controller's parameters properly, one can achieve a compliant behavior (where the robot is more lenient with the disturbances imposed by the environment) or a stiff behavior (where the robot is more precise in tracking a goal and more stable against unpredictable disturbances). The main problem is: selecting good impedance parameters ensuring the desired behavior is not an easy straightforward task [31]. It requires either an accurate model of the environment where the contact will occur or a human expert tuning the parameters simply by trial and error. The parameters values that will result in a satisfactory performance heavily depend on the nature of the task and the physical and dynamic characteristics of the drone and the environment.

It was noted that it is the ability to quickly adapt the impedance parameter of the body to the faced situation that allows human and other biological systems to have a robust and safe interaction with the environment [28]. In fact, some sports are harder to master because they require more precise balance between compliance and stiffness. It can take humans more than two decades to develop and tune this balance [28]. Giving this learning ability to robots is the key towards robust and safe robotic interaction.

The aim of this thesis is to design and apply a suitable model-free approach to allow a drone, equipped with an impedance controller, to autonomously learn suitable impedance parameters values for a certain task. Successfully achieving this will be a step forward towards a fully autonomous interactive drone.

1.1 Related Works

In general, active interaction control schemes can be divided to two main categories: direct and indirect force control. In direct force control, the contact force and moment are explicitly controlled with a force feedback loop. This requires reasonably precise 6D force measurements. Instead, in this thesis, we work with indirect force control. One of the well known and widely used schemes in this category is impedance control, where the deviation of the end-effector motion from the desired motion due to the interaction with the environment is related to the contact force through a mechanical impedance. For more, see Appendix A.

Techniques of learning impedance parameters in literature vary depending on the application and usage of the impedance controller. In a medical application, human demonstrations were employed to learn impedance parameters for a lower-limb prostheses [3]. The scope of the technique is limited to situations where demonstrations of the correct behavior can be easily obtained and the number of needed demonstrations is small.

Learning by demonstration is more connected to supervised learning than it is to reinforcement learning. The agent in this case does not perform any form of trial-and-error. It directly infers the parameters values from the demonstration. Hence, it is not suitable for general autonomous learning in unpredictable environments and dynamics. Only reinforcement learning can offer a general learning approach that can deal with different unpredictable situations.

Another supervised-learning-based method was proposed in [19], where a model of the dynamics is learned using locally weighted projection regression. This model is then used to derive an optimal control policy for variable impedance actuators. In addition to lacking the interactive learning feature in reinforcement learning, the accuracy of this method is limited by the level of details in the physical model and the precision of identifying this model through supervised learning.

Other works investigated learning a *trajectory* of impedance parameters such that the parameters values vary with time depending on the need of the task. For example, in [6, 28], the PI^2 algorithm (policy improvement with path integrals) is used to learn impedance parameters trajectories (gains schedules) for robots with high degrees-of-freedom. The work combines aspects form optimal control and reinforcement learning to train robots to do tasks like flipping a light switch and pushing open a door. During such tasks, both the position trajectory and the impedance parameters trajectories are learned.

While such methods have the advantage of being model-free and hence able to deal with different environments and unpredictable situations, they suffer from two main problems. The first one is that the learning is very specific to the task. For example, if the size of the switch, its weight or its stiffness changes, the whole switch flipping task needs to be *relearned*. Further, because the learning objective is to find a trajectory of the parameters along time not a fixed set of parameters, the state-action space is huge, which requires a large number of iterations (trials) for the learning to reach reasonable results. To remedy this, kinesthetic teaching (moving the robot by hand to do the task) is used to give meaningful initial trajectories. Still, learning needs more than a hundred updates to converge.

1.2 Problem Formulation and Challenges

Given an interaction controller (impedance/admittance) implemented on a drone, the goal of this thesis is to devise a suitable framework for model-free learning of the optimal controller's parameters values for achieving a certain interaction task. The focus is on tasks requiring a fixed set of parameters values throughout operation. Examples of such tasks include surface contact tasks like polishing, cleaning and writing.

Considering such tasks, the optimal set of parameters values are greatly dependent on the physical nature of the surface as well as the end-effector. In addition to that, unpredictable

aerodynamics play a significant role. The aim is to perform efficient trial-and-error learning to automatically find the optimal parameters values. The idea is illustrated in Fig. 1.1. The drone, with its initial set of parameters values, performs unsatisfactorily (left circle) in the task of drawing a circle as the physical characteristics of the arbitrary surface are unknown to the user. Through smart trial-and-error, the drone learns impedance parameters values that give better performances (middle circle). By the end of training (or learning), the drone finds the optimal set of parameters values (right circle).

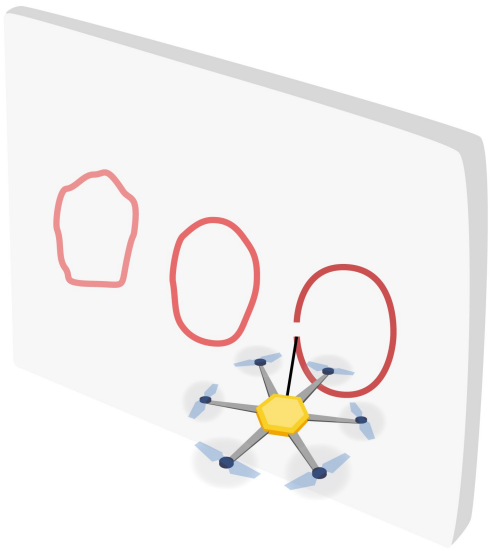


Figure 1.1: A drone training to find optimal impedance parameters for drawing a circle on a surface with unknown physical characteristics.

It can be realized that the task of finding the optimal set of impedance parameters values is actually equivalent to the optimization of an expensive black-box function. The function being optimized $f(\mathbf{x})$ represents a performance measure for the drone under a selected set of impedance parameters values $\mathbf{x}$. We call it the reward function. It is black-box because there is not a closed form mathematical expression for the function in terms of the parameters values $\mathbf{x}$. Otherwise the problem can be solved without learning. Instead, the function is costly evaluated at a certain point $\mathbf{x}$ (a certain set of impedance parameters values) by observing the performance of the drone while doing an application-specific predetermined task (like following a trajectory to draw a circle as in Fig. 1.1). Based on that, $f(\mathbf{x})$ gives a single number indicating how satisfactory the performance was.

The learning problem becomes then the problem of finding the position of the global maximum of that black-box reward function $\mathbf{x}^* = \arg\max f(\mathbf{x})$ using as few evaluations as possible to minimize the learning cost. A simple block diagram of the learning problem is shown in Fig. 1.2.

This formulation has two main advantages over the formulations of the methods in literature reviewed in section 1.1. First, compared to supervised learning approaches, the proposed formulation requires the drone to do a model-free trial-and-error learning without relying on providing demonstrations. Second, compared to methods that learn a *trajectory* of impedance parameters making the learning too specific to the learned task, the proposed formulation involves learning the single optimal set of impedance parameters values for a task given the environment. This allows the same learning result to be used for similar tasks. For example, if the optimal impedance parameters values for drawing a square on specific surface were learned, the same learned parameters can be used to draw bigger squares.

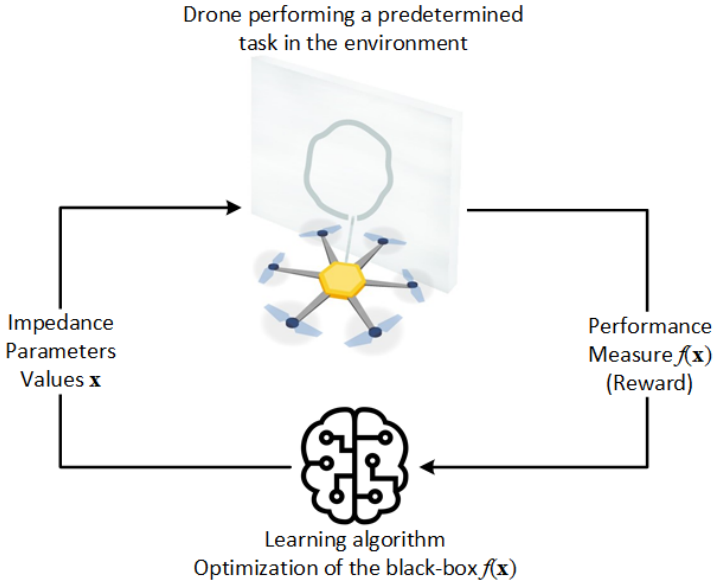


Figure 1.2: Simple block diagram of the proposed learning problem formulation.

The main goals of this work are then to find a suitable technique for autonomously learning the optimal parameters values for a task (by optimizing the black-box reward function), adapt this technique to the problem at hand considering the limitations imposed by working with a robotic drone system, and finally validate the proposed method via simulations.

Challenges

Achievement of these goals is a challenging task. Two challenges should be highlighted for their significant influence. The first one is the high cost of learning. Aside from the power costs of operating the drone, operation cannot be done without human supervision. In addition, in case of crashes due to instability or bad performance often encountered during exploration (trying new impedance parameters values) in learning, fixing the drone is expensive financially and time-wise. For that, to be suitable for the problem, the learning method has to be sample-efficient such that it doesn't require much operation time to learn.

The second challenge is the continuity and high dimensionality of the parameters space. The number of impedance parameters to learn (the degrees-of-freedom in this learning problem) can reach 27 (See Appendix A for details). It is very challenging to find an interactive learning technique that scales up to this high number of dimensions with few samples (short operation time). Other approaches can be also taken to go around this high dimensionality problem. One of them is reducing the dimensionality of the learning depending on the task by adding constraints on the parameters values.

Another challenge closely related to the previous two is that the parameters space is not completely safe to explore. In other words, setting the impedance parameters to certain values might result in the instability and consequently collapse of the drone. It becomes necessary then to provide a way for an early failure detection. If instability is detected, the impedance parameters should be quickly set on some default safe values and the operation of the drone should be stopped before any crashes.

1.3 Proposed Method

In this thesis, an episodic learning framework is proposed. The proposed framework is illustrated in Fig. 1.3. At each episode, the drone receives a set of parameters values. The impedance is configured according to the received values and then the drone performs a task designed by the user to assess the performance of the chosen impedance settings. In our case, this task is following a prescribed trajectory (sent from the motion planner). While performing the task, all relevant sensory measurements (from on-board sensors or off-board ones in the environment) are stored and can be also used to detect any failure in real time. In case the task was completed successfully, the stored sensors measurements are used to compute the reward (the performance measure) which can include different elements (For example, the root-mean-square (RMS) of the position error) depending on the objective of the learning. This reward is sent back to the learning algorithm, which should intelligently select another set of parameters values to try in order to quickly find a satisfactory set of parameters values (a one that gives a high reward). If a failure is detected, the impedance settings are all restored to safe default values and a very small reward (or a penalty in this case) is returned to the learning algorithm indicating that the performance was highly undesirable. This failure reward is application-specific and is given by the user.

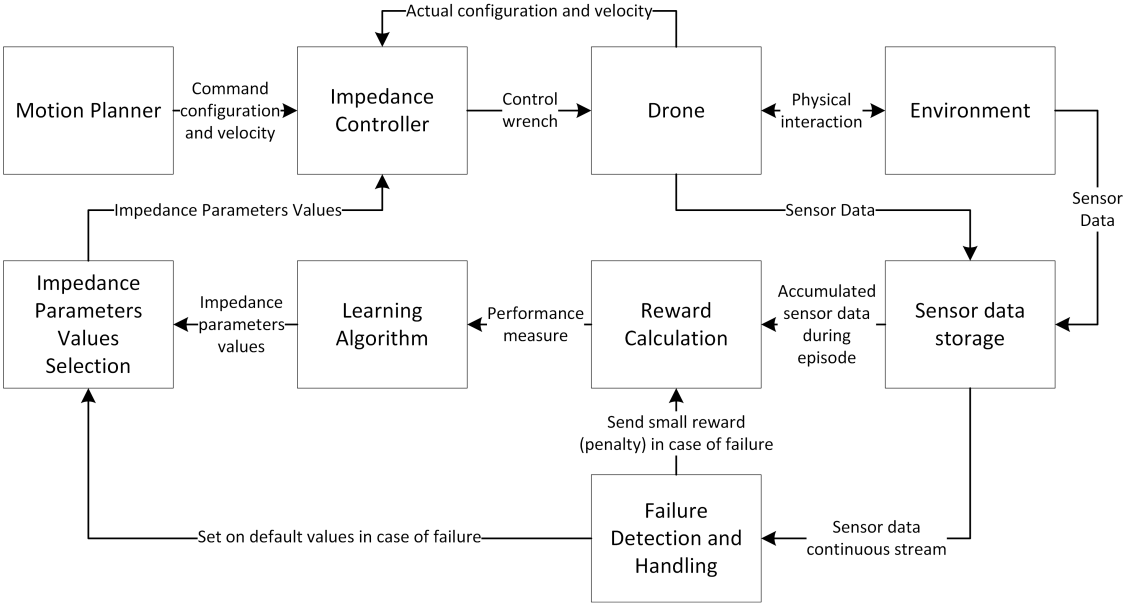


Figure 1.3: Simplified block diagram of the proposed learning framework.

As mentioned in section 1.2, the learning block in Fig 1.3 works on finding the position of the global maximum of the black-box reward function $\mathbf{x}^* = \arg\max f(\mathbf{x})$ where:

- $\mathbf{x}$ is a vector containing the selected impedance parameters values.
- $f(\mathbf{x})$ represents a performance measure of the drone in the interaction task under the selected impedance parameters values.

In this thesis, I propose the use of the Bayesian optimization framework combined with Gaussian processes to optimize this expensive black-box function. Bayesian optimization has recently received an increasing attention from the research community for its potential in solving difficult problems in different areas (see chapter 2). It is very sample-efficient, and provides various schemes for effectively balancing exploration (examining a position $\mathbf{x}$ with high uncertainty about its outcome $f(\mathbf{x})$) and exploitation (trying at positions highly probable to produce high rewards) as it searches for the position of the maximum. It also provides means for incor-

porating human expert knowledge or intuition, if available, in the learning. Above all of that, it scales polynomially with the number of dimensions escaping the curse of dimensionality (the need for exponentially more data and computations as the number of dimensions grows). To the author's best knowledge, this thesis represents the first research effort to apply Bayesian optimization for learning impedance parameters.

1.4 Research Questions

The initial question addressed in this thesis was:

RQ1. Which model-free method is suitable for automatically learning a single set of impedance parameters values for a drone interacting in the environment?'.

After surveying the literature in section 1.1 and formulating the problem as in section 1.2, Bayesian optimization was found to be the suitable solution for the reasons mentioned in section 1.3.

Given the proposed learning framework that uses Bayesian optimization, the following research questions arise:

RQ2. How to adapt Bayesian optimization to the requirements and limitations faced in learning impedance parameters values for a drone.

RQ3. How to apply the proposed learning framework to find impedance parameters values for an interaction task and verify its validity in a simulated environment?

The rest of this thesis is dedicated to answering these two questions.

1.5 Report Layout

The rest of this thesis is arranged as follows:
Chapter 2: Explores the different components of the Bayesian optimization framework and discuses its design space making suitable choices with respect to the application at hand.
Chapter 3: Investigates the performance of Bayesian optimization with different settings by applying it to simulated randomly generated black-box functions with different numbers of dimensions. The gained insights from these results are then used to innovate an effective modification to the Bayesian optimization framework making it safer and more efficient.
Chapter 4: Discusses how to design a reward function presenting a novel modification to the Bayesian optimization framework improving its sample-efficiency. After that, the results of applying the proposed framework to a simulated drone for learning impedance parameters values for sliding tasks are displayed and discussed
Chapter 5: Provides another perspective into the proposed learning framework as a reinforcement learning method.
Finally, the thesis is wrapped up in the **conclusion**.

2 Bayesian Optimization Framework

Bayesian optimization is a framework for sample-efficient search for the extremum (the maximum or the minimum) of a black-box function. That is a function that can only be evaluated or inquired (even via noisy observations) at any arbitary point in its domain. Bayesian optimization is particularly useful for situations where the target function has no closed-form expression, is difficult to get derivatives for, is very expensive to evaluate and/or is non-convex [5, 24].

In this chapter, the different elements of the design space of Bayesian optimization are explored making suitable choices with respect to the application of learning drone's impedance parameters. The chapter starts with a background about the principles of Bayesian optimization and a brief survey of its notable applications in section 2.1. Section 2.2 discusses the probabilistic surrogate model and describes Gaussian processes. Section 2.3 investigates the different ways Bayesian optimization offers for incorporating initial human knowledge (or intuition) in the learning. Section 2.4 touches on the main acquisition functions found in literature describing the intuition behind them. Finally, the chapter is concluded by summarizing the main assumptions and decisions made when using Bayesian optimization for the application at hand.

2.1 Background

Formally, the goal of Bayesian optimization is to find the position $\mathbf{x}^*$ of the global maximum (or minimum) of an unknown target function $f(\mathbf{x})$:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\chi} f(\mathbf{x}) \tag{2.1}$$

where χ is the (multi-dimensional) domain of the target function.

There exists a solid body of research in mathematics for this kind of problems. However, the suggested methods require a large number of evaluations to guarantee reaching the global maximum. Moreover, the number of needed evaluations grows exponentially as the number of dimensions of the problem increases [14, 5]. Such number of evaluations is impractical for expensive-to-evaluate functions, like the one we are considering here.

Instead of relying on the density of evaluations, Bayesian optimization takes a different approach towards the problem. It exploits all the evaluations it can get by building a surrogate model that represents the current belief about the black-box function. It uses this model to decide where to prop (evaluate) the function next in order to make the position of the maximum of the surrogate model as close as possible to the position of the maximum of the black-box function. The decision of the next point to prop is carried out via *acquisition functions*, which provide a way to balance exploration (trying at positions where there is more uncertainty about the value of the target function) and exploitation (trying at positions where the current belief shows the target function has a high value).

The name 'Bayesian' comes from the fact that the framework relies on one of central theorems in statistics known as Bayes' Rule. The theorem simply states that the *posterior* probability of a model (or hypothesis) M given evidence (or observations) E is proportional to the *likelihood* of E given M multiplied by the *prior* probability of M:

$$P(M|E) \propto P(E|M)P(M) \tag{2.2}$$

The used terminology emphasizes the sequential nature of the process of updating our model. We begin by having a *prior* info about the probability of the model $P(M)$. After obtaining a certain evidence (or observation), it can be used to update our *posterior* belief about the model $P(M|E)$ provided that the likelihood of the evidence given the model $P(E|M)$ is known.

In Bayesian optimization, a prior belief over the space of possible target functions is formed based on initial knowledge about the target function (discussed in section 2.3). This model is then sequentially refined via Bayesian posterior updates as new observations are obtained.

A pseudo-code of the general Bayesian optimization framework can be seen in Algorithm 1. At the start, the user collects w observations to form an initial surrogate model. In particular, the user selects points[1] $\mathbf{x}_{1:w}$ and samples the target function $f(\mathbf{x})$ at these points obtaining $y_{1:w}$, where

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

is a (noisy) observation of the target function with noise ϵ_i. The points and their corresponding observations combined in tuples are denoted $\mathscr{D}_{1:w}$ where $\mathscr{D}_i = (\mathbf{x}_i, y_i)$. Each iteration, a surrogate model is built (discussed in section 2.2), this model is used to find the best point to try next via acquisition functions. The selected point is sampled and the process continues. The procedures of Bayesian optimization on a simple 1D example are shown in Fig. 2.1.

Algorithm 1: The General Bayesian Optimization Framework

1 Get observations $\mathscr{D}_{1:w}$ by probing the target function f at w points
2 **for** $n = w, w+1, w+2...$ **do**
3 Build/update the surrogate statistical model $M(f(\mathbf{x})|\mathscr{D}_{1:n})$
4 Select $\mathbf{x}_{n+1}$ by globally optimizing the acquisition function:
$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\chi} \alpha(\mathbf{x} \mid \mathscr{D}_{1:n})$
5 Obtain a (noisy) new observation at $\mathbf{x}_{n+1}$: $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$
6 Augment the data $\mathscr{D}_{1:n+1} = \{\mathscr{D}_{1:n}, (\mathbf{x}_{n+1}, y_{n+1})\}$

Bayesian optimization has a minimal number of assumptions regarding the target function [24]. It requires that the domain χ is given. It also assumes that the target function can be sampled at any position within its domain. It does not require extra conditions like the possibility to evaluate the derivative.

The discussed concepts and elements of the design space along with the main considered choices in this chapter are summarized in Fig. 2.2.

2.1.1 Notable Applications

The technique has proved successful in optimizing expensive black-box functions arising in different fields. The classical example, which is believed to be one of the early drives of research into Bayesian optimization is *Kriging*, a technique that dates back to the sixties and aims at finding the optimal location to drill for extracting a valuable mineral [24, 14]. Evaluating the function being optimized here requires actual drilling at a location. Another example often found in complex machine learning contexts, where the objective is to find the model hyperparameters that will result in the lowest possible cross-validation error when used to fit a certain dataset. Evaluating this function requires actual training of the model and testing it. Bayesian optimization have been recently applied very successfully to find the optimal hyperparameters for such complex models in a way that even surpasses human experts [26]. For a through recent review about Bayesian optimization techniques, applications and practical considerations, see [24].

Bayesian optimization has recently seen few applications in robotics. It was used to learn policy parameters for robots navigation and planning [18]. It was also used to learn grasping parameters of unknown objects [16, 21]. Further, it produced impressive results in helping robots

[1] In this report, we use the notation $x_{i:j} = \{x_i, x_{i+1}, \ldots, x_j\}$.

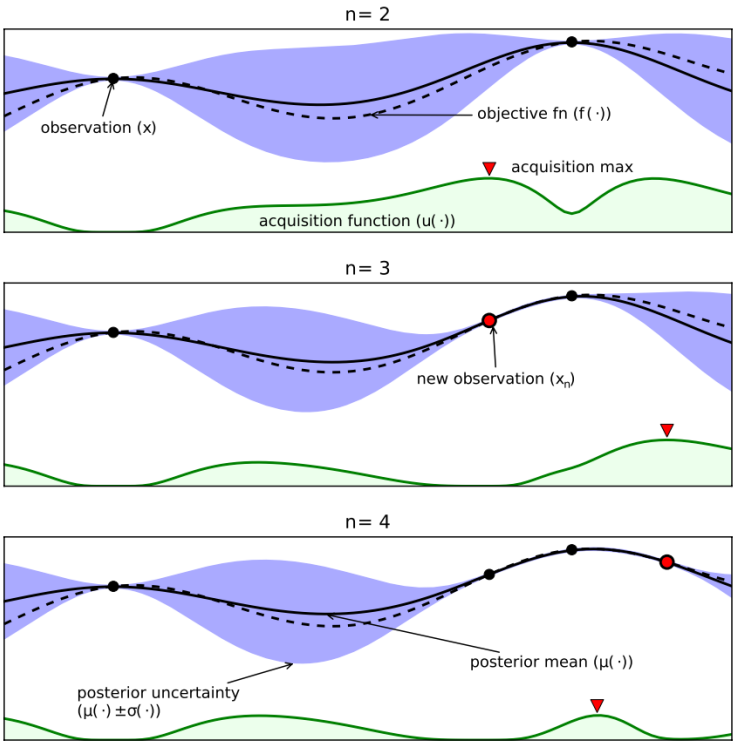


Figure 2.1: Multiple iterations of Bayesian optimization on a 1D example problem. The statistical model used here is a Gaussian process (GP). Starting from two initial points, the GP model is built and the acquisition function is optimized to find the next point to sample. The acquisition function here is high in places where the model predicts a high value for the target function (exploitation) and where the prediction uncertainty is high (exploration). Positions with both features are sampled first. After each new sample, the model is rebuilt and the process is repeated. Notice how no samples where drawn from the area on the far left as it was correctly predicted by the model to offer a litter improvement over the initial point close to that area. (Taken from [5]).

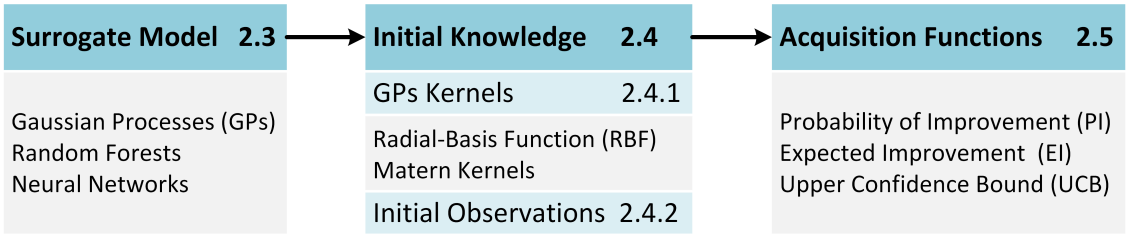


Figure 2.2: Elements and principles of Bayesian optimization investigated in this chapter with the corresponding section number. Arrows indicate prerequisites.

adapt their behaviors when damaged inspired from the way animals adapt when injured [8]. It was also employed to learn gait parameters for robot locomotion [7].

2.2 Surrogate Model

The Bayesian optimization framework is very versatile and offers, due to the active ongoing research, different design choices on all of its elements. Among the most essential choices to be made is the kind of the surrogate statistical model that will be used to approximate the black-box function.

Essentially, the choice here deals with the stochastic model that will act as a prior in the Bayesian update. This model is a distribution over the space of possible target functions given the observations that have been seen until now.

The classical and the most popular choice in literature is Gaussian processes (GPs). However, other models have been used also, like random forests [24], which is especially suitable for categorical data. Recently also, the use of neural networks as models in Bayesian optimization was investigated [27]. The most significant reason for using models other than the classically used GPs is their computational complexity with respect to the number of observations. Exact inference in GP regression is $\mathcal{O}(n^3)$ where n is the number of observations. For application where very a large number of observations is expected/required, this cost becomes prohibitive [24, 27].

The other models scale (nearly) linearly with the number of observations. This, however, comes with the very significant cost of sacrificing the natural way GPs provide uncertainty (variance) measure. To get this uncertainty measure with the other models, artificial methods are used, resulting in a poor variance estimate [24, 27].

For the application at hand, we don't expect a large number of observations. In fact, the aim is to reach satisfactory results with the lowest number of observations possible due to the high cost of each evaluation as discussed in section 1.2. Hence, the cubic complexity of GPs is of a little concern. On the other hand, the superiority of the uncertainty measures obtained by using GPs is highly desirable for our application as it results in a lower number of needed iterations (observations) to reach the maximum than any other method [5, 16], which makes Bayesian optimization with GPs an extremely sample-efficient approach.

Gaussian Processes

A Gaussian process $\mathcal{GP}(\mu_0, k)$ is a non-parametric model, completely specified by a mean function $\mu_0(\mathbf{x})$ and and a *kernel* (covariance function) $k(\mathbf{x}, \mathbf{x}')$. In most cases the mean is chosen to be a fixed function, generally equal to zero or a value inferred from the data [5]. The following definition of GPs is included for the sake of completeness and is mainly adapted from [24] but can also be found in many other books.

Consider any finite collection of n points $\mathbf{x}_{1:n}$, and define variables $f_i := f(\mathbf{x}_i)$ and $y_{1:n}$ to represent the unknown function values and noisy observations, respectively. In GP regression, we assume that $\mathbf{f} := f_{1:n}$ are jointly Gaussian and the observations $\mathbf{y} := y_{1:n}$ are normally distributed given $\mathbf{f}$. These assumptions result in the following model:

$$\mathbf{f} \mid \mathbf{x}_{1:n} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \tag{2.3}$$

$$\mathbf{y} \mid \mathbf{f}, \sigma^2 \sim \mathcal{N}(\mathbf{f}, \sigma^2 \mathbf{I}) \tag{2.4}$$

where the mean vector $\mathbf{m}$ and the positive semi-definite covariance matrix $\mathbf{K}$ are defined elementwisely as $m_i := \mu_0(\mathbf{x}_i)$ and $K_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$ respectively, and σ^2 is the given variance of the Gaussian noise. Equation (2.3) represents a prior distribution over the space of functions. A function can be drawn from this distribution by selecting n arbitrary points $\mathbf{x}_{1:n}$ and assigning each point of them a value drawn from a multivariate normal distribution parameterized by the mean function and the covariance matrix of all the selected points $\mathbf{x}_{1:n}$.

Now, say we received n (noisy) observations $\mathcal{D}_{1:n}$ of the target function. Then the posterior of the random variable $f(\mathbf{x})$ at an arbitrary point $\mathbf{x}$ given the observations $\mathcal{D}_{1:n}$ is normally dis-

tributed with the following mean and variance functions:

$$f(\mathbf{x}) \mid \mathscr{D}_{1:n} \sim \mathcal{N}(\mu_n(\mathbf{x}), \sigma_n^2(\mathbf{x})) \tag{2.5}$$

$$\mu_n(\mathbf{x}) = \mu_0(\mathbf{x}) + \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \mathbf{m}) \tag{2.6}$$

$$\sigma_n^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}(\mathbf{x}) \tag{2.7}$$

where $\mathbf{k}(\mathbf{x})$ represents a vector of covariance values between the arbitrary point $\mathbf{x}$ and the previous observations points $\mathbf{x}_{1:n}$, and $\mathbf{K}$ is the covariance matrix of these previous points.

Intuitively, the GP posterior in Eq. (2.5) is analogous to a function, but instead of returning a scalar $f(\mathbf{x})$ for an arbitrary $\mathbf{x}$, it returns the mean and variance of a normal distribution over the possible values of f at $\mathbf{x}$ [5]. Samples from a 1D GP prior and posterior are shown in Fig 2.3. The mean of the prediction can be looked at as the function with the highest probability given the observations.

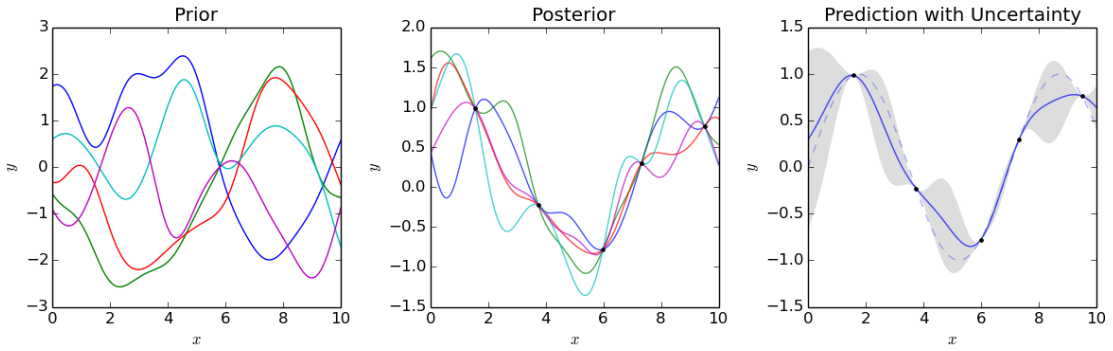


Figure 2.3: 1D GP with a zero mean and a squared exponential kernel. Right: five functions sampled from the prior as in Eq. (2.3), Middle: five functions sampled from the posterior as in Eq. (2.5) after obtaining five observations. Right: the sampled function (dashed) as well as the mean (solid line) and the standard deviation (shaded) of the prediction. (By Cdipaolo96 [CC BY-SA 4.0], from Wikimedia Commons)

2.3 Initial Knowledge

One of the very important features of Bayesian optimization with GPs is its ability to incorporate and take advantage of the provided initial knowledge. Providing a meaningful initial knowledge is crucial for a sample-efficient optimization. Initial knowledge can be provided through three main ways: selecting the mean function μ_0, selecting a suitable kernel k, and providing relevant initial w observations $\mathscr{D}_{1:w}$.

As mentioned in the previous section, the mean function is usually selected as a constant function equal to zero. This is both convenient and reasonable in black-box optimization [5], since selecting a variable mean function can only be justified by a detailed knowledge of the target function that is naturally unavailable. Hence, we are left with the two other ways.

2.3.1 Kernels

Kernels or covariance functions are simply functions that provide a similarity measurement between two points in the target function domain. The selection of the kernel function determines the class of functions the target function is expected to belong to. For example, if the target function is expected to be polynomial, a polynomial kernel should be used, if the target function is expected to be periodic, a periodic kernel should be used... etc [9].

Fig 2.4 shows samples from a 1D GP prior with three different kernels. The figure shows how strong is the influence of the kernel on the shape of the predicted functions and hence on the acquisitions functions (discussed next section). In fact, the authors of [24] take the perspective,

based on experience and literature, that careful choice of the underlying surrogate model (controlled by the kernel) is more significant than the choice of the acquisition function (discussed in section 2.4).

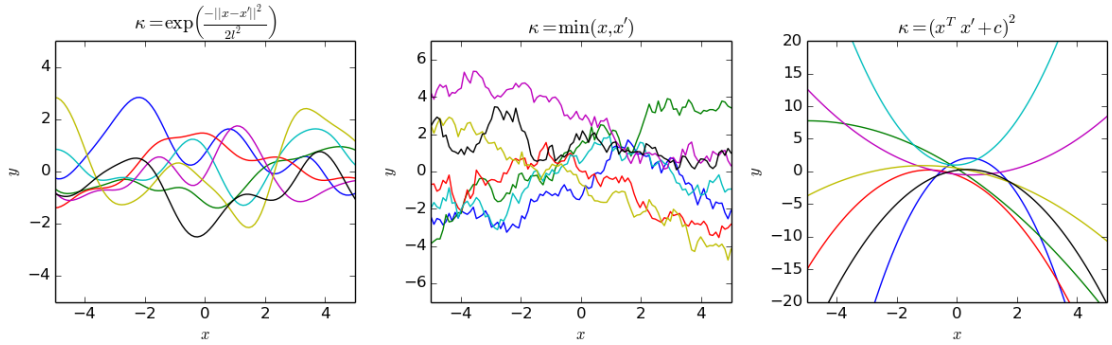


Figure 2.4: Samples from 1D GP prior with a zero mean and three different kernels. Left: Isotropic RBF. Middle: Brownian. Right: quadratic. (By Cdipaolo96 [CC BY-SA 4.0], from Wikimedia Commons)

Since no specific assumption can be made about the black-box function of our application (like being periodic or polynomial), we stick to a commonly used general-purpose kernel known as the squared exponential kernel, also known as the Radial-basis function (RBF) kernel and expressed as:

$$k_{RBF}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}[(\mathbf{x}-\mathbf{x}')^T \boldsymbol{\theta} (\mathbf{x}-\mathbf{x}')]\right) \tag{2.8}$$

where $\boldsymbol{\theta}$ is a diagonal matrix of $\dim(\mathbf{x})$ squared length scales θ_i^2. The kernel can be isotropic by setting all θ_i^2 to the same value making a variation along any dimension have the same effect. Otherwise it is anisotropic and a θ_i^2 with low value will make variation along dimension i of low or no significance. In both cases, this kernel is stationary (transition-invariant).

Through the choice of hyperparameters $\boldsymbol{\theta}$, one can control how rapid are the changes in the functions predicted by the GP prior as seen in Fig. 2.5. Typically, the values of the hyperparameters are learned using the observations that were obtained so far through seeding random values and maximizing the marginal likelihood of the fitted GP model [5, 9].

The squared exponential kernel is a special case of a broad class of kernels named Matern Kernels and parameterized by the smoothness parameter $\nu > 0$. Samples from GPs with these kernels are differentiable $\nu - 1$ times [5, 24]. The Matern kernel converges to RBF when ν goes to infinity, which simply means that using RBF implicitly assumes that the target function is infinitely differentiable.

Although this is a strong assumption that cannot be safely placed in most applications, the kernel is still widely used as this assumption is unlikely to cause problems unless many samples are taken around an area with discontinuity in the target function. If this happened, it can be detected and then the kernel should be changed [9]. On the other hand, the RBF kernel has the advantage of being computationally simpler than all other variants of the Matern kernel class.

2.3.2 Initial Observations

Normally, before starting Bayesian optimization iterations, a number of initial observations are provided to form the initial surrogate model as in the first line of Algorithm 1. In this step, the user has the ability to actively select the locations for sampling these observations.

The user can then use these initial observations to provide *hints* for Bayesian optimization about the regions with high chances of having the maximum. Combined with local improvement acquisition strategies (see next section), this technique is very effective for higher dimensional target functions. It was successfully employed to learn six robot grasping task param-

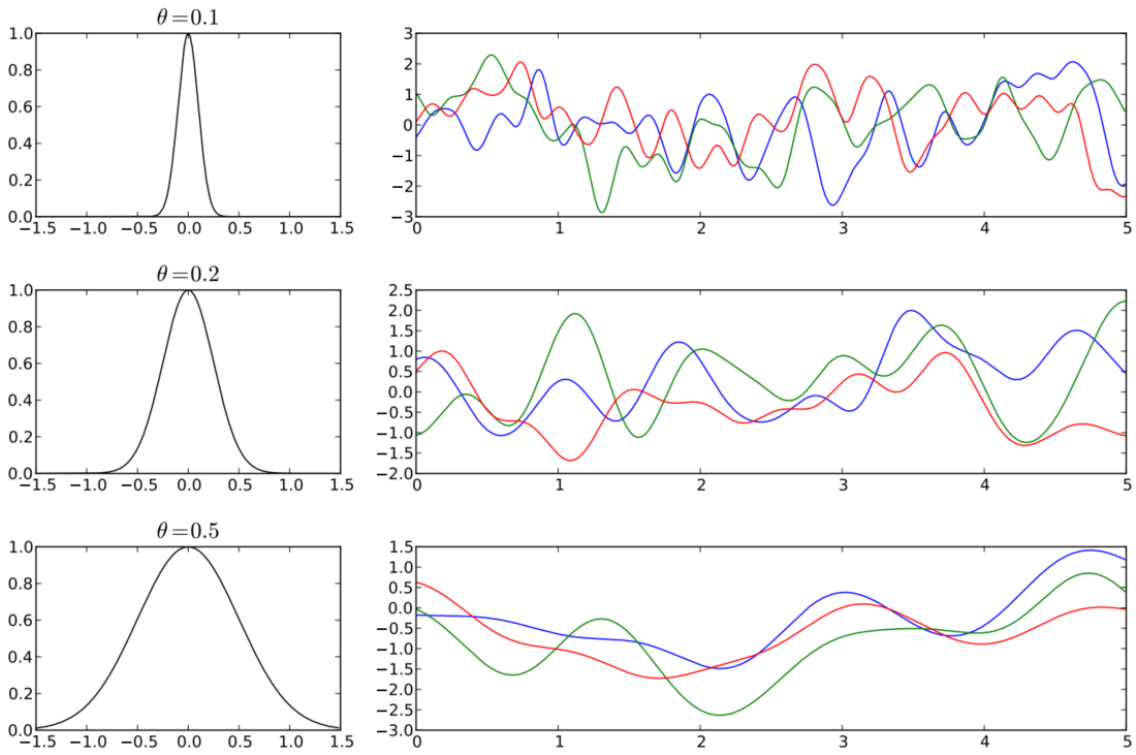


Figure 2.5: Samples from 1D GP priors with isotropic RBF kernel and $\theta = 0.1, 0.2, 0.5$. Left: the function $k(0, \mathbf{x})$. Right: the corresponding samples from the GP priors. (taken from [5]).

eters by providing 25 initial observations via successful demonstrated grasps (i.e. Kinesthetic learning) [16].

2.4 Acquisition Functions

The acquisition function is another component of Bayesian optimization where many design choices are available. The role of acquisition functions is to balance exploration and exploitation by providing a measure for the utility of obtaining a new observation at the point $\mathbf{x}_{n+1}$ based on the surrogate model built using the past observations $\mathscr{D}_{1:n}$. The next point to sample is then obtained by optimizing the acquisition function:

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\chi} \alpha(\mathbf{x} \mid \mathscr{D}_{1:n}) \tag{2.9}$$

The idea of Bayesian optimization is that, instead of optimizing an expensive black-box function, a function that is relatively cheaper is optimized. In other words, the difficult original optimization problem is solved by tackling an easier secondary optimization problem. Still, optimizing the acquisition function is not an easy task, especially in high dimensional spaces. Techniques used in literature to do this optimization range from random sampling, to discretization and adaptive grids. Multistarted quasi-Newton hill-climbing approaches can also be used when the gradient of the surrogate model is available or can be efficiently approximated [24]. Regardless of the taken approach, it is very difficult to ensure that the global maximum of the acquisition function was really found [24].

Different acquisition functions lead to different exploration/exploitation trade-offs and behaviors that are suitable for certain applications but not suitable for others. We concentrate here on the three most widely used functions in literature. The shapes of these functions over a simple 1D GP posterior are shown in Fig 2.6 for several values for their hyperparameters. Each one of the three is then explained briefly.

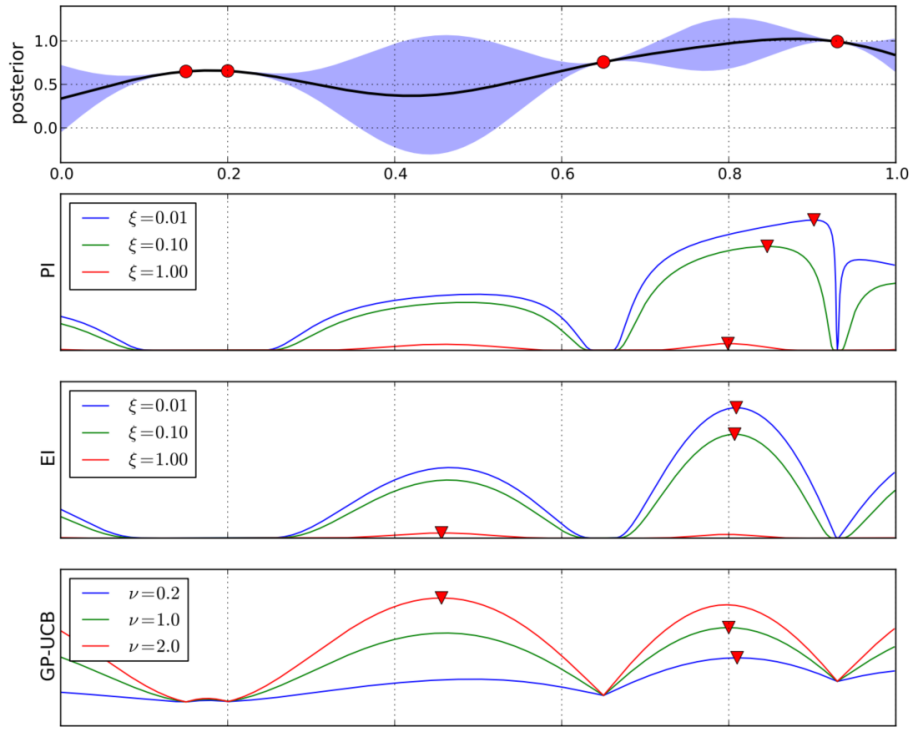


Figure 2.6: The shapes of three acquisition functions over a simple 1D GP posterior. (taken from [5])

As part of the flexibility of the Bayesian optimization framework, it is not necessary to stick to one hyperparameter value for the selected acquisition function. The value can be scheduled to change from one iteration to another. One of the popular schedules is to start with values that encourage exploration and then gradually move to a more exploitative behavior [5].

In fact, it is not even necessary to stick to one acquisition function throughout the optimization. Several techniques have been suggested to alter and mix between them [12].

2.4.1 Probability of Improvement (PI)

This function assesses, for a certain point $\mathbf{x}$, the probability of finding a target function value (or reward) that is higher than the maximum found observation by the hyperparameter ξ:

$$PI(\mathbf{x}) = P(f(\mathbf{x}) \geq \max(y_{1:n}) + \xi) = \Phi\left(\frac{\mu_n(\mathbf{x}) - \max(y_{1:n}) - \xi}{\sigma_n(\mathbf{x})}\right) \tag{2.10}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function.

Maximizing the PI function results in selecting points most likely to offer an improvement of at least ξ. This allows ξ to affect how local/global the search is. If ξ is very small, Bayesian optimization will be highly local and will not move to points far from the obtained observation points before exhaustively investigating the area around the best observation so far. On the other hand, choosing a large value for ξ will encourage more exploration and will prevent Bayesian optimization from *fine-tuning* [5]. So, a middle point should be found.

Note that, because the fitted Gaussian process defines, at each point $\mathbf{x}$, a normal distribution $P(f(\mathbf{x}))$ over the value of $f(\mathbf{x})$ at this point, Eq. 2.10 can be evaluated for any value of ξ.

2.4.2 Expected Improvement (EI)

This function assesses not only the probability of improvement when sampling at a certain point $\mathbf{x}$, but also the magnitude of this improvement.

$$EI(\mathbf{x}) = (\mu_n(\mathbf{x}) - \max(\mathbf{y}_{1:n}) - \xi)\Phi(Z) + \sigma_n(\mathbf{x})\phi(Z) \tag{2.11}$$

where $\sigma_n(\mathbf{x}) > 0$ (otherwise the function vanishes to zero), $\phi(\cdot)$ is the standard normal probability density function and

$$Z = \frac{\mu_n(\mathbf{x}) - \max(\mathbf{y}_{1:n}) - \xi}{\sigma_n(\mathbf{x})}$$

The full derivation of this function can be found in [24]. Here also the hyperparameter ξ has a similar role as the one it has in PI: controlling the locality of the search.

2.4.3 Upper Confidence Bound (UCB)

This function simply gives higher values for points where the mean of the model is high as well as the uncertainty:

$$UCB(\mathbf{x}) = \mu_n(\mathbf{x}) + \kappa\sigma_n(\mathbf{x}) \tag{2.12}$$

The hyperparameter κ offers a direct trade-off between exploration and exploitation. High values of κ result in exploring by selecting regions with high uncertainty. Low values of κ results in exploiting by sticking to the regions where the mean is high.

2.5 Conclusion

After touching on the different parts of Bayesian optimization, it is important to relate back to our application of finding the optimal impedance parameters for an interactive drone. The design choices and assumptions done in this chapter are summarized here.

Considering the assumptions required by Bayesian optimization on the target black-box function, it can be realized that they are not difficult to satisfy. First, the user has to provide the domain of the function. The simplest way of expressing the domain is as a hyper-rectangle in the space of impedance parameters values. In other word, the lower and upper bounds of each impedance parameter. Knowing that impedance parameters cannot take negative values as they result in controller's instability, the lower bound is automatically provided. The upper bound is a little harder to determine as it is controller- and robot-specific. Based on the capabilities of the drone, the controller cannot be set on arbitrarily high gains.

The second assumption is that, within the given domain, the black-box reward function (performance measure) can be sampled at any location. Extra caution has to be taken concerning this assumption since, as mentioned in section 1.2, the parameter space of the impedance controller is not completely safe to explore and some locations within it lead to unstable behaviors. That is why the proposed learning framework in Fig 1.3 (section 1.3) contains a failure detection block. The job of this block is to early detect if a certain selection led to instability, in which case it should immediately stop the current training episode, set the impedance parameters on some given safe default values, and send to Bayesian optimization a small reward (or a largely negative one) indicating that the performance was highly undesirable.

For the probabilistic surrogate model of Bayesian optimization, Gaussian processes were selected for their ability to naturally provide uncertainty measure over the whole target function domain allowing more effective exploration/exploitation trade-offs through acquisition functions and hence increasing the sample-efficiency of the technique.

The importance of providing meaningful initial knowledge was emphasized as it significantly affects the sample-efficiency of Bayesian optimization. Initial knowledge can be provided through the initial observations provided at the start of Bayesian optimization to build the initial surrogate model. Carefully selecting the locations of these initial observations, the user can give hints to the learning algorithm that can make the task of finding high rewards easier.

Another very important way of providing initial knowledge is through the selection of a suitable kernel function. It was mentioned that, since there is no assumption that can be made over the black-box reward function encountered in the application at hand, the widely used

RBF kernel is selected. Although this function builds a surrogate model that is infinitely differentiable (assuming that the target function is also infinitely differentiable), this is unlikely to cause problems. It is important here to remember that the purpose of building the surrogate model is not to resemble the target function. The only important thing in the surrogate model is that the position of its global maximum converges to the position of the global maximum of the target function.

Regarding the selection of kernel hyperparameters values, it was explained that an isotropic kernel is used since we have no reason to believe that variation among one of the learned impedance parameters is more important than variations among the others. The exact values of the kernel hyperparameters are usually continuously adapted and inferred from the data at each Bayesian optimization iteration.

Finally, the role of acquisition functions in Bayesian optimization was discussed as providing ways to balance explorations and exploitation. It was pointed out that it is not necessary to stick to one acquisition function or one hyperparameter settings throughout Bayesian optimization. In order to prevent the design space from exploding, however, we use just one acquisition function set on one hyperparameter value throughout the optimization. The suitable acquisition function and hyperparameter value for our application are investigated in chapter 3.

3 Bayesian Optimization Empirical Studies

In this chapter, Bayesian optimization is studied thoroughly by applying it to simulated black-box functions of different numbers of dimensions. The concentration is on the performance and behavior of the different acquisition functions and the effects of varying their hyperparameters. The aim is to find settings that provide sample-efficiency and can scale to higher dimensions.

It was not possible to do this study directly on a simulated drone where the target function of Bayesian optimization is a black-box reward function calculated for a simulated impedance controller. This is because doing this study in a complete simulated environment would in fact take very long time for all the needed experiments to be finished (weeks of processing time). Additionally, this will make it difficult to generate various black-box functions as to generate a different black-box function, a new environment or a different task is needed.

To get around this, a software was written to generate random functions at any number of dimensions. The aim is not to produce functions that resemble the actual black-box reward functions encountered when working with the impedance controller of a drone, because this is practically impossible (otherwise, they are not black-box functions). Instead, the aim is to have a test bench that offers a relatively fast way of testing the different settings and seeing the *relative* differences between them and across different numbers of dimensions.

For easier navigation, Fig. 3.1 shows a roadmap of the chapter. The chapter starts with section 3.1 explaining the principles behind the software that generates random functions. After that, section 3.2 presents the setup and conditions ensured throughout the experiments. It also discusses the important measures that should be monitored to assess the performance of Bayesian optimization. Then, section 3.3 presents and discusses the results of a parameter study over different acquisition functions in different dimensions. The insights gained from these results were used to propose a novel way of optimizing acquisition functions. The new method is explained in section 3.4 along with results showing its advantages. Section 3.5 (can be skipped) investigates an alternative automatic way of providing initial observations. Finally, the chapter is concluded with the main take-home lessons in section 3.6.

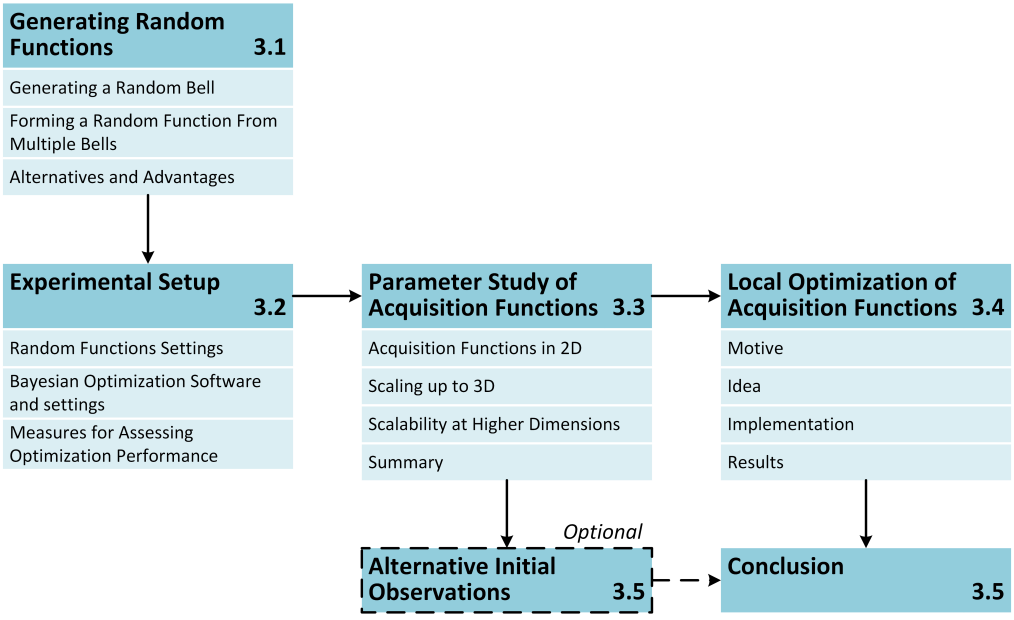


Figure 3.1: A roadmap for the chapter. Arrows indicate prerequisites.

3.1 Generating Random Functions

As there are significant well-documented software contributions to Bayesian optimization [10, 4] and Gaussian processes [25], the idea was to generate random functions that represent an intuitively similar situation to the one that will be faced when dealing with the real black-box reward function. Then by applying Bayesian optimization on these functions, one can gain insights into various elements of the technique, like the acquisition functions and their hyperparameters, the scalability to higher dimensions, the effect of the lack or presence of meaningful initial observations... etc.

The problem is, no ready solutions are available to do this task satisfactorily. So, a new module was written to do the job. The module is called `RandomFunction` and is written in Python. It will be released on `GitHub` as an open-source contribution to the research community. I explain here the main concepts of the used techniques, but refrain from discussing the code detailed implementation and leave that to the documentation included with the code.

The goal of the module is to produce a random function S in any dimension d given the domain (denoted χ) of the function as a list of closed intervals that form box boundaries in $\mathbb{R}^d$, and the range (denoted $\mathscr{R}$) as a closed interval.

3.1.1 Generating a Random Bell

Random functions are generated at any dimension with the help of the probability density function (PDF) of the multi-variate normal distribution. This function has the familiar *bell* shape and is parameterized by the mean vector (determining the position of the center of the bell in $\mathbb{R}^d$) and the covariance matrix (controlling the shape of the bell). It was chosen because it can be easily generated in any dimension. The module can generate a random bell b_i:

$$b_i(\mathbf{x}) = \begin{cases} \frac{1}{f(\boldsymbol{\mu}_i|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)} f(\mathbf{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) & \text{if } \mathbf{x} \in \chi, \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where f is the PDF of the multivariate normal distribution, $\boldsymbol{\mu}_i$ is the mean of the distribution chosen uniformly randomly from domain χ and $\boldsymbol{\Sigma}_i$ is a randomly generated positive semi-definite convenience matrix with its elements kept within a range given by the user. Note that the PDF is normalized by dividing over its maximum (its value at the position of the mean).

Generating a random positive semi-definite matrix within a range of values given by the user was a challenging task. The current implementation relies on the fact that any matrix $\mathbf{M} = \mathbf{Q}^T\mathbf{D}\mathbf{Q}$ is a positive semi-definite matrix provided that $\mathbf{Q}$ is a full-rank matrix and $\mathbf{D}$ is a positive diagonal matrix. By selecting $\mathbf{Q}$ to be random full-rank matrix with its diagonal elements equal to zero, and selecting $\mathbf{D}$ as a random diagonal matrix, one can have separate control on the range of values of the diagonal and off-diagonal elements of the covariance matrix, resulting on maximum control over the shape of the generated bells. Figure 3.2 shows ten 2D randomly generated bells. For these bells, $\chi = \{[0,1],[0,1]\}$, and the covariance diagonal range is $[0.01, 0.05]$.

3.1.2 Forming a Random Function With Multiple Bells

Random functions of various shapes can be formed by combining u (an integer given by the user) randomly generated bells. The module provides two ways of combining the bells. Both of them are illustrated on a simple 1D example in Fig 3.3.

The first way creates what we call a *rough* function S_{rough}:

$$S_{\text{rough}}(\mathbf{x}) = \frac{h_i}{\max(h_{1:u})} \max(\{b_i(\mathbf{x})\}_{i=1}^{u}) \tag{3.2}$$

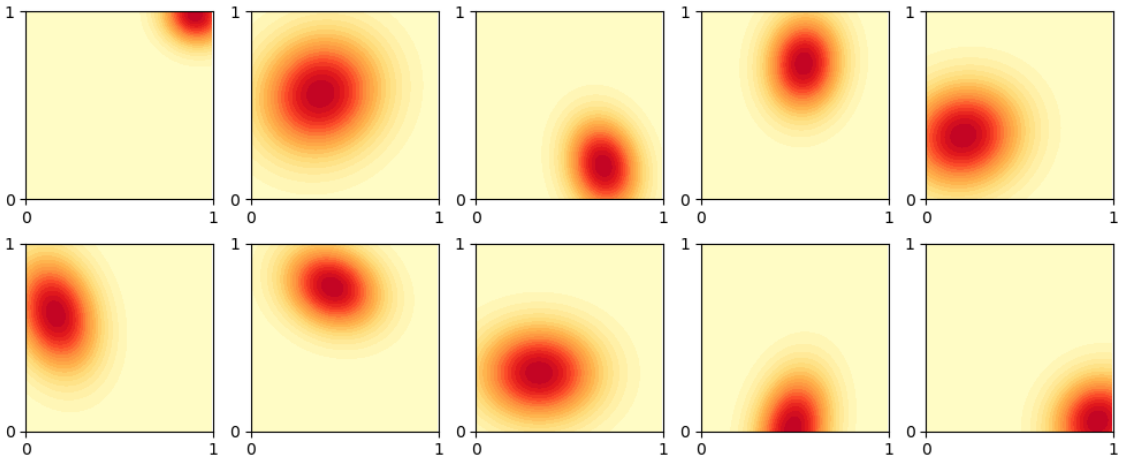
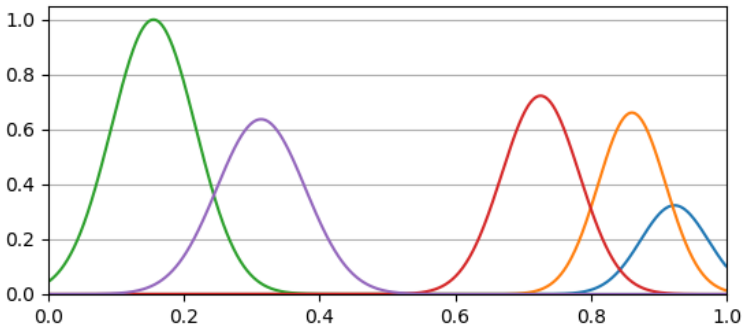
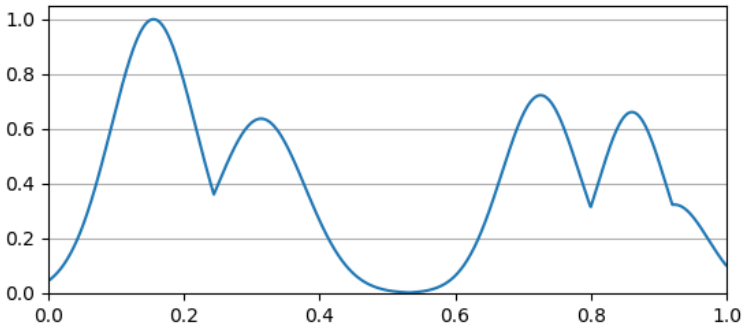


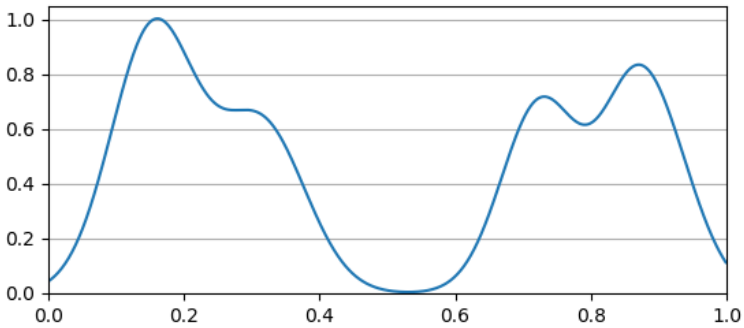
Figure 3.2: Contours of 2D randomly generated bells



(a) Individual five randomly generated bells



(b) The bells combined to generate a rough function



(c) The bells combined to generate a smooth function

Figure 3.3: A 1D example of how a function can be created using individual bells.

where h_i is a scaling height selected uniformly randomly within $[0,1]$. The function is normalized by dividing over the maximum height. One of the nice things about this function is that it is easy to locate its maximum, which is simply the mean of the bell associated with the maximum height $\max(h_{1:u})$. On the other hand, the first derivative of the resulting function is discontinuous at positions where two bells intersect as seen in Fig 3.3b.

The second way of forming a function creates a *smooth* function S_{smooth}:

$$S_{\text{smooth}}(\mathbf{x}) = \frac{1}{\sum_{i=1}^{u} h_i b_i(\boldsymbol{\mu}_m)} \sum_{i=1}^{u} h_i b_i(\mathbf{x}) \tag{3.3}$$

where m is the index of the bell corresponding to the maximum height $m = \arg\max_{i \in [1:u]} h_i$. The nice feature of smooth functions is that they are infinitely differentiable. However, the position of the maximum cannot be obtained analytically. It is not necessarily at the mean of the bell with the maximum height $\boldsymbol{\mu}_m$ since the maximum of the sum of two Gaussians might be, in fact, somewhere between their means if they are close enough. With that in mind, the function in (3.3) is not completely normalized, but, intuitively, given that no other bell with the same height is close to the bell with the maximum height, the function maximum is very close to unity, and the position of the maximum is very close to $\boldsymbol{\mu}_m$. Providing bounds on the distance between $\boldsymbol{\mu}_m$ and the global maximum as well as necessary conditions for these bounds requires a lengthy mathematical analysis and is beyond the scope of this report.

Notice that the generated (nearly) normalized functions can be easily scaled to match the function range $\mathscr{R}$ given by the user.

One might intuitively think that the number u of the bells that form a function determines the number of local maxima in that function. Looking to Fig 3.3, it can be seen that this is not true. The rough function constituted from five bells has only four local maxima, while the corresponding smooth one has only three. However, if the domain is large and the covariance of the randomly generated bells is chosen to be small, then controlling u directly determines the sparsity of the maxima over the function domain.

3.1.3 Alternatives and Advantages

A random function can also be generated at any number of dimensions by sampling it from a Gaussian process prior. This is because Gaussian processes provide probability distributions over the space of functions in $\mathbb{R}^d$ as seen in the previous chapter. However, one of the disadvantages of this method compared to the method proposed here is that the position of the maximum cannot be easily located. Being able to locate the maximum of the generated function is a key feature to allow clearer understanding of the behavior of Bayesian optimization when applied to that function.

More significantly, generating a random function by using a Gaussian process would make the Bayesian optimization experiments biased especially if the used Kernel in the GP used to generate the function is the same as the one used in building the surrogate model in Bayesian optimization. Comparing equations (3.2) and (3.3) of the rough and smooth functions with the the equation of the GP prior (2.3), and also the shapes of the generated functions (Fig. 3.3) with those of the GP priors with the RBF kernel (Fig. 2.5), it can be clearly seen that the generated functions according to the described methods are totally different than those generated by GP priors. One of the clear differences is that, compared to GP priors, the generated random functions characteristics are not controlled by a kernel at all and so can be varying rapidly in one area but varying very slowly in another one. Hence, the proposed method does not suffer from this bias problem.

In addition to forming rough/smooth functions and controlling various aspects about them, the written software also has multiple graphing capabilities, allowing the user to graph 1D

graphs, 2D contours or 3D surfaces for the generated random function or a cross-section of it (if $d > 2$).

3.2 Experimental Setup

To ensure the meaningfulness of the Bayesian optimization simulations performed on the randomly generated functions, some initial conditions and settings were kept throughout generating all the results seen in next suctions.

3.2.1 Random Functions Settings

All the experiments shown in this chapter were done over ten[1] randomly generated functions per number of dimensions. The group of functions were generated once and kept throughout the experiments to produce comparable results. The measurements taken during these experiments were averaged over the ten functions. Figure 3.4 shows the ten 2D randomly generated functions used in the experiments.

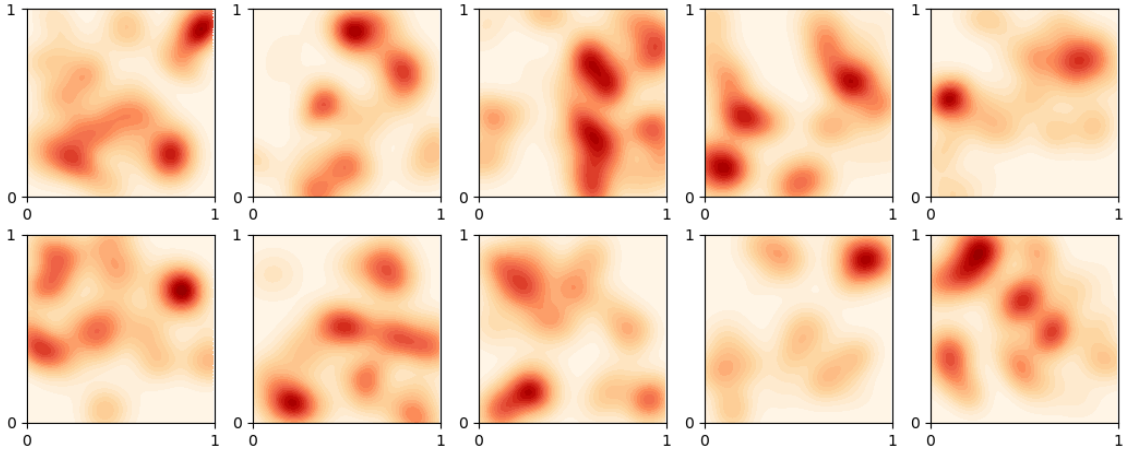


Figure 3.4: Contours of ten 2D randomly generated smooth functions. Darker areas have higher values (rewards).

All the functions were generated with a domain that forms a unit hyper-cube in $\mathbb{R}^d$. The range of the functions was kept $[0,1]$. Because sooth functions cannot be completely normalized since the position of the maximum cannot be determined analytically as mention in section 3.1, it is not possible to ensure that the maximum of the generated smooth functions is one. In fact, by doing a grid search (0.001 precise) on the ten used 2D functions, it was found that the average of their global maxima is 1.04. All the experiments were performed on smooth functions unless otherwise mentioned. This is because, for our application that involves working with dynamic robotic systems, we do not expect the black-box reward function that will be optimized to have a discontinuous derivative.

The number u of bells per function was selected to be 30 for all numbers of dimensions. The range of the values of the diagonal elements of the covariance matrix was also selected to be the same for the functions generated in all numbers of dimensions. This is to ensure similar variations among the functions generated in different numbers of dimensions. This range was selected to be $[0.005, 0.01]$. This generates small bells to allow functions with various complex

[1] Doing an experiment for 10 functions is already computationally expensive. It can take up to 40 minutes to finish 50 Bayesian optimization iterations for 10 functions for just one setting. There were trials to utilize machines with more computational power in the RaM lab, but, given the tight time frame, it was difficult to make the experiments run smoothly on these machines and the time needed to finish one experiment did not improve. It can be seen from the generated results, however, that performing the experiments over only ten functions per number of dimensions was enough to show clear trends.

shapes as in Fig 3.4. It is easy to imagine functions with similar shapes being encountered when optimizing the reward function of the drone's interaction controller.

3.2.2 Bayesian Optimization Software and Settings

At an early stage of the research, I implemented Bayesian optimization following the algorithm described in [16]. The implementation contained a simple Gaussian process inference with the squared exponential kernel. The UCB acquisition function was used and was optimized using gradient decent with a variable learning rate. The implementation was intended to understand the technique and explore its various elements. However, expectedly, readily available open-source packages offer more efficiency and flexibility.

Two packages were investigated and the decision at the end was in favor of using `BayesianOptimization` [10] over the older `GPyOpt` [4]. What makes the selected package appealing is the clarity and simplicity of its implementation and documentation, allowing easy modification and development in the different parts of the Bayesian optimization framework. This feature was exploited when introducing a novel method for optimizing acquisition functions as will be seen in section 3.4.

In addition, the package is built over `scikit-learn`'s Gaussian processes modules [25], which offers a wide range of kernels and, more importantly, the possibility to optimize the kernel hyperparameters through multiple-seeded maximization of the marginal likelihood of the obtained models (as mentioned in section 2.3.1).

All the experiments shown in this chapter are done with ten random initial observations to generate the initial model (line 1 of Algorithm 1). For each randomly generated function, ten random points are chosen uniformly randomly within the domain of the function and are maintained to be used in all experiments where this random function is used. This ensures that results of experiments done for different settings are comparable as the initialization is always the same. After these ten initial observations, Bayesian optimization runs for another forty iterations, meaning that the function will be propped for a total of fifty times. This is done because the interest is in finding sample-efficient techniques.

As decided in section 2.3.1, the RBF kernel is used. During each iteration of Bayesian optimization, the `scikit-learn`'s Gaussian process module is set to optimize the kernel hyperparameter $\boldsymbol{\theta}$ when building a posterior over the observations received so far (line 3 of Algorithm 1). The module is set to randomly seed the optimization 25 times to find the best value for the hyperparameter. This is of great help to the user of Bayesian optimization as it takes away the burden to set these very significant hyperparameters.

As we expect no noise in these randomly generated functions, the σ^2 hyperparameter seen in equations (2.4), (2.6) and (2.7) was set to 10^{-5}. According to the documentation of the `scikit-learn` package [25], even in the presence of no noise, it is useful to set this parameter to a value slightly greater than zero to ease some numerical issues faced during Gaussian process fitting. The value of the hyperparameter should be increased depending on the anticipated noise in the observations of the real black-box function.

The `BayesianOptimization` package contains the three acquisition functions discussed in section 2.4. The selected acquisition function is optimized every iteration (line 4 of Algorithm 1). The default way this optimization is done in the package is that the acquisition function is randomly sampled at 10^5 different points. The found point that gives the maximum is then saved and a local quasi-Newton optimization (L-BFGS-B) is initiated at 250 random points in the function domain. The maximum obtained by these local optimizations is compared to the random sampling maximum and the best is selected.

3.2.3 Measures for Assessing Optimization Performance

From the author's point of view, three measures are of special importance to the understanding of Bayesian optimization performance:

- y_n^+: the maximum obtained reward so far

$$y_n^+ = \max y_{1:n} \tag{3.4}$$

- $\bar{y}_n$: the average of all obtained rewards so far

$$\bar{y}_n = \sum_{i=1}^{n} y_i / n \tag{3.5}$$

- d_n: the Euclidean distance between the position of the current observation and the closest past observation

$$d_n = \min\{\sqrt{\mathbf{x}_n - \mathbf{x}_i}\}_{i=1}^{n-1} \tag{3.6}$$

The graph of y_n^+ corresponds to the learning rate graph in classical reinforcement learning literature. It shows how fast the optimization found the position of the maximum. If y_n^+ is increasing with the number of iterations n, then a new maximum is found every iteration, else, if it stays constant, then no new maximum has been found since the last increase (positive slope). Clearly, y_n^+ cannot decrease with time.

Looking at the curve of $\bar{y}_n$ gives an indication if the optimization method is constantly trying at areas of low reward, which will make $\bar{y}_n$ decrease with time. In other words, $\bar{y}_n$ can be used as a safety measure of the learning considering that falling to areas of low rewards, when dealing with the impedance controller's black-box reward function, corresponds to bad or dangerous performance during performing the predetermined task.

Finally, d_n can be used to see how local/global the search is. If d_n is high for a number of iterations, then the search is constantly looking to positions that are far from already seen observations.

It is important to emphasize that these measures, shown in the coming figures, are averaged over experiments performed for ten different randomly generated functions as mentioned in section 3.2.1.

3.3 Parameter Study of Acquisition Functions

In this section, an empirical parameter study is done over the three acquisition functions discussed in section 2.4. Two main questions are of special importance:

1. How does changing the hyperparameters of the acquisition functions affect the learning performance and the exploration/exploitation trade-off?

2. how do acquisition functions settings scale with higher number of dimensions?

To isolate the effect of changing the hyperparameters values, it is studied and compared over 2D functions in subsection 3.3.1. To make sure that the observations and conclusions reached in 2D still apply as the number of dimensions increases, the same experiments are repeated for 3D functions in subsection 3.3.2. Then, to investigate more the effect of increasing the number of dimensions of the learning problem, the experiments results are presented and compared over four different numbers of dimensions in subsection 3.3.3.

Remark All figures in this section are best seen in colors.

3.3.1 Acquisition Functions in 2D

PI Results

Figure 3.5 shows the three measures mentioned above for the PI acquisition function with a range of values for its hyperparameter ξ over ten randomly generated 2D functions. Looking at the top y_n^+ graph, it can be seen that for high values of ξ (more exploration) the maximum value reached by the end of the 50 observations is low with respect to values reached by other settings. This is due to the fact that more global searches cannot properly fine-tune to reach the position of the maximum. On the other hand, very small values of ξ (0.001 on the extreme) slow down the search. It can be noticed that a middle value of 0.01 produces the best result in this case, with a curve that quickly reaches a high value.

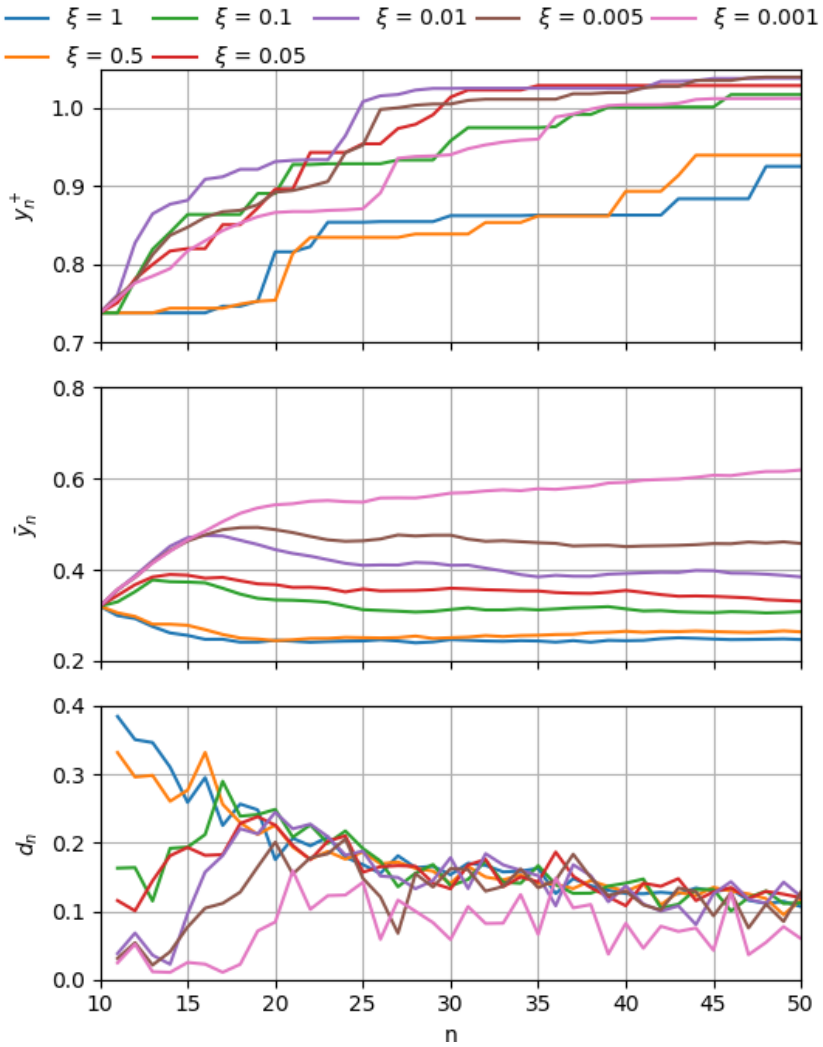


Figure 3.5: Performance of PI with different values for ξ over 2D functions.

Observing the middle $\bar{y}_n$ graph, another interesting trend can be noticed. Bayesian optimization with more local acquisition function (low values of ξ) accumulates a greater average reward over time even though it might have not necessarily reached the postilion of the maximum reward. This is understandable, since more local search have a lower risk of falling into areas of smaller rewards than what was already obtained.

Finally, investigating the d_n curves shows that, at the start of Bayesian optimization (after 10 initial observation), searches with smaller values of ξ remain closer to the already seen observations. However, as n increase, d_n does not converge to zero even after already finding points

with very high rewards. Instead, at every iteration, the maximum of the acquisition function with nearly all values of ξ is about 0.1 away from the closest seen observation.

EI Results

Figure 3.6 shows the performance of the EI function. All of the observations mentioned for the PI function can be repeated here with few exceptions. While higher values of ξ still leads to bad y_n^+ curves, performance improves as ξ decreases. In fact, the best y_n^+ curve is that of the lowest value of ξ. However, it should be noted that the y_n^+ curves of EI need more iterations to start going up than those of PI. It is also clear that the $\bar{y}_n$ curves of PI are generally much higher than those of EI. On the other hand, the d_n curves of both functions are very similar.

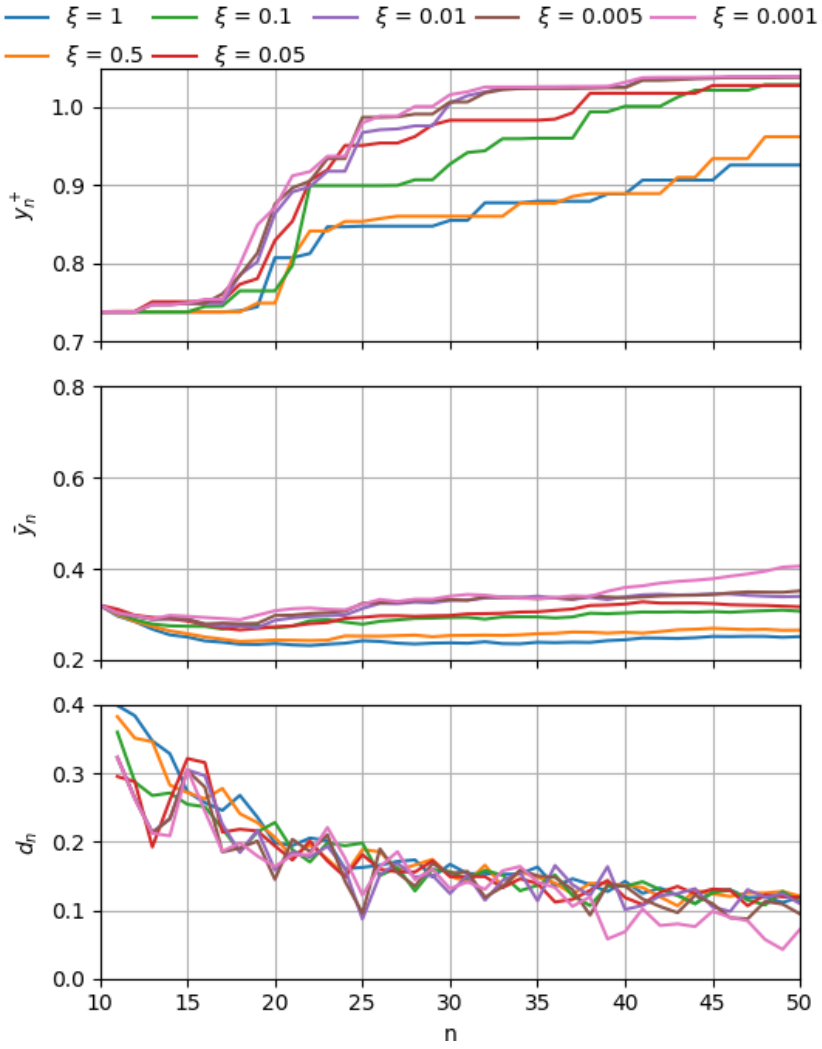


Figure 3.6: Performance of EI with different values for ξ over 2D functions

UCB Results

In Fig. 3.7, the performance of Bayesian optimization with the UCB function is shown. It can be seen that for the higher values of κ, y_n^+ increases slowly with time. For the lowest values of κ, however, the search quickly converges to a local maximum and starts exploiting it (hence the horizontal line at the end).

It can be noticed, like in the PI and EI cases, the $\bar{y}_n$ curves are higher for lower values of the hyperparameter κ. However, the $\bar{y}_n$ curves of UCB are, in general, higher than those of PI and EI.

For the d_n curves, similar to PI and EI, UCB with lower values of κ starts exploring locations closer to the already seen observations. Unlike PI and EI, on the other hand, as the number of iterations increases, the d_n gradually decreases reaching around zero at the end for all values of κ.

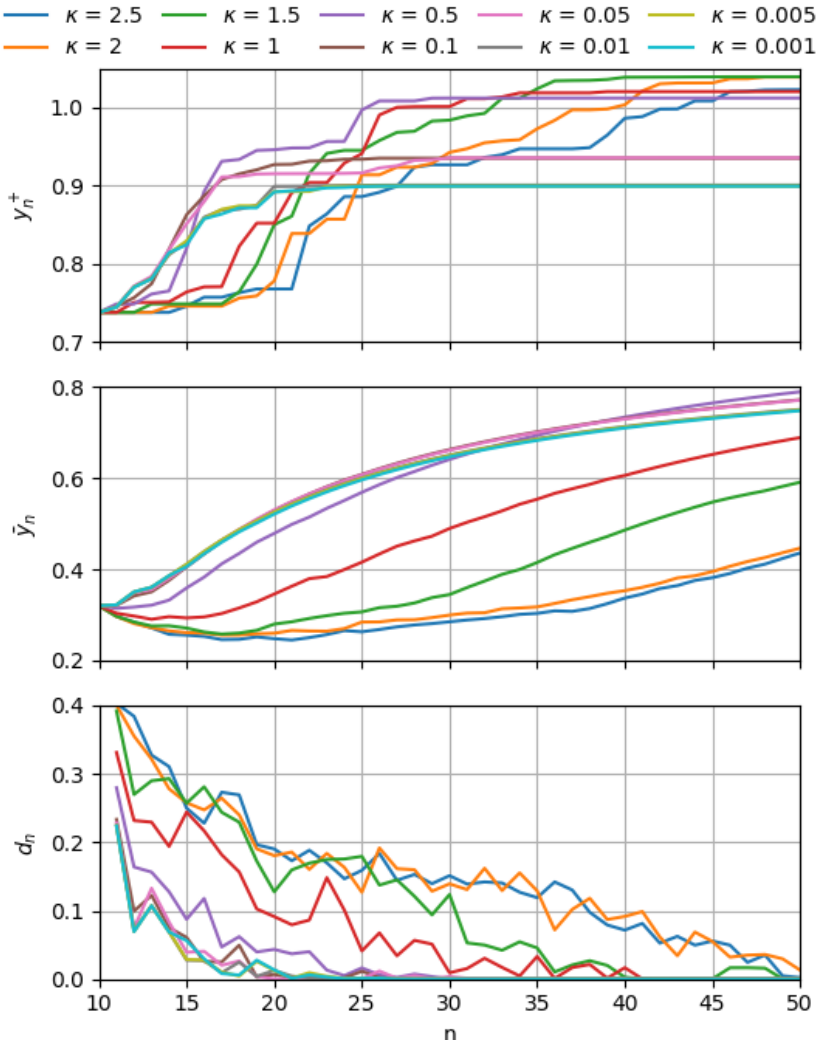


Figure 3.7: Performance of UCB with different values for κ over 2D functions

Discussion

Considering the shown results, it should be emphasized that the objective of Bayesian optimization is not to maximize $\bar{y}_n$, but y_n^+. So a method that quickly finds higher y_n^+ while getting a lower $\bar{y}_n$ (because it occasionally falls into areas of smaller rewards while exploring) is favorable (observe the y_n^+ and $\bar{y}_n$ curves for PI with $\xi = 0.01$). However, methods that can do both tasks are very desirable, especially for a cost- and safety-critical system like the dynamic robotic system we are dealing with here. Finding a Bayesian optimization configuration that can maintain an increasing y_n^+ and $\bar{y}_n$ during the learning, actually means that the robot (drone) will be less probable to fall into areas of bad performance or, worse, instability.

Noticing the d_n curves of PI and EI opens the eye to a very important characteristic of improvement-based acquisition functions (PI and EI). As long as there is uncertainty in some areas of the surrogate model, exploration will continue with these acquisition functions because there is a (even very slight) probability of improvement at these areas of uncertainty. Hence d_n never reaches zero. This is also one of the reasons why the $\bar{y}_n$ curves of PI and EI generally lower than

those of UCB as continuous exploration leads to higher chances of occasionally getting lower rewards.

It is interesting to see that, unlike PI and EI, for lowest values of its hyperparameter κ, UCB doesn't fine-tune and reach higher y_n^+. Instead it begins exploiting quickly around a relatively low reward. What happens is that when the surrogate model evolves to a situation where the sum of the mean and the weighted variance of the model (as in Eq. (2.12)) are smaller than the current y_n^+, the maximum of UCB will be always at the location of y_n^+ or locations that are very close to it and d_n will converge to zero. This happens early for low values of κ resulting in an earlier exploitative behavior. This in turn results in higher steadily increasing $\bar{y}_n$ curves.

Choosing higher values for κ delays this exploitative behavior, allowing the search to explore more positions and preventing it from easily falling in local maxima. However, it results in a lower $\bar{y}_n$ curves due to the increased risk of falling in areas of low rewards because of exploration.

A very important insight here is that, while PI and EI manage exploration and exploitation indirectly by controlling the desired improvement, UCB directly balance exploration and exploitation and hence is able to stop looking further when 'satisfied'. This makes the $\bar{y}_n$ curves of UCB (even for the high values of κ) the highest among the three functions. However, because UCB sooner or later stops exploration, it is generally more probable to get stuck in a local maximum.

Looking at the performance of three functions in the 2D case, it can be concluded that PI is more capable of reaching the maximum y_n^+ and UCB has the lowest probability of falling into areas of low rewards (hence has high $\bar{y}_n$ curves). The performance of EI in both criteria is inferior. It is important to investigate if these findings hold for higher numbers of dimensions.

3.3.2 Scaling up to 3D

Figures 3.8, 3.9 and 3.10 show the behavior of PI, EI and UCB respectively on 3D functions for a range of settings. One note to begin with is that the averages of the maximum reward y_{10}^+ and the mean reward $\bar{y}_{10}$ during initialization are much lower that these of the 2D functions. This is because of the increased sparsity due to the added dimension making it harder for random initial points to pick potions of high rewards.

y_{10}^+ can be considered the 'initial knowledge' the optimization algorithm has about the value and the corresponding position of the maximum. It can be seen that with this small initial knowledge, there was no way of reaching the global maximum (whose value is slightly higher than 1.0) with 10 initial observations and another 40 Bayesian optimization iterations. Note that y_{10}^+ and $\bar{y}_{10}$ are always that same regardless of the setting. This is because every randomly generated function has its own set of random initial points that is retained throughout the experiments as explained in section 3.2.2.

PI Results

Looking to Fig. 3.8, the same PI curves trends for y_n^+, $\bar{y}_n$ and d_n curves seen in the 2D case can be seen here. However, in the case of 3D functions, we find that the setting leading to the highest y_n^+, is the same as the one with the highest $\bar{y}_n$ curve and is the smallest value ($\xi = 0.001$). This raises the concern that the optimal setting can be different depending on the number of dimensions of the black-box function, and brings the natural question: how can one determine the optimal setting?

In a practical realistic scenario, this would be very inconvenient. Consider the black-box reward function of the interaction controller of a drone. The user may decide to use Bayesian optimization to learn only the translational and rotational parameters of the virtual spring (see Appendix A), with the limitation that all the translational parameters are equal to each others and the same limitation for the rotational parameters. This problem is 2D. The user might after

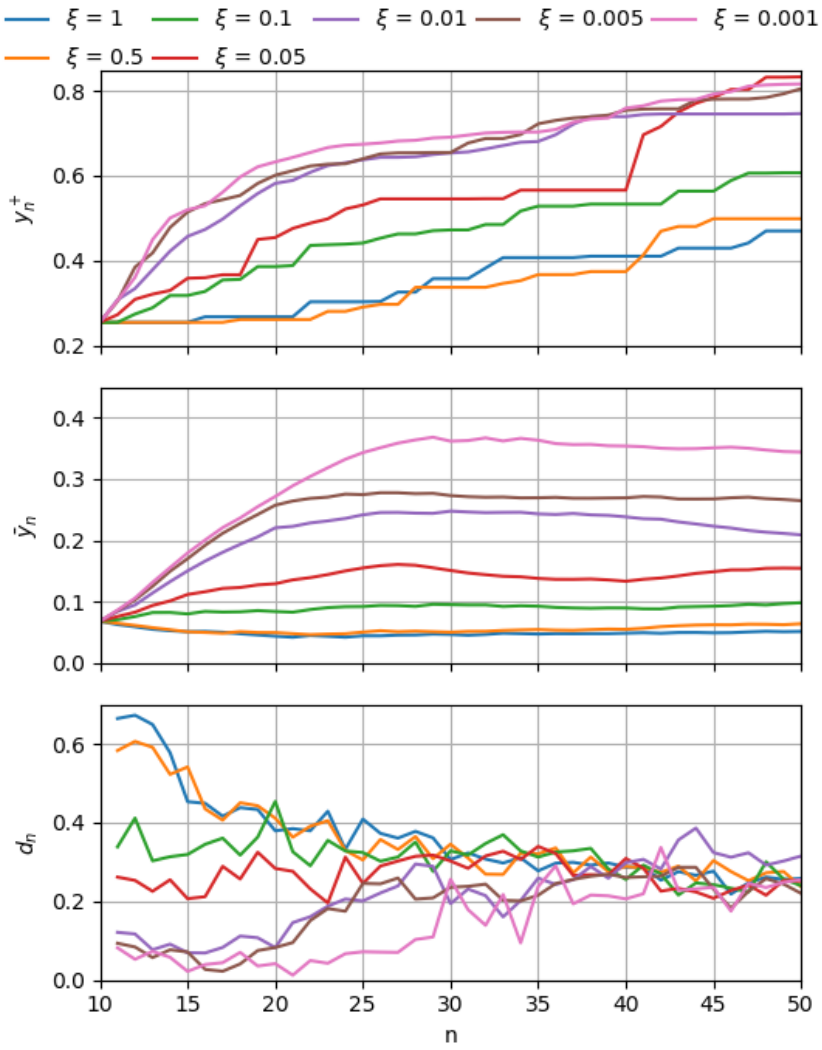


Figure 3.8: Performance of PI with different values for ξ over 3D functions

that decide to learn each of the spring parameters individually, which converts the problem to a 6D one although we are still working with the same dynamic system. What this observation says is that the user might need to change the settings of the acquisition function or use a completely different one.

Notice that these issues are only significant due to the limited number of allowed black-box function observations (again due to the high cost of a single observation). In problems where hundreds or thousands of observations are allowed, these issues are not of great influence, and hence are not often tackled in literature.

One important notice about the d_n curves of PI is that, as in the 2D case, the search remains, at the beginning, closer to the position of the already seen observations for lower values of ξ, all the settings finally converge to nearly the same value of around 0.25 due to the continuously explorative nature of improvement-based acquisition functions. The value d_n converges to is more than the double of what was seen in the 2D case. This is again due to the increased sparsity at higher numbers of dimensions. Notice, however, that for the highest y_n^+ curve ($\xi = 0.001$), the iterations where d_n was small witnessed the highest rate of growth. This observation will be essential for the suggested improvement in section 3.4.

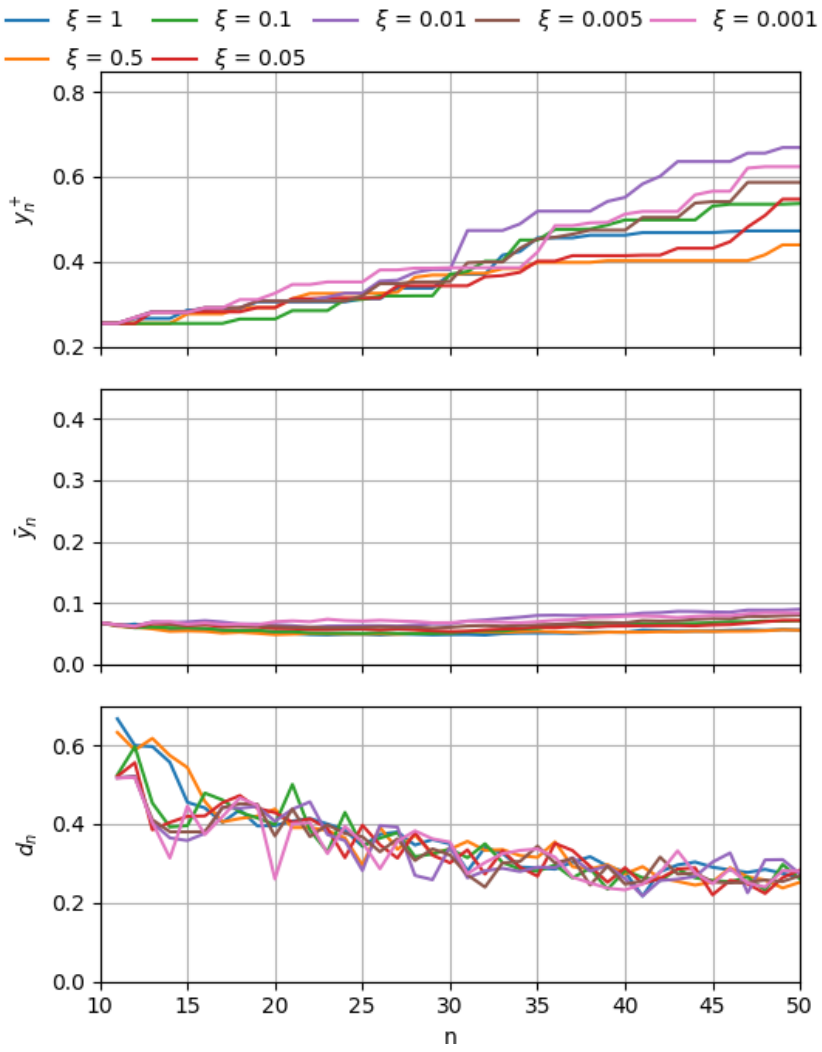


Figure 3.9: Performance of EI with different values for ξ over 3D functions

EI Results

The performance of EI (Fig. 3.9) deteriorates more with one higher number of dimensions. The function is much slower than the other two in reaching high y_n^+ values and also has, by far, the lowest $\bar{y}_n$ curves. This clearly indicates that EI is not suitable for sample-critical situations. It needs a higher number of observations for its merits to be seen, which makes it unsuitable for the application at hand.

UCB Results

The performance of UCB (Fig. 3.10) with 3D functions is interestingly different than it is with 2D functions. While $\bar{y}_n$ and d_n keep the same trend seen before, we notice the performance is better here with smaller values for κ. This is also consistent with what happened with the PI function, which performed better in 3D with smaller values for its hyperparameter. This supports the intuition that, in higher dimensions, more local/exploitative search is favorable. It is more likely to get lost in a big jungle if you are moving too fast without deliberate thinking.

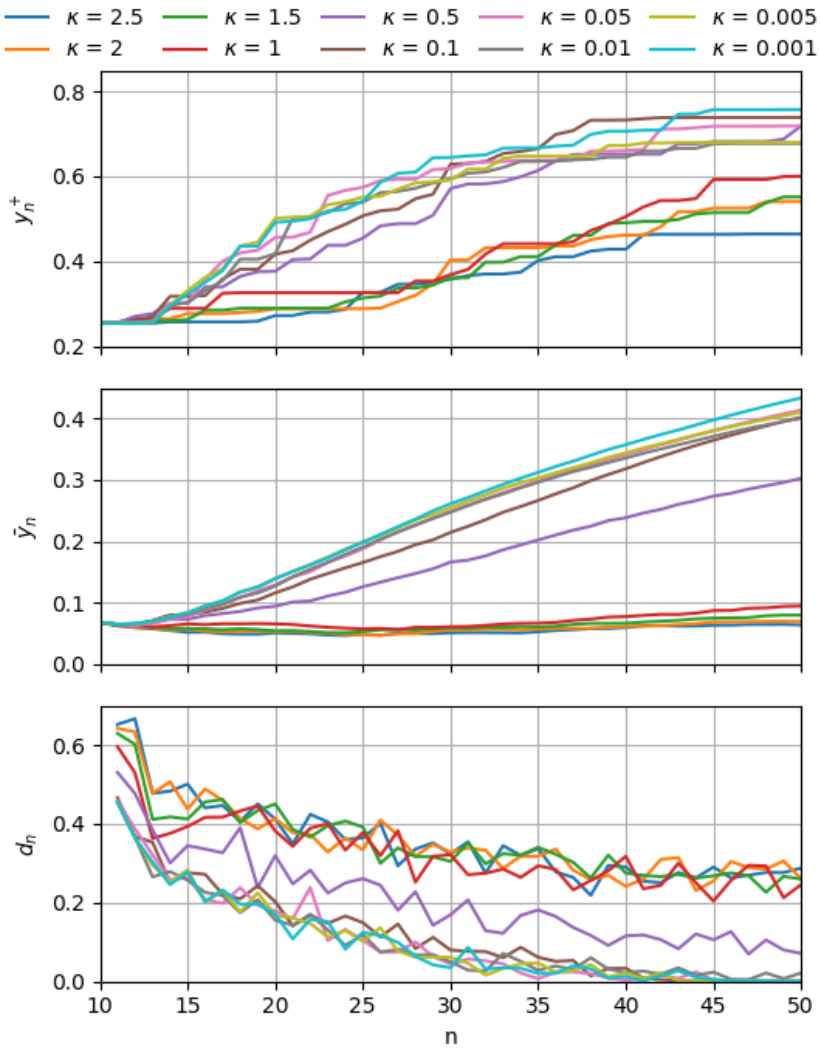


Figure 3.10: Performance of UCB with different values for κ over 3D functions

3.3.3 Scalability at Higher Dimensions

After investigating the performance of the three acquisition functions with a range of values for their hyperparameters along random black-box functions of four consecutive numbers of dimensions. The single setting of each function that produced the best average y^+_{50} along the four different numbers of dimensions was selected and is shown in Fig. 3.11.

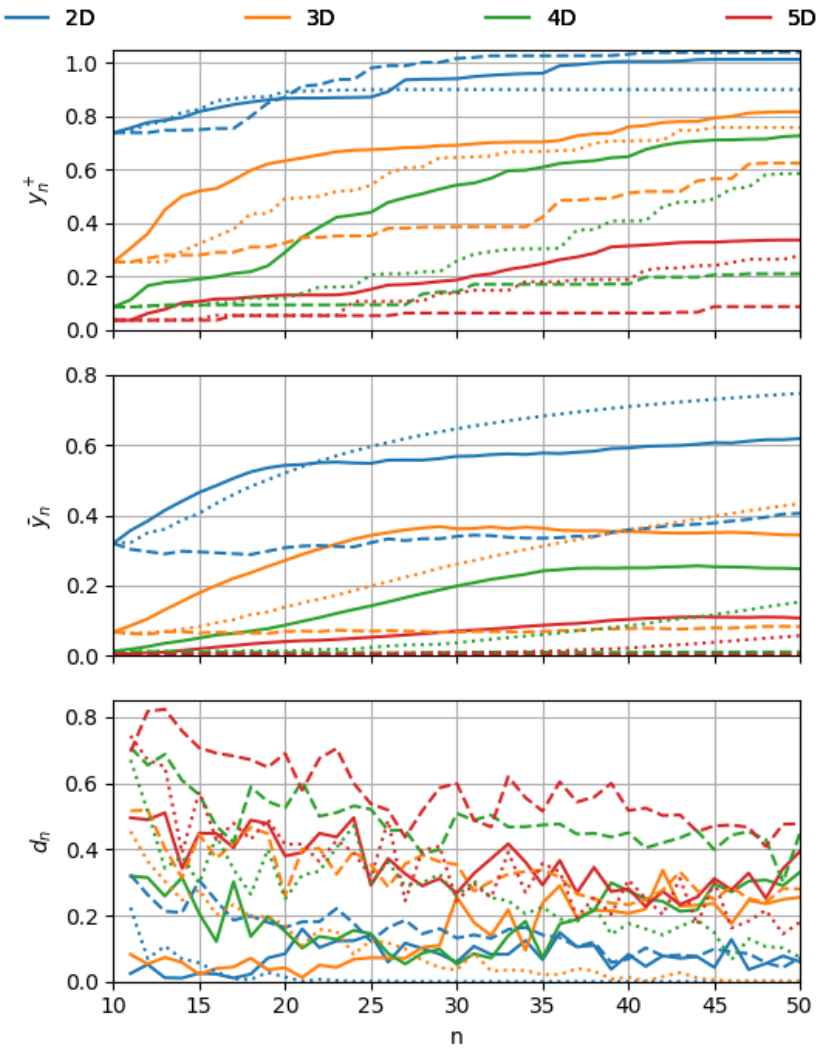


Figure 3.11: Best performing setting for each acquisition function along four consecutive numbers of dimensions. Solid: PI ($\xi = 0.001$). Dashed: EI ($\xi = 0.001$). Dotted: UCB ($\kappa = 0.001$).

Notice that as the number of dimensions increases, the initial knowledge y^+_{10} obtained by random initial samples significantly decreases. It then becomes harder to reach to the global maximum.

The single setting that gave the best results for the three acquisition functions along the different numbers of dimensions was the most local/exploitative one ($\xi = 0.001$ for PI and EI and $\kappa = 0.001$ for UCB). This was especially true for higher number of dimensions where any higher ξ or κ values make the performance worse. It is still possible to note that within these 50 observations, the best performance of EI is worse than that of PI and UCB with respect to both y^+_n and $\bar{y}_n$ curves. The only exception is the y^+_n curve for 2D random functions, where a more explorative PI and UCB settings (higher ξ and κ) are needed to overcome EI. This general result confirms that EI is not suitable for few number of observations.

Other general trends can be obtained also. It can be seen that PI is always superior in y_n^+ curves. Looking to $\bar{y}_n$, however, we notice that PI ones tend to be lower than those of UCB with lower numbers of dimensions and higher with higher numbers of dimensions. This is because, as the number of dimensions increase, the very exploitative nature of UCB (with very low κ) prevents it from being able to reach areas of high rewards quickly. Notice, however that more explorative settings of UCB (higher κ) produced even worse results as the search was not able to fine-tune and climb hills surrounding the initial observations.

It is important to see that the d_n curves of UCB at higher numbers of dimensions (4D and 5D) are still high similar to those of PI. These higher d_n curves are associated with lower $\bar{y}_n$ curves, while in lower numbers of dimensions, UCB has mush lower d_n curves associated with superior $\bar{y}_n$ curves. This indicates that a search that props closer to the positions of the observation seen so far has a much lower risk of falling to areas with small rewards. This is also a very important observation that will be key to the discussion in section 3.4.

3.3.4 Summary

To summarize this parameter study section, improvement-based acquisition functions like PI and EI are more proof against getting stuck in local maxima due to their continuously explorative nature. For the same reason, however, they tend to have lower $\bar{y}_n$ curves as they have more probability of falling into areas of low rewards. In general, the performance of PI, in terms of y_n^+, is much better than that of EI within a low number of observations, and is also superior to that of UCB. However, as with the other acquisition functions, the optimal hyperparameter value may be different from a number of dimensions to another. UCB allows direct exploration/exploitation trade-off and is able to completely stop exploring when *satisfied* about the obtained maximum, because of that, it is more likely to get stuck in local maxima, especially for low values of κ. For the same reason also, it is more capable of having high $\bar{y}_n$ curves and avoiding areas of low rewards.

3.4 Local Optimization of Acquisition Functions

In this section, a novel[2] idea for optimizing the acquisition function is presented. Although an essential part of Bayesian optimization, it is very difficult to ensure that the real global maximum of the acquisition function is found as discussed in section 2.4. This problem becomes more evident as the number of dimensions of the target black-box function increases [24].

Even though all the recently established theoretical proofs of the convergence of Bayesian optimization are only valid given that the position of the true global maximum of the acquisition function is found every iteration and is selected as the next point to observe the target black-box function at, in practice it is nearly impossible to ensure this condition [24]. In this section I take a completely different approach and give up the quest for finding this global maximum for a more local way of optimizing. I argue that the presented approach is more safe and efficient, especially for black-box functions related to robotic dynamic systems. I then show empirical results displaying the advantages of the new strategy.

3.4.1 Motive

Based on the observations seen so far in parameter study in section 3.3, using Bayesian optimization as it is has some issues when applied to dynamic robotic systems. They are summarized here:

- Different numbers of dimensions may require different settings. It is unclear how to adjust the settings (the hyperparameters of the acquisition functions) given the number of dimensions of the target function.

[2] To the author's best knowledge, as always.

- Working with robotic systems, using explorative acquisition functions is risky as the search is more probable to fall into areas of low rewards (corresponding to bad performance or even instability).
- While very local/exploitative settings do not suffer from this issue, they are more probable to get stuck in a local maximum, especially for the UCB function.
- While PI presents a good compromise as it continues exploration even with very small values of ξ, making it able to quickly reach high y_n^+, the exploration is not properly controlled making it occasionally and repeatedly fall into areas of low rewards. This results in low $\bar{y}_n$ curves.

Considering these issues, it appears that the key to a successful employment of Bayesian optimization in sensitive robotic applications is to find a way to make the search for the next point to sample explorative enough to prevent getting stuck in local maxima, but also local enough to avoid suddenly falling to areas of low rewards.

3.4.2 Idea

The idea is very simple. Instead of trying to find the global maximum of the acquisition function (line 4 in Algorithm 1), only areas close to the positions of the already seen observations are considered. Specifically, at each iteration n and for all $i \leq n$, a hyper-rectangle B_i is virtually placed centered around each point $\mathbf{x}_i$ with dimensions equal to that of the domain of the target function scaled down by the hyperparameter $d < 1.0$ (distance).

The next point to sample is then selected as the position of the maximum of the acquisition function within hyper-rectangles $B_{1:n}$:

$$\mathbf{x}_{n+1} = \underset{\mathbf{x} \in \bigcup_{i=1}^{n} B_i}{\arg\max}\, \alpha(\mathbf{x} \mid \mathscr{D}_{1:n}) \tag{3.7}$$

The way a hyper-rectangle B_i is formed is as follows. Suppose χ the domain of the black-box function is given as a two-columns matrix such that row j represents the lower and upper limit of the input of the function along dimension j:

$$\chi^{j,1} \leq x^j \leq \chi^{j,2}$$

where superscripts here are used as row-based matrix indices and x^j is the component of the input $\mathbf{x}$ along dimension j. Then B_i is formed also as a two-columns matrix such that its row j is:

$$B_i^{j,1:2} = [\max(\chi^{j,1}, x_i^j - \frac{D_j}{2}), \min(x_i^j + \frac{D_j}{2}, \chi^{j,2})] \tag{3.8}$$

where $D_j = d(\chi^{j,2} - \chi^{j,1})$ is the scaled distance along dimension j. The max and min operations above are used to ensure that the hyper-rectangles do not go outside the domain of the target function.

At each iteration of Bayesian optimization, a new hyper-rectangle is added as the number of obtained observations increases. Then the next point is chosen by optimizing the acquisition function in the areas within all the hyper-rectangles since the acquisition function within them changes its shape as the surrogate model evolves.

The idea is illustrated in Fig. 3.12 where the behavior of Bayesian optimization with global optimization of the acquisition function is compared to that with the proposed local optimization on a simple 1D Example. Both Bayesian optimization runs started with the same three initial observations (red dots in figures 3.12a and 3.12b) and are using the same UCB acquisition function. In the first iteration (upper row), the location of the global maximum of the acquisition

function (Fig. 3.12a) offers a low value of the objective function, while with the locally optimized acquisition function (Fig. 3.12b), the search was able to avoid falling into this area of low reward by moving only in small steps.

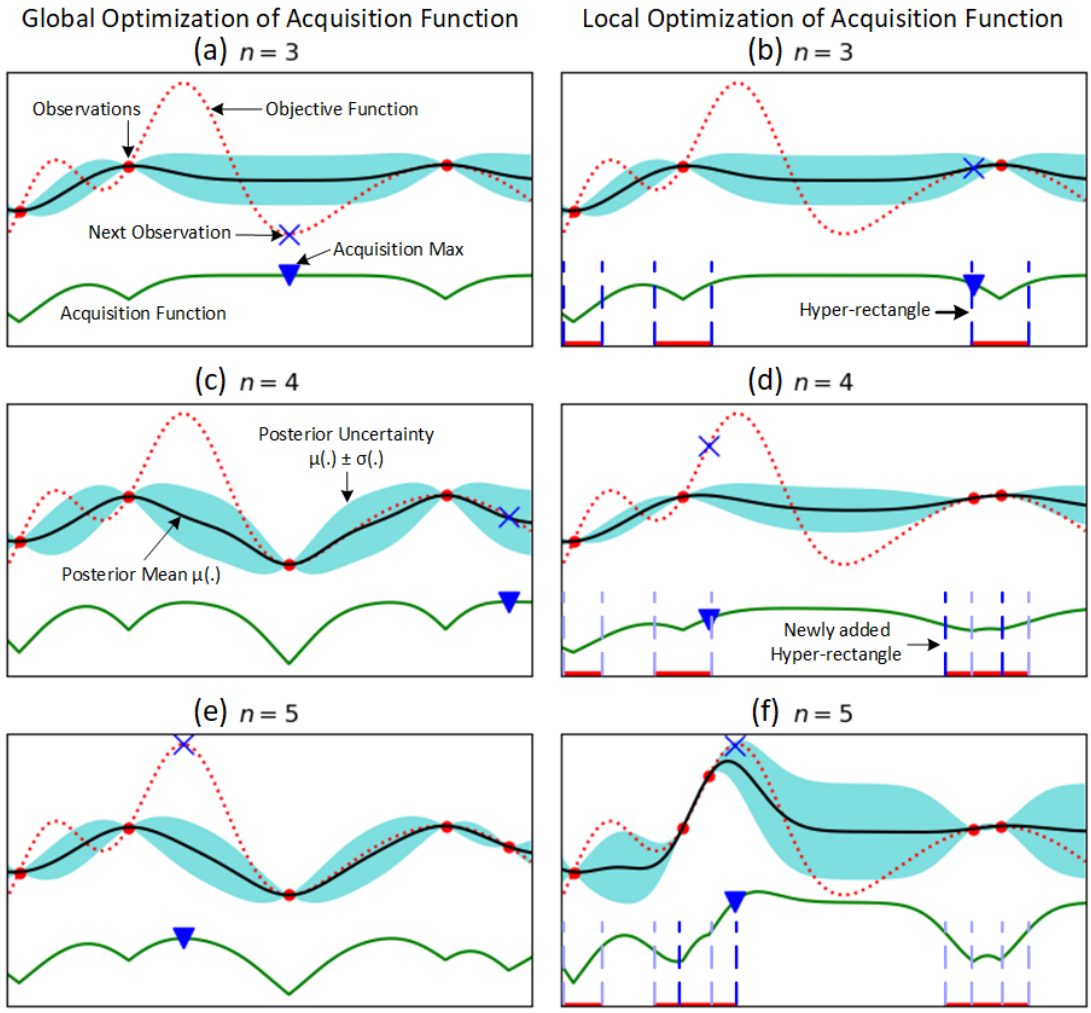


Figure 3.12: Behavior of Bayesian optimization with global optimization of the acquisition function compared to that with the proposed local optimization on a simple 1D Example. The meaning of the different elements of the subfigures are indicated in the graphs. Theses meanings are the same across all subfigures.

With the proposed local optimization, at each new iteration (figures 3.12d and 3.12f), a new 1D hyper-rectangle is added intersecting with an existent one. The acquisition function is optimized within all hyper-rectangle to select the position of the next observation. This allows local optimization to get early warnings along paths leading to low rewards and to quickly find paths leading to higher rewards. After three iterations (bottom row), both global optimization and local optimization searches are about to find the global maximum. However, local optimization got higher rewards along the way. This can be seen by observing that at each row the value of the next observation resulting from local optimization (blue cross in the right graph) is higher that resulting from global optimization (blue cross in the left graph).

In a robotic system context, this approach provides the following benefits:

- Exploitation of initial observations: As discussed before, meaningful initial observations are vital, especially to problems of high number of dimensions. By using this local optimization idea, setting d to a properly small value, the search will be forced to consider areas around these initial observations first where it is more likely to find the high rewards. So, given good initial knowledge, it can be reasonably expected that this technique will lead to higher y_n^+ curves.

- Less probability of falling into areas of low rewards: Setting d to a properly small value allows taking advantage of the explorative acquisition functions with much less risk of falling to areas of very bad performance or instability. This is because the search is allowed to move away from the positions of the already seen observations only in small steps, allowing it to get early warnings along paths that lead to minima. So, it can be expected that this method also leads to higher $\bar{y}_n$ curves.

- Cheaper optimization of the acquisition function: Because the area being optimized is much less, the optimization computational cost is significantly less than that of global optimization. This is especially important if the learning algorithm will be implemented on an on-board computer.

3.4.3 Implementation

The principle of locally optimizing the acquisition function in the area around already seen observation is independent of the way this optimization is achieved, whether that was random sampling, adaptive grids and/or the other methods mentioned in section 2.4.

In the implementation used to generate the results shown next subsection, the `BayesianOptimization` package was modified to add the possibility of forming the hyper-rectangles and optimizing the acquisition function within them. The optimization is done in a similar way to the default global optimization used in the package (described in section 3.2.2) but scaled down to the size of each hyper-rectangle. The default settings for the current implementation of this presented local optimization technique consists of obtaining 2,000 random samples per hyper-rectangle and running a quasi-Newton optimization (L-BFGS-B) initiated at 5 random points within the hyper-rectangle and limited by its borders. The point that results in the highest value among all the hyper-rectangles is selected.

Considering these settings for a Bayesian optimization with ten initial observations and a subsequent 40 iterations ($n = 10$ up to and including $n = 49$ in algorithm 1), at each iteration n the area belonging to n hyper-rectangles is optimized. This means that in 40 iterations, a total of 1180 hyper-rectangles will be optimized. Subsequently, a total of 2,360,000 random samples will be obtained and L-BFGS-B will be run for 5,900 times. With the default global optimization settings, the same number of iterations cost 4,000,000 random samples and 10,000 runs of L-BFGS-B. So, using the presented local optimization technique can save nearly 50% of the computational cost due to optimizing the acquisition function, while giving a better performance as will be seen in the next subsection.

Notice that, because each new observation is selected within an already present hyper-rectangle, new hyper-rectangles will certainly intersect with the existing ones as seen in Fig 3.12. Current implementation is kept simple and does not take advantage of this fact, so areas of intersection receive more concentration in the optimization process. When taking this into consideration, more reduction of computational costs can be obtained.

3.4.4 Results and Discussion

The following results show some of the merits of the presented method. They are all obtained by running Bayesian optimization on the same randomly generated functions used to generated the results seen so far. The simulations were done with the hyperparameter $d = 0.1$, which means that the dimensions of the generated hyper-rectangles are 10% of those of χ the domain of the black-box target function.

Figure 3.13 shows PI locally optimized with different values for its hyperparameter ξ. It can be seen that the new technique cured the problem of needing to change the settings as the number of dimensions changes. We see that the more explorative setting $\xi = 0.05$ is able to maintain a good performance along all shown numbers of dimensions. This also suggests that a more explorative acquisition function combined with local optimization of the acquisition function is a superior strategy in terms of getting high y_n^+ curves.

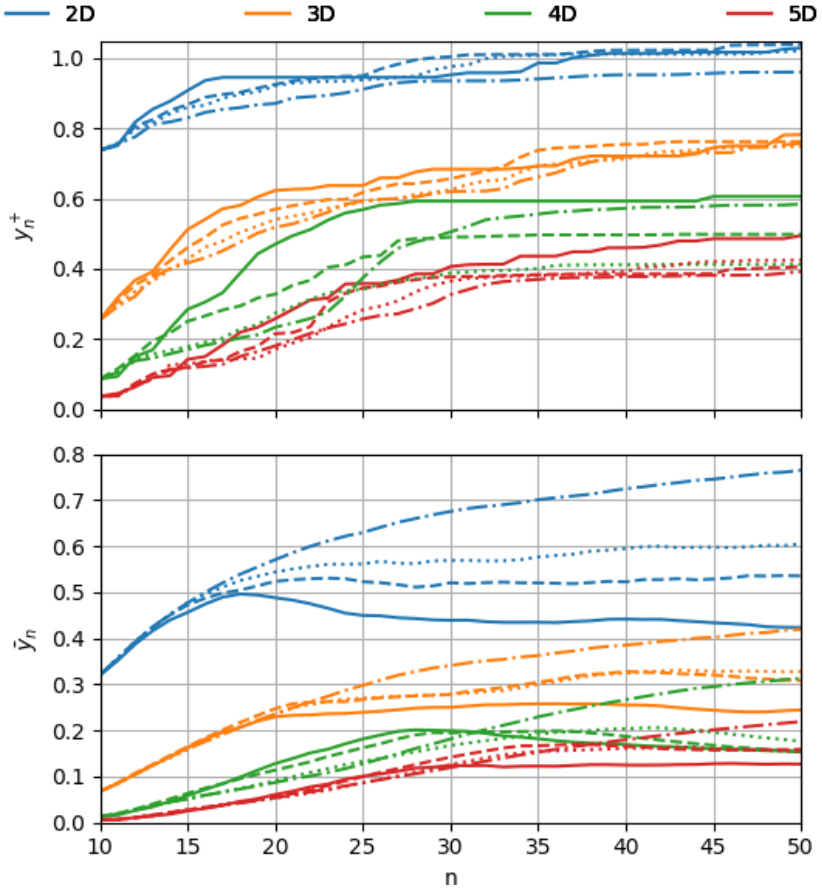


Figure 3.13: Performance of PI locally optimized with $\xi = 0.05$ (solid), 0.01 (dashed), 0.005 (dotted) and 0.001 (dash-dotted), along different numbers of dimensions.

Notice also that this local optimization technique did not result in losing the possibility to control the locality of the search. We can see that for smaller values of ξ, y_n^+ increases slowly as the search is very local. On the other side, these small values of ξ result in the highest $\bar{y}_n$ curves. For all settings, however, the $\bar{y}_n$ curves with locally optimized PI are higher than those of the globally optimized PI, which supports the hypothesis made in section 3.4.2. To compare, refer to Fig. 3.5, 3.8 and 3.11.

It is important also to see that the y_n^+ curves resulted from locally optimizing PI are very close to those resulted from global optimization of PI for lower numbers of dimensions (2D to 4D), but locally optimized PI is significantly superior with higher numbers of dimensions (again compare with Fig. 3.11). This supports the observations and intuitions gained in section 3.3 that at

higher numbers of dimensions, it becomes more important to limit the distance between the new observation and already seen observations.

Another very interesting result can be seen in Fig. 3.14. It was concluded in section 3.3 that EI is not suitable for sample-critical cases because of its very explorative nature (even for very small values of its hyperparameter ξ. Here we see local optimization of EI significantly boosting its sample-efficiency.

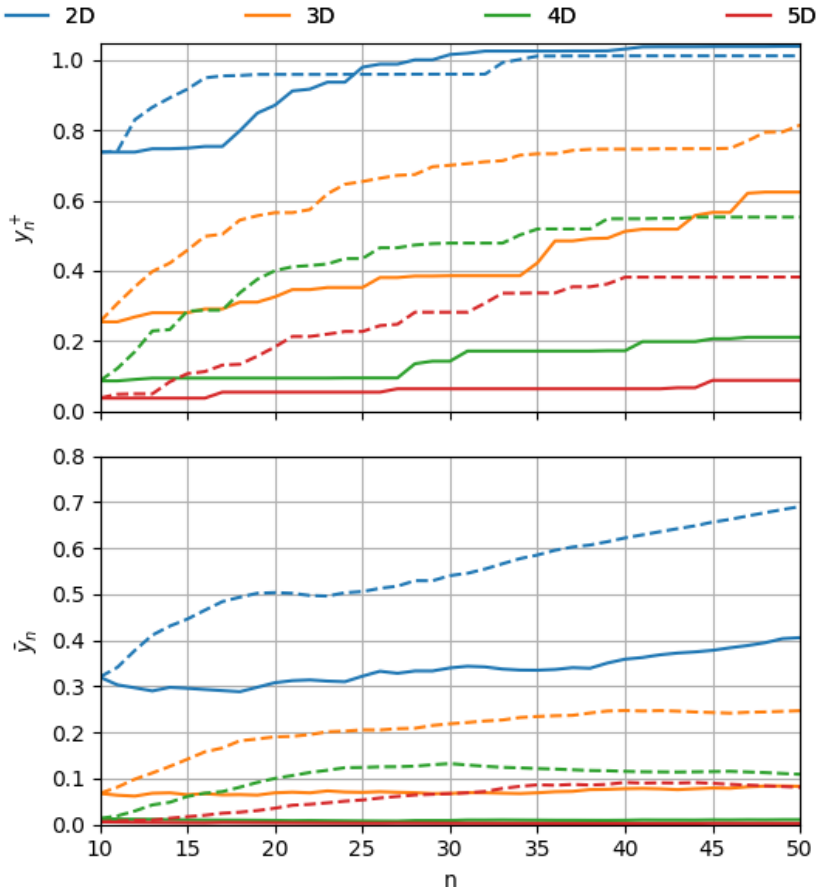


Figure 3.14: Performance of EI globally (solid) and locally optimized (dashed) with $\xi = 0.001$.

Comparing the locally optimized EI curves (dashed) with the globally optimized ones (solid) along different numbers of dimensions, it can be clearly seen that locally optimized EI is substantially both faster in finding higher y_n^+ and abler to avoid areas of low rewards, hence has much higher $\bar{y}_n$ curves. This is due to the early warning provided by limiting the maximum distance between the selected observation and the previous observations.

Finally, Fig 3.15 shows how local optimization significantly improved the performance of UCB, especially with target functions having higher number of dimensions.

It was seen in section 3.3, that very exploitative UCB settings (low values of κ) result in getting stuck early in a local maximum, but are also much less probable of falling to areas of low rewards, which is an important feature for robotic systems as discussed before. On the other hand, very explorative settings of UCB are not capable of fine-tuning the search to reach areas of higher rewards. Looking to Fig. 3.15, it can be seen that the locally optimized explorative UCB (dotted) is able to get the advantages of both the globally optimized explorative (solid) and exploitative (dashed) settings while greatly curing their disadvantages.

It can be noticed that the locally optimized UCB is always faster to reach higher y_n^+ in a lower number of iterations. The difference is more significant as the number of dimensions increases.

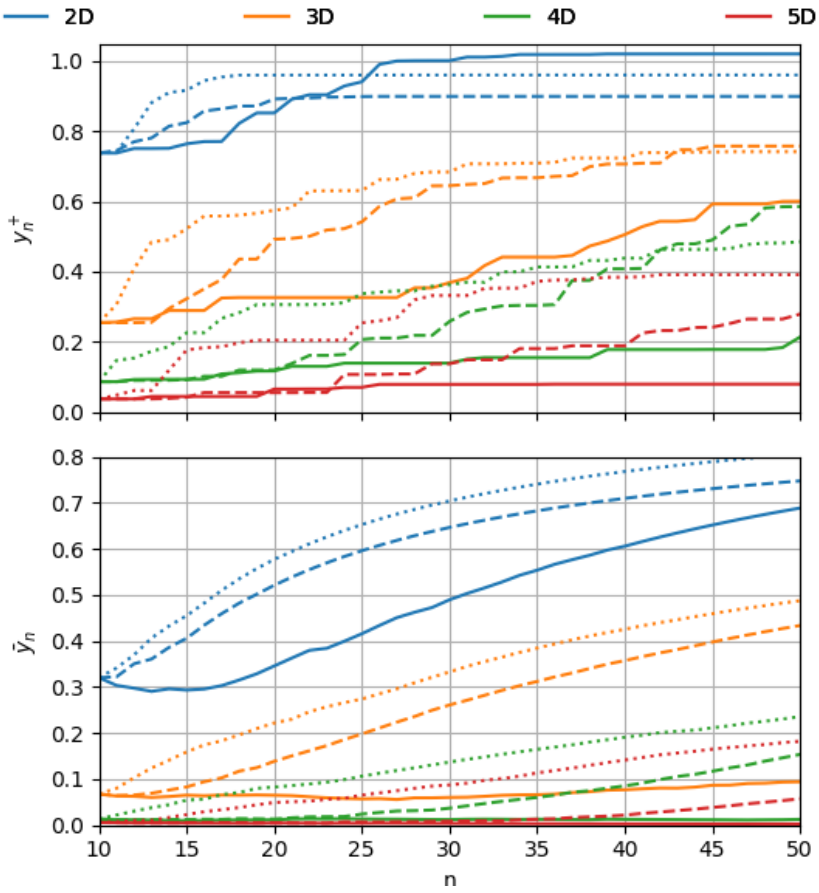


Figure 3.15: Performance of UCB globally optimized with $\kappa = 1$ (solid) and 0.001 (dashed), and locally optimized with $\kappa = 1$ (dotted).

It can be also seen that its $\bar{y}_n$ curves are always higher than both the explorative and exploitative settings of the globally optimized UCB.

To conclude, the presented results showed a clear evidence that this simple modification has the ability to solve the issues mentioned in section 3.4.2 and make Bayesian optimization faster, safer, and computationally cheaper. It can be said that it is best to combine this strategy with an explorative acquisition function. PI (with $\xi = 0.05$) proved to be superior based on the performed tests.

3.5 Alternative Initial Observations

As seen in section 3.3.3, as the number of dimensions of the target black-box function gets higher, initial random points give less indication about the potential positions of the maxima and offer no help in guiding the search. This in turn makes it very difficult to reach areas of high rewards as the search needs a number of iterations of clueless exploration until being able to pick traces of positions of high values (hills).

While the choice of the surrogate model (via selecting a kernel) is the most significant choice in Bayesian optimization as discussed in section 2.3, providing meaningful initial observations is vital for a sample-efficient optimization. Notice that in Fig. 3.11, for example, the rate of change in y_n^+ is not very different for PI among the different numbers of dimensions. The difference was in y_{10}^+, which indicates that initial observations play a key role in allowing reaching the global maximum within a reasonable number of observations.

The ideal way of providing meaningful initial observations is through expert knowledge or intuition. For example, for the black-box reward function of the interaction controller of a drone,

an expert engineer can use his/her knowledge in selecting parameters values that most likely lead to a good performance. By providing it a range of observations at such parameters values, Bayesian optimization can quickly build a meaningful surrogate model and use it to efficiently navigate through the black-box function reaching the global maximum in a minimal number of iterations.

Supposing, however, that this ideal way is not available, we investigate here the effectiveness of sampling the initial points in a more systematic way, to cover different areas of the target function, in obtaining better initial knowledge. In particular, Latin hyper-cube sampling is used [30]. This is a near-random sampling technique where, when sampling a function of N variables, the range of each variable is divided into M equally probable intervals. M sample points are then placed to satisfy the Latin hyper-cube requirements such that there is only one sample in each row and each column. Figure 3.16 shows possible positions of 8 Latin hyper-cube samples obtained from a function with two variables. In a Bayesian optimization context, this sampling technique was used for initialization in [16].

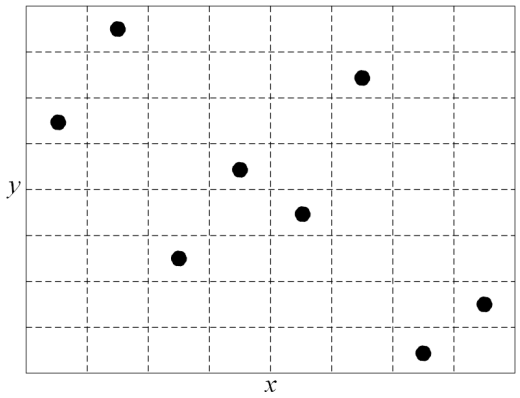


Figure 3.16: A possible way to draw 8 Latin hyper-cube samples from a function with two variables.

To examine this idea, ten random Latin hyper-cube samples were generated per random function. Bayesian optimization was run with PI set on the best performing setting ($\xi = 0.001$) along different numbers of dimensions and the result is compared to the result with ordinary random initial points (used in all experiments so far) in Fig. 3.17.

It can be seen that no significant performance improvement can be noticed, especially for higher numbers of dimensions. The reason might be that the number of initial observations is small so using Latin hyper-cube sampling did not make a difference. To investigate that, Bayesian optimization was run with 25 initial observations per random function. The results with the same setting as before are shown in Fig. 3.18.

It can be clearly noticed that the initial knowledge y^+_{25} is higher than the initial knowledge obtained by only 10 initial observations, and hence, Bayesian optimization was able to reach higher values by the end of the 50 observations although it was run only for 25 iterations (see the 5D curve for a clear significant improvement). This emphasizes the importance of providing meaningful initial observations. However it can be seen that Latin hyper-cube sampling did not result in a higher y^+_{25} than random sampling. contrarily, it gave a worse performance for all numbers of dimensions except for 4D. Which indicates that there is no reason to think that this method provides a better initialization than the ordinary uniformly random sampling. For a reliable sample-efficient initialization, however, there is no alternative for expert rough knowledge or intuition.

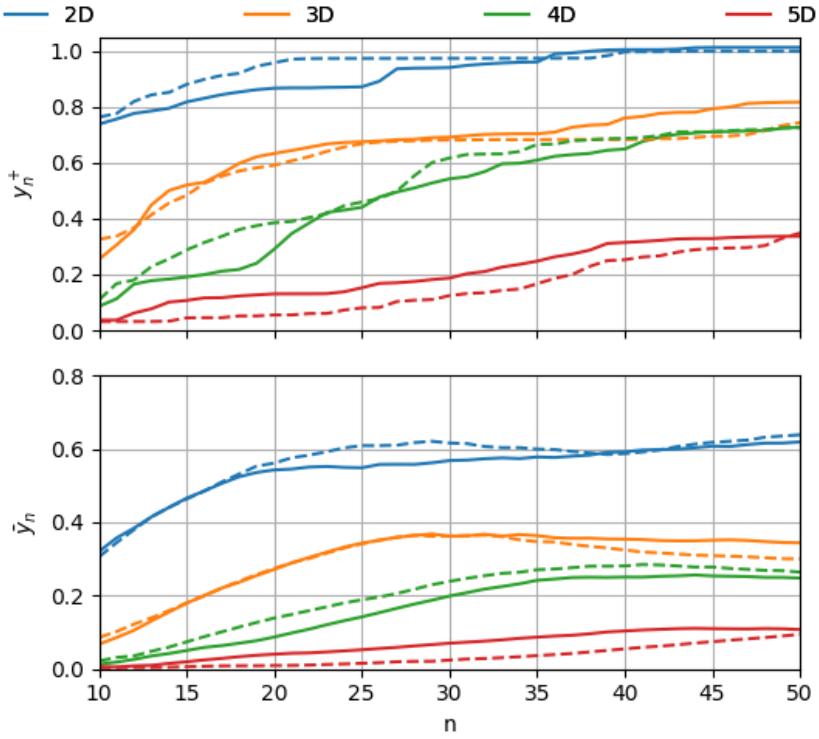


Figure 3.17: Performance of Bayesian optimization with PI ($\xi = 0.001$) along different numbers of dimensions initialized uniformly randomly (solid) and with a Latin hyper-cube (dashed).

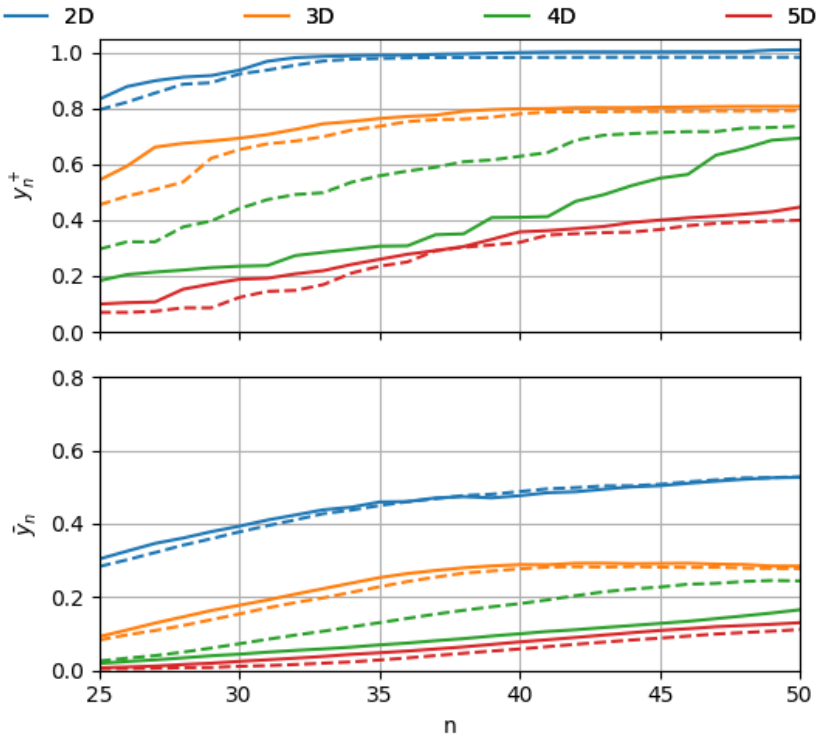


Figure 3.18: Performance of Bayesian optimization with PI ($\xi = 0.001$) along different numbers of dimensions initialized with 25 initial observations selected uniformly randomly (solid) and with a Latin hyper-cube (dashed).

3.6 Conclusion

Bayesian optimization was applied to randomly generated functions with different numbers of dimensions to gain deeper insights about it. A parameter study was performed over the different acquisition functions to see how the performance of Bayesian optimization changes from one acquisition function to another and how the values of the acquisition functions hyperparameters affect the behavior of the learning. The study found that PI is able to achieve higher rewards more quickly than other acquisition functions and UCB is less probable to fall into areas of low rewards during searching. It was also found that EI requires more samples to start reaching areas of high rewards and hence is not suitable for sample-critical applications like the one at hand.

The study concluded that the performance of Bayesian optimization with the different acquisition functions presents potential problems when used for learning the impedance parameters values of a drone. In particular, it was found that, while exploratory methods are less prone to getting stuck in local maxima, exploration causes the search to repeatedly fall into areas of small reward. Very exploitative searches are safer, but suffer from the problem of getting stuck at local maxima. Additionally, it was noticed that the settings leading to the best learning performance change from a number of dimensions to another making the task of tuning the settings harder.

To address these issues, a novel method for optimizing acquisition functions was proposed. The method relies on optimizing the local areas around the already seen observation instead of optimizing globally. That way, the risk of falling into areas of low rewards is decreased because the search is able to receive *early warnings* along paths leading to minima. Empirical experiments showed the proposed technique is able to reach higher rewards quickly exploiting the given initial knowledge. It also helps making the performance of learning settings more consistent across different numbers of dimensions. As a result of using the proposed technique, the performance of the three considered acquisition functions significantly improved. However, PI is still superior in sample-efficiency.

The parameter study also showed that as the number of dimensions of the target function increases, the maximum achieved reward by the provided randomly sampled initial observations decreases. This is due to the increased sparsity at higher number of dimensions. A more systematic way of obtaining initial observations locations that relies on Latin hyper-cube sampling was investigated. However, empirical results showed that this technique gives no advantage over random sampling. It is then inevitable to provide meaningful initial observations using human expert intuition to make learning problems with higher numbers of dimensions tractable.

In the next chapter, these insights, findings and developments are used to successfully learn the impedance parameters values for a simulated drone.

4 Simulation Results and Discussion

The simulations done in this thesis are centered around a fully-actuated hexarotor drone developed in the robotics and mechatronics lab (RaM) in University of Twente. The drone is named *betaX* and is able to pose in any configuration due to the full actuation, which gives it an advantage in interaction and contact tasks.

In this chapter, Bayesian optimization, with the choices and developments done in the two previous chapters, is used to learn the impedance parameters of a simulated version of the betaX hexarotor in several situations. First the simulated interaction tasks are described in section 4.1. The design of the reward function is discussed in section 4.2. The section also presents a novel method for conditioning the reward function. After that, section 4.3 contains a detailed description of the procedures followed in the experiments as well as the used learning settings. Section 4.4 shows and discusses the obtained simulation results. Finally, the chapter is concluded in section 4.5.

4.1 Simulation Description

Simulations will be done for the surface sliding task. In this simple learning task, the drone learns the optimal impedance parameters for drawing a certain shape on a wall. The aim is to provide a clear proof-of-concept for using the proposed learning framework to find impedance parameters values.

For the purpose of this simulation, the impedance controller described in Appendix A was implemented for betaX. In addition, a software and GUI to learn the implemented impedance controller's parameters were built. The whole software is implemented over `ROS` (Robotics Operating System) allowing it to be used for the real physical drone or for a simulated version of it. The controller is used on a simulated version of the drone via `Gazebo` and `RotorS` simulators [11]. Figure 4.1 shows the simulated betaX sliding on a wall. The controller, learning and simulation software is described in details in Appendix B.

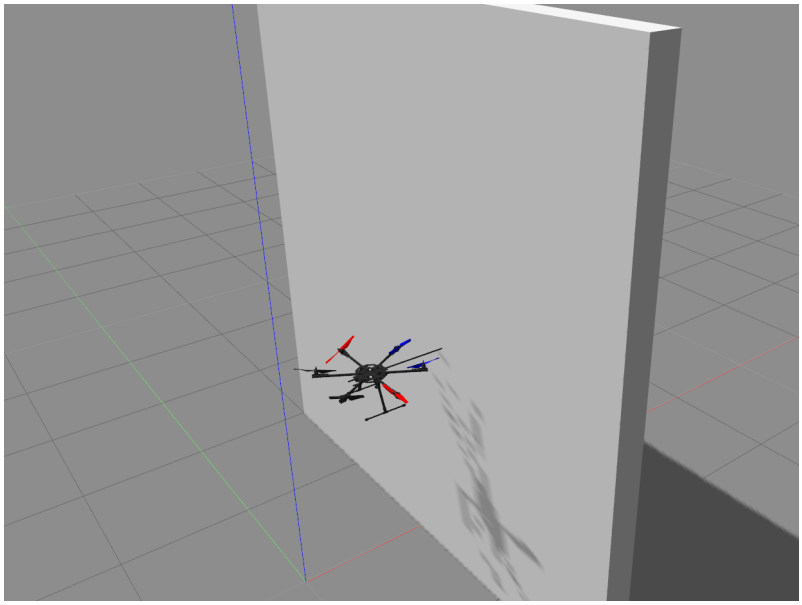


Figure 4.1: Simulated betaX drawing in a wall in `Gazebo`.

In the current implementation of the impedance controller considered in this work, the impedance parameters that can be controlled are the three diagonal elements of each of $\mathbf{K}_t$

and $\mathbf{K}_r$ (the translational and rotational stiffness gains matrices respectively) as well as the six elements of $\mathbf{K}_d$ (the diagonal damping matrix). The total is twelve controllable parameters. All other parameters are fixed at zero. These controllable parameters allow separate control over the translational and rotational stiffness and damping along each axis, without the possibility of coupling the stiffness of multiple axes. For the task at hand, this is more than enough.

4.1.1 Sliding Task

To assess the performance of the drone given certain values for the impedance parameters and calculate the reward, the drone must perform an application-specific predetermined task at each training episode as discussed in section 1.3.

For the learning of the impedance parameters values for drawing a certain shape on a wall, the predetermined task that must be done at each episode is simply following a trajectory to actually draw the shape on the wall. The end-effector of the drone is commanded to follow a trajectory defined by $\mathbf{P}_C^I(t)$ and $\mathbf{R}_C^I(t)$, the command position vector and rotation matrix respectively with respect to the inertial frame Ψ_I at each time $0 \leq t \leq T$ where T is the total time of a training episode.

During a sliding task, the drone exerts a normal force on a surface while moving along its lateral. In particular, consider the wall in Fig. 4.2 which represents the same wall used in simulation (shown in Fig. 4.1). The 2D shape drawn by the drone is described in the plane of the wall (x-y plane). To exert a force on the wall while drawing, the command frame Ψ_C is placed *deep* within the wall. The depth is set by a parameter denoted by Δ. The magnitude of the force exerted on the wall is then a function of the impedance translational spring stiffness in the direction normal to the wall (stiffness along x-axis in our case) and the depth Δ of the command trajectory.

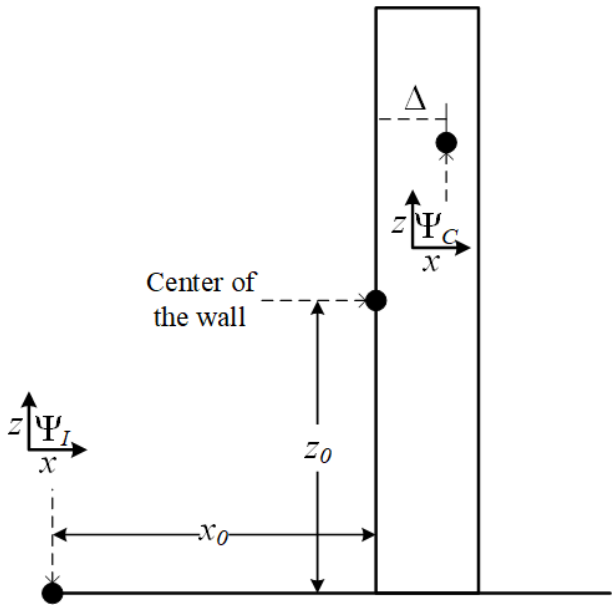


Figure 4.2: The surface sliding task from the x-z perspective. Right-handed frames are used.

Throughout a sliding task, the drone's orientation remains fixed. Only the position changes, i.e.

$$\mathbf{R}_C^I(t) = \mathbf{I} \tag{4.1}$$

Without using a learning technique and because the used impedance controller has a physical intuition behind it (modeled as a spring and a damper as mentioned in Appendix A), impedance parameters values can be selected intuitively as follows. Since the task involves

no change in orientation, rotational stiffness and damping should be set on a high value to minimize the rotational error. For the translational spring and damper, a distinction should be made between the direction normal to the wall and the directions parallel to the wall. Normal to the wall (x-axis), spring stiffness and damping should be low to allow more freedom of movement, but not too low to maintain proper contact with the surface. In the directions parallel to the wall, the stiffness and damping are selected to be high to maintain high accuracy in tracking the commanded trajectory.

This intuition, however, says nothing about the exact values of the suitable impedance parameters and in no way can it predict how these parameters values should change when the task changes. The suitable impedance parameters values for a sliding task are governed by several factors. Most significantly, the physical characteristics of the wall and the end-effector of the drone as well as the geometry of the drawn shape and the speed of the drone during drawing. For example, sliding/drawing on a smooth homogeneous surface would require different values for the impedance parameters than a rough surface. Also, for a single surface, drawing a smooth shape would probably require different impedance values than drawing a shape with discontinuities or rapid and narrow changes. Hence, we see that, even for this simple interaction task, it is difficult for human intuition and expert knowledge to find the suitable impedance parameters values.

To investigate the ability of the proposed learning framework to automatically find the suitable parameters values as the task changes, two different kinds of changes are included in the simulation. First, two different shapes (trajectories) will be used for training; a square and a circle. Second, surfaces of different materials will be put in the simulations. These two variations in simulations are detailed in the next two subsections.

4.1.2 Trajectory Types

For the trajectory of a circle centered at the center of a wall placed in the y-z plane and has a center at (x_0, y_0, z_0) as in Fig. 4.2:

$$\mathbf{P}_C^I(t) = \left[x_0 + \Delta, y_0 + \cos(\frac{vt}{r}), z_0 + \sin(\frac{vt}{r}) \right] \tag{4.2}$$

where r is the radius of the drawn circle and v is the lateral speed of the drone. A circle training episode would then last for $0 \leq t \leq \frac{2\pi r}{v}$, equivalent to drawing a circle once.

The trajectory of a square centered around the center of the same wall is defined in a piecewise way:

$$\mathbf{P}_C^I(t) = \begin{cases} [x_0 + \Delta, y_0 + r, z_0 + vt] & \text{if } 0 \leq vt < r, \\ [x_0 + \Delta, y_0 + 2r - vt, z_0 + r] & \text{if } r \leq vt < 3r, \\ [x_0 + \Delta, y_0 - r, z_0 + 4r - vt] & \text{if } 3r \leq vt < 5r, \\ [x_0 + \Delta, y_0 - 6r + vt, z_0 - r] & \text{if } 5r \leq vt < 7r, \\ [x_0 + \Delta, y_0 + r, z_0 - 8r + vt] & \text{if } 7r \leq vt \leq 8r, \end{cases} \tag{4.3}$$

where r is half side of the square. A square training episode would last for $0 \leq t \leq \frac{8r}{v}$ to draw a square once. Note that for the same speed v and radius/half-side r, drawing a square takes a larger time ($T = 8r/v$) than drawing a circle ($T = 2\pi r/v$). Drawing a square is also a harder task than drawing a circle because of the four discontinuities in the derivative of the trajectory. These discontinuities would cause large errors if the impedance parameters are not tuned correctly.

4.1.3 Surface Types

In addition to varying the training tasks by using two different drawing shapes, the tasks will be varied by changing the friction coefficients between the end-effector of the drone and the wall to simulate different drawing scenarios. Four different sliding friction coefficient μ values are used to represent four different end-effector and wall materials as illustrated in Fig. 4.3.

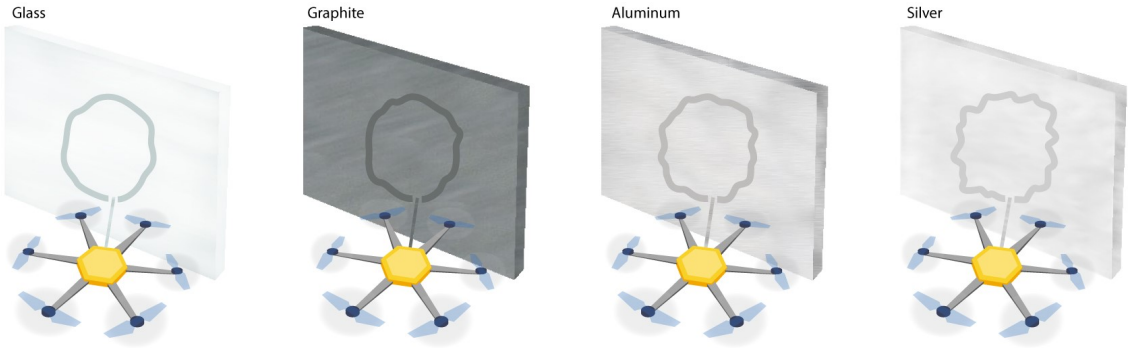


Figure 4.3: The four different friction scenarios used in simulation.

The coefficients values are shown in table 4.1. As the sliding friction increases, the drawing task becomes more challenging and it becomes more probable for the impedance controller, without proper fine-tuning, to give a bad performance as indicated in Fig. 4.3 (drawn circles get worse as the friction increases).

Table 4.1: Different friction values used in simulations [1].

μ	End-Effector Material	Wall Material
0.40	Glass	Glass
0.80	Graphite	Graphite
1.16	Aluminum	Aluminum
1.40	Silver	Silver

4.2 Reward Shaping

Reward is one of the most essential and effective parts of the proposed learning framework. Yet, it is also one of the hardest challenges. The reason is that handcrafting a reward is considered an art rather than a solid science [29]. A good reward often makes the difference between a tractable learning problem and an intractable one as will be discussed in section 5.1.

There are no specific guidelines for designing a good reward. However, one of the most important notices that should be kept in mind is that the reward signal is not the place to tell the learning algorithm *how* to achieve a goal, but *what* to achieve [29]. In other words, the reward should not *encourage* specific subtasks but only the general goal. The learning algorithm should then use this reward to discover the necessary steps or actions to reach the end goal without being restricted by hard-coded subtasks.

This section first discusses how a reward function was designed for the sliding/drawing tasks considered in this chapter. It then presents a novel idea to condition reward functions. The idea is especially suitable for the proposed learning framework where the number of dimensions of the learning problem as well as the nature of the encountered reward functions is largely varying.

4.2.1 Designing the Reward Function

In the learning problem at hand, we consider a simple goal: maximizing the accuracy of drawing while minimizing the used impedance gains. Excessively large gains are associated with higher motor speeds and hence higher power consumption. They are also associated with a more aggressive interaction behavior which is highly discouraged. However, higher gains (especially proportional ones) also increase the accuracy of tracking the setpoint, which means that maximizing the accuracy actually contradicts minimizing the impedance gains. The reward function should manage this conflict.

To design a reward function that represents the desired goal, translational and rotational errors throughout a training episode have to be calculated. At each time t, $\mathbf{P}_E^I(t)$ and $\mathbf{R}_E^I(t)$ the current position and orientation (as a rotation matrix) of the end-effector with respect to the inertial frame are received. Then:

$$\mathbf{E}_t(t) = \mathbf{P}_E^I(t) - \mathbf{P}_C^I(t) \tag{4.4}$$

$$\mathbf{E}_r(t) = \frac{1}{2}\left(\mathbf{R}_C^I(t)^\top \mathbf{R}_E^I(t) - \mathbf{R}_E^I(t)^\top \mathbf{R}_C^I(t)\right)^\vee \tag{4.5}$$

where $\mathbf{E}_t(t)$ is a 3D vector representing the position error along the three axes in the Euclidean space at time t, $\mathbf{E}_r(t)$ is also a 3D vector representing the rotational error along the three rotational axes at time t, and $\bullet^\vee$ is the map from $so(3)$ to $\mathbb{R}^3$ (map from 3D skew-symmetric matrices to 3D vectors) [23]. The translational root-mean-square error along each axis j for a training episode is then:

$$E_t^j = \sqrt{\frac{\sum_{t=0}^{T} \left(\mathbf{E}_t^j(t)\right)^2}{T}} \tag{4.6}$$

where $j \in \{x, y, z\}$ and T is total time needed to finish the training episode. The rotational root-mean-square error along each axis j is calculated in the same way. The root-mean-square error was used to make the reward more independent of the time duration of a training episode. Which allows rewards to be comparable between different tasks.

By calculating the translational and rotational error for each axis individually, one can design reward functions where an error along an axis is more significant than it is for the other axes. For the reward function designed for the task at hand, the error along each axis is given the same importance.

The used reward function for this task, in terms of the impedance parameters values $\mathbf{x}$, is defined:

$$f(\mathbf{x}) = -c_1 E_t - c_2 E_r - c_3 \frac{\text{sum}(\mathbf{x})}{\dim(\mathbf{x})} \tag{4.7}$$

where $E_t = E_t^x + E_t^y + E_t^z$ is the sum of the translational root-mean-square errors along the three axes, $E_r = E_r^x + E_r^y + E_r^z$ is similarly the sum of the rotational errors, $\dim(\bullet)$ is the number of elements in a vector, and c_1, c_2 and c_3 are weighting factors.

Remark Because the sensor data used to calculate the reward function is often noisy, we can only get noisy observations $y = f(\mathbf{x}) + \epsilon$ where ϵ is an unpredictable noise. Bayesian optimization is able to take this noise into account modeling it as normally distributed around $f(\mathbf{x})$ with a zero mean and variance σ^2 (also called the noise hyperparameter). σ^2 has to be set properly depending on the anticipated noise in the received observations. The used value of σ^2 and how it was selected in simulations is discussed in section 4.3.2.

Choosing the Weights

For this reward function, as it is typically the case for many reward functions seen in literature, the units of the individual parts of the function are different. So, the definition of the function has no direct physical meaning. The weights have to balance the difference in units and set the priorities of optimization. For all simulations done in this chapter, $c_1 = 100$, $c_2 = 150$ and $c_3 = 0.15$.

In this reward function, the translational error is *penalized* equally for all axes by including it as a negative term. Translational error is inevitable (after all, the impedance controller only moves the drone in response to errors). However, a large translational error indicates that the impedance controller is too compliant such that tracking is not accurate, or that the controller is too stiff in the direction perpendicular to the wall increasing the interaction force and restricting the movement of the drone.

The rotational error is also penalized, yet with a higher weight of 150. This is for two main reasons. First, the range of rotational error is much smaller than that of translational error because of the difference in units. Second, since the task doesn't involve rotating the drone, rotational error is more discouraged. Rotational error is also inevitable. Although the command trajectory doesn't rotate the drone throughout the task, rotational error arises from the friction force exerted by the wall on the end-effector causing the drone to tilt in the opposite direction of that force. A large rotational error, however, maybe the result of a very compliant behavior of the controller in the axes along the wall (y and z). It can be also the result of oscillating behavior of the drone experienced with some combinations of impedance parameters values.

Finally, the reward function also penalizes the average value of the controlled (or learned) impedance parameters $\bar{\mathbf{x}} = \text{sum}(\mathbf{x})/\dim(\mathbf{x})$ to force the learning agent to try to minimize the values of the controlled impedance parameters for the reasons mentioned above. The penalization, however, is done with a small weight of 0.15. Intuitively, this weight determines the amount of decrease in the translational and rotational error that would justify a certain increase in the average value of the impedance parameters. With the used weights, assuming the E_r doesn't change, a decrease of $c_3/c_1 = 0.0015$ in the translational error E_t justifies an increase of 1 in the average value of the learned parameters $\bar{\mathbf{x}}$. Increasing c_3 means that more significant decrease in the translational and rotational errors is required to justify a certain increase in $\bar{\mathbf{x}}$. Hence, this weight is used to manage the conflict between minimizing the errors and minimizing the selected values for impedance parameters.

4.2.2 Conditioning the Reward Function

The range of the reward function suggested in the previous section is very difficult to predict in advance. This is inconvenient when using Bayesian optimization. For instance, the meaning the hyperparameter ξ of the probability of improvement (PI) acquisition function is directly related to the range of the optimized function as can be seen in section 2.4.1. So, the same value for ξ can lead to explorative behavior for one reward function but an exploitative behavior for another one with a much larger range. The same problem arises with the noise hyperparameter σ^2.

To solve this issue, the author suggests limiting the range of the target function between zero and one using the Sigmoid logistic function defined:

$$S(x) = \frac{1}{1 + e^{-l(x-m)}} \tag{4.8}$$

where m is the location of the midpoint of the function and l is the steepness of the curve. Logistic functions are shown in Fig. 4.4 with different values for l. Logistic functions have been used in different areas in machine learning. Most notably, in logistic regression and as the "activation" function for neural networks.

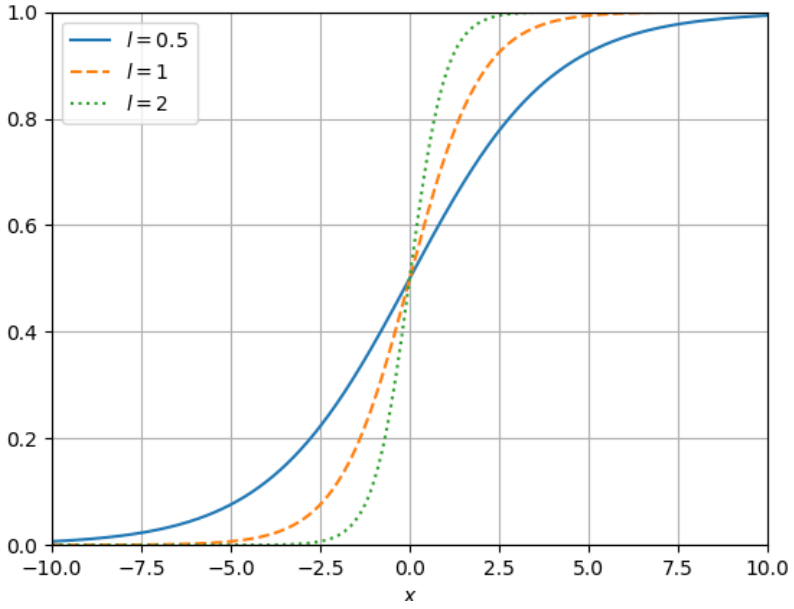


Figure 4.4: The logistic function with $m = 0$ and different values for l.

The domain of a logistic function is $\mathbb{R}$ and the range is $(0,1)$. Hence, inputting the reward function (4.7) to a logistic function will always result in a value between 0 and 1. However, it can be seen from Fig 4.4 that the sensitivity of the logistic function decreases significantly as the input goes away from the midpoint m. This simply means that, if the midpoint is in the wrong place, changes in the reward function might not be noticed at all after it is inputted to a logistic function. This leads to concluding that, for a successful employment of logistic functions with Bayesian optimization, a justifiable way for selecting the midpoint must be found.

A reasonable solution is to use the initial observations to find a suitable midpoint. Specifically, the maximum received reward in the initial points is considered the midpoint of the used logistic function. Consequently, Algorithm 1 is modified with the highlighted modifications in Algorithm 2 below.

Algorithm 2: Bayesian Optimization with a Logistic Function

1 Get w initial observations $\mathscr{D}_{1:w} = \{(\mathbf{x}_i, y_i)\}_{i=0}^{w}$
2 Calculate the midpoint of the logistic function $m = \max(y_{1:w})$.
The logistic function is then: $S(x) = 1/\left[1 + e^{-l(x-m)}\right]$
3 Transform the initial observations: $\mathscr{D}_{1:w} = \{(\mathbf{x}_i, S(y_i))\}_{i=0}^{w}$
4 **for** $n = w, w+1, w+2 \ldots$ **do**
5 Build/update the surrogate statistical model $M(S(f(\mathbf{x}))|\mathscr{D}_{1:n})$
6 Select $\mathbf{x}_{n+1}$ by optimizing the acquisition function $\alpha(\mathbf{x}|\mathscr{D}_{1:n})$
7 Obtain a (noisy) new observation at $\mathbf{x}_{n+1}$: $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$
8 Augment the data $\mathscr{D}_{1:n+1} = \{\mathscr{D}_{1:n}, (\mathbf{x}_{n+1}, S(y_{n+1}))\}$

With this modification, it is necessary from now on to discriminate between:

- $f(\mathbf{x})$: The black-box reward function (the function that gives a performance measure for a selection of impedance parameters values). We get a noisy observation of this function $y = f(\mathbf{x}) + \epsilon$.

- $S(f(\mathbf{x}))$: The target function (the function being optimized by Bayesian optimization). We get a noisy observation of this function $S(y)$ (the observation of the reward function inputted to a logistic function).

The surrogate model built by the Gaussian process is for the target function $S(f(\mathbf{x}))$ as indicated in line 5. Because the logistic function is monotonic and always increasing (has a positive derivative everywhere), however, optimizing it is equivalent to optimizing the original reward function.

Similar to the definitions in (3.4) and (3.5), we define:

- S_n^+ the maximum obtained value of the target function so far

$$S_n^+ = \max S(y_{1:n}) \tag{4.9}$$

- $\bar{S}_n$ the average of all obtained values of the target function so far

$$\bar{S}_n = \sum_{i=1}^{n} S(y_i)/n \tag{4.10}$$

The interpretation of the graphs of these two measures as the number of iterations increases is very similar to the interpretation of the corresponding y_n^+ and $\bar{y}_n$ discussed in section 3.2.3.

Note that, regardless of the value of the maximum reward in the w initial observations, the maximum seen value in the initial observations for the target function S_w^+ will be 0.5 after executing line 3 in Algorithm 2. This means that S_n^+ in the iteration n only represents the maximum seen *improvement* so far over the maximum seen reward in the initial observations. So, the exact value of the maximum received reward so far y_n^+ (defined in Eq. (3.4)) cannot be inferred from S_n^+ without knowing the value of the maximum seen reward in the initial observations y_w^+.

Considering that the maximum received reward during initialization y_w^+ represents the initial knowledge about the maximum of the target function, setting the midpoint of the logistic function at it means that the sensitivity to changes in the reward function will be greater around that initial knowledge. Looking to Fig. 4.4, it can be realized that this makes the steepness hyperparameter l in Eq. (4.8) control the amount of the desired increase in the reward with respect to the initial knowledge. Smaller values of l will encourage finding rewards much higher than the maximum reward in the initial observations (if such higher rewards actually exist). Greater values of l will make the optimization *content* with smaller improvement to the initial knowledge. If reaching the global maximum is the goal, it is a good idea to change l to a smaller value if it was found, during optimization, that the value of the target function goes very close to 1 to prevent the optimization from getting stuck in local maxima.

To summarize, the presented modification to Bayesian optimization gives two main advantages.

1. It limits the range of the target function between 0 and 1, allowing easier generalization of the meaning and use of acquisition functions hyperparameters as well as the noise hyperparameter.

2. It concentrates the sensitivity of the target function around the initial knowledge (represented by the maximum reward in the initial observations). This makes it easier to quickly find points that provide improvement over the initial knowledge as will be confirmed by the results in section 4.4.3.

As a bonus, this modification also gives the user a hyperparameter to limit the amount of improvement over the initial knowledge (if that was ever needed).

4.3 Experimental Setup

This section describes the procedures and settings of learning used in the experiments. The goal is to provide a recipe for allowing conducting similar experiments in simulation or in the physical world.

4.3.1 Learning Procedures

To successfully allow a drone to learn the suitable impedance parameters values for the sliding interaction tasks discussed in this chapter, a number of procedures are followed every time the black-box reward function $f(\mathbf{x})$ is evaluated (the custom trajectory is run) for an examined set of impedance parameters values $\mathbf{x}$, whether this evaluation was for getting initial observations for Bayesian optimization (line 1 in Algorithm 1) or during training episodes (line 5 in Algorithm 1). These procedures are detailed here.

Custom Trajectory Initialization

Before starting learning, the user has to provide a set of default values for the controller's parameters ensuring stable free-flight performance.

To insure that the received reward (performance measure) after each episode (run of the custom trajectory) is independent and repeatable, extra care has to be taken in initializing the trajectory at the start of the episode. For the task at hand, the following procedures are done:

1. All impedance parameters are set on the default values (suitable for free-flight) provided by the user.

2. The end-effector of the drone is commanded to go to an initial configuration determined by the start point of the custom trajectory, but without pressing on the wall[1]. This initial configuration is then determined by $\mathbf{P}_C^I(0)$ and $\mathbf{R}_C^I(0)$ with $\Delta = 0$.

   The drone is moved to this initial configuration using the default impedance parameters values to ensure that it is initialized in the same way each episode. Initializing with different impedance parameters values might result in small (but effective) differences in the starting configuration.

3. After sensor data has confirmed that the drone is fixed and stable at the initial configuration, the impedance parameters are changed from the default values to the values being examined. The drone then starts following the custom trajectory $\mathbf{P}_C^I(t)$ and $\mathbf{R}_C^I(t)$ (involving also pressing on the wall with nonzero Δ) for $0 \leq t \leq T$ to determine the performance of the impedance controller under these examined parameters values.

After initialization has finished and over the course of the whole episode ($0 \leq t \leq T$), all relevant sensor data are stored for the calculation of the episodic reward after the custom trajectory has finished (as discussed in section 4.2.1).

Failure Detection and Handling

To minimize the possibility of crashes and dangerous performance during exploration of the impedance parameter space in learning, it is necessary to equip the system with a failure detection and handling capability. In this case, all relevant sensor data during a training episode will be monitored and used to detect if the examined parameters values have caused instability or dangerous performance for the drone. In case such failure is detected the following procedures are followed:

[1] The initial configuration does not include pressing (applying a force) on the surface to eliminate the movement restriction imposed by this press and allow the drone to achieve the highest possible accuracy in positioning at this initial configuration.

1. The drone is stopped immediately from following the custom trajectory.
2. The impedance parameters are set on the default free-flight values given by the user.
3. The drone is then be commanded to be in a safe configuration away from the wall.
4. A very small reward, or a largely negative *penalty* predetermined by the user (we call it the *failure penalty*) is outputted as $f(\mathbf{x})$ indicating that the performance of the drone with the examined impedance parameters values $\mathbf{x}$ was highly undesirable.

Being able to automatically detect failure will greatly reduce the need for human supervision during learning and hence decrease the learning cost. As failure detection techniques are beyond the scope of this thesis, it is left, in the current implementation to the human user who must supervise the learning and when necessary press 'FAILURE' button in the learning GUI (described in section B) to trigger the failure handling procedures above.

Remark Note that failure detection and the implemented failure handling procedures are especially important in practical experiments with a real physical drone. In simulation, however, it is not a problem if the drone crashed or had a very bad performance during a training episode. In this case, without even stopping the episode and sending a failure penalty as the value of $f(\mathbf{x})$, the normally calculated reward (from the stored sensor data) will produce a very low value for $f(\mathbf{x})$ because the performance deviated from the learning goal embedded in the reward calculation. That is why in all performed simulations shown in this chapter, the failure procedures (invoked by pressing 'FAILURE' button in the learning GUI) were never actually used to minimize the need to supervise the simulations[2].

4.3.2 Learning Initialization

This subsection goes through the common decisions made to initialize learning for all simulated interaction tasks in this chapter.

Impedance Parameters for Learning

As noted in Appendix A, the complete number of impedance parameters reaches 27. The number of controllable parameters is 12 as mentioned in section 4.1. Depending on the interaction task, some parameters are relevant and others can be just fixed at zero or other suitable values. It is then the responsibility of the user to select the relevant impedance parameters values to be learned as well as the range of values for each parameter as discussed in section 2.5. This represents an additional way of incorporating human knowledge in the learning. Selecting more parameters to learn than necessary would increase the number of dimensions of the problem making it unnecessarily harder.

We then distinguish between different kinds of parameters in the learning process:

1. **Controlled parameters**
   These are the parameters Bayesian optimization controls their values as inputs to the black-box reward function $f(\mathbf{x})$ during learning. The number of controlled parameters determines the number of dimensions of the learning problem (the number of elements in $\mathbf{x}$). By providing the bounds of these parameters, the user effectively specifies the domain of the black-box reward function.
2. **Linked parameters**
   These are parameters constrained to have the same value of another controlled parameter. This is an effective way of reducing the dimensionality of the learning if the user

[2] That way, I was also able to run simulations during sleeping time.

knows that two impedance parameters should have the same value. Bayesian optimization will directly control only the values of controlled parameters during learning, but each time a controlled parameter value is changed, the value of any other parameter linked to it will change in the same way.

3. **Fixed parameters**
   These are the parameters the user wishes to keep with specific values throughout the learning because the interaction task does not require learning them.

For the impedance controller used in simulations, a set of default values where chosen for the controllable parameters:

$$\begin{aligned} \text{diag}(\mathbf{K}_r) &= [10, 10, 10] \\ \text{diag}(\mathbf{K}_t) &= [6, 6, 6] \\ \text{diag}(\mathbf{K}_d) &= [5, 5, 5, 8, 8, 8] \end{aligned} \tag{4.11}$$

where diag(•) is the diagonal of a matrix. All other parameters are set on zero for the current implementation as mentioned in section 4.1.

In the simulations, the learning was first done for a simple 2D problem in which the spring translational stiffness parameters where learned. In particular, the controlled parameters were the stiffness along the x-axis[3] $K_t^{1,1}$ and the stiffness along the y-axis $K_t^{2,2}$. In addition, the stiffness along the z-axis $K_t^{3,3}$ was linked to $K_t^{2,2}$ such that both represent one parameter in learning. All other impedance parameters were fixed at the default free-flight settings in Eq. (4.11).

The learning was also done for a 4D problem, where, in addition to the parameters above, the translational damping parameters were learned. So, $K_d^{4,4}$ and $K_d^{5,5}$ the translational damping along x- and y-axes respectively were added as controlled parameters, and also the damping along the z-axis $K_d^{6,6}$ was linked to that along the y-axis. Linking the translational stiffness and damping parameters along the z-axis to these along the y-axis comes from the fact that both axes are parallel to the wall. So, intuitively, the optimal impedance behavior in both directions should be similar.

Initial Observations

As an initial step in learning, the user should provide w initial observations (line 1 of Algorithm 1) at w different locations. Through these initial observations, the user can give *hints* for Bayesian optimization about possible locations of the high rewards as mentioned in section 2.3.2.

For each selected location $\mathbf{x}$, the user runs an episode for the drone to get an observation of the value of $f(\mathbf{x})$. The initial observations are given to Bayesian optimization as a list of tuples $\mathscr{D}_{1:w}$ where $\mathscr{D}_i = (\mathbf{x}_i, y_i)$ and $y_i = f(\mathbf{x}_i) + \epsilon_i$ is a noisy observation.

Determining the number w as well as the locations of initial observations is the responsibility of the user and should be based on the application. Higher number of initial observations might be needed as the number of dimensions of the learning problem increases. However, what is more important than the quantity is the quality of the provided observations. The ideal way of providing initial observations is by using expert knowledge or intuition. For instance, the intuition behind selecting impedance parameters values for a sliding task was discussed in section 4.1.1. The user can use this intuition by selecting impedance parameters values $\mathbf{x}$ consistent with it. It is also a good idea to provide initial observations around areas with high chances of getting low rewards so that learning does not need to go to these areas to find out.

Ten initial observations were provided for both the 2D and 4D problems. The locations of these initial observations (the values of the impedance parameters) where selected to reflect the ini-

[3] superscripts are row-based indices.

tial knowledge (or intuition) mentioned in section 4.1.1. The locations cover areas with high chance of producing large rewards (according to intuition) as well as areas with high chance of producing very low rewards so that the learning agent has good initial knowledge about different parts of the parameters space. The locations of the initial observations provided for the 2D learning problems are shown in Fig. 4.5. All learning problems with the same number of dimensions and the same controlled parameters were provided with a single set of locations of initial observations for the sake of comparability. After providing ten initial observations, Bayesian optimization was run for 40 more iterations for each learning problem.

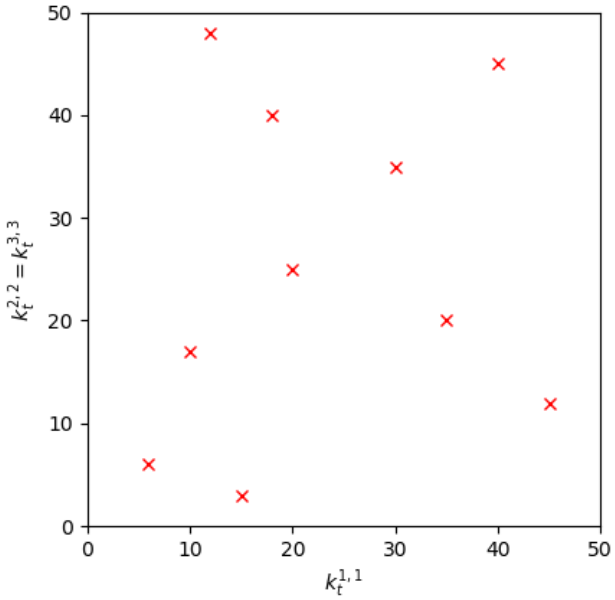


Figure 4.5: Locations of the initial observations for the 2D problems.

Remark While the *locations* $\mathbf{x}_{1:w}$ of the provided w initial observations were kept the same for all learning problems with the same number of dimensions, the values of the observations at these locations $y_{1:w}$ are different for each learning problem, since the black-box reward function is different for different tasks (different trajectories or frictions).

Learning Settings

Based on the insights gained in chapter 3, PI was selected as the used acquisition function. The hyperparameter ξ was set to 0.01. This value was selected based on a number of trial-and-error trainings. It was found to offer a moderately explorative performance with the black-box reward functions at hand.

In all the scenarios presented in this chapter, the lower and upper limits of each selected controlled impedance parameter were set to be 0.01 and 50 respectively. The lower limit is above zero to prevent Bayesian optimization from setting any of the parameters to zero or lower during training as this might cause instability. The upper limit is sufficiently large such that it is known that the optimal values of the parameters will be within the range.

Local optimization of acquisition functions as presented in section 3.4 was used. Consequently, the d (distance) hyperparameter was set to be 0.05. Given the selected lower and upper limits of the controlled parameters mentioned above, this essentially means that the learning agent can change a controlled impedance parameter with a maximum difference of approximately $\pm 0.025 \times 50 = \pm 1.25$ per iteration. This is to make learning safer by providing exploration an *early warning* as mentioned in section 3.4.

The Gaussian process noise hyperparameter σ^2 was selected to be 10^{-4}. This was done by observing the repeatability of the reward function during simulation (i.e. setting the impedance parameters to certain values and doing the simulation several times to see how the resultant reward varies). Thanks to the trajectory initialization procedures discussed in section B, the repeatability of the reward function is very high and hence the noise is minimal. In a practical situation, however, where many other factors (like aerodynamics) contribute to the reward, the noise hyperparameters should certainly be set to a larger value.

Finally, the method described in section 4.2.2 was used. The logistic function hyperparameter l was set to 0.5. This means that the sensitivity of the function will be high for changes from the initial knowledge up to 7.5 units of the reward function. For all experiments shown in section 4.4, the target function $S(y_t)$ never crossed 0.97, indicating that the chosen value for l doesn't limit reward improvement as mentioned in section 4.2.2.

Table 4.2 summarizes the selected values for the discussed settings in this section.

Table 4.2: Summary of the used learning settings in the simulations.

Setting	Notation	Selected Value
Acquisition Function		PI
PI hyperparameter	ξ	0.01
Domain of $f(\mathbf{x})$ along a dimension i	χ_i	$(0.01, 50)$
distance in local optimization	d	0.05
GP noise hyperparameter	σ^2	0.0001
Logistic function steepness	l	0.5

4.4 Results and Discussion

4.4.1 Drawing a Circle

For all the results in this subsection, the task was to draw a circle with radius $r = 1m$, velocity $v = 0.2m/s$ and depth $\Delta = 0.05m$.

Circle 2D Learning Results

The learning curves for the circle 2D problems with different friction coefficient μ values are shown in Fig. 4.6. The curves in the upper graph represent the learning rates. they show the improvement in the optimized target function as the number of iterations increases. Note that S^+ curves do not say anything about the value of the black-box reward function. They only show the amount of improvement over the maximum seen rewards in the initial observations as mentioned in section 4.2.2. It can be seen that amount of improvement over the initial knowledge increases as the friction coefficient increases.

The middle graph in Fig. 4.6 can explain that more. It shows the corresponding maximum seen value of the black-box reward function. It can be seen that, as μ decreases, the maximum seen reward in the initial observations becomes higher y^+_{10}. This indicates that the initial observations provided better performance for easier sliding tasks (with lower μ). At the end of the 40 Bayesian optimization iterations, the maximum obtained rewards converged to almost the same value of around -7.8 for all μ values.

The bottom graph in Fig. 4.6 shows the average of all received values of the target function as the number of iterations increases. The slope of the average is almost always positive for all μ values indicating that the agent did not fall into areas of small rewards (areas of bad performance) during learning as mentioned in section 3.3. This means that the learning was relatively safe, which is one of the main advantages of using local optimization for the acquisition function as discussed in section 3.4. The effect of using global optimization will be seen in section 4.4.3.

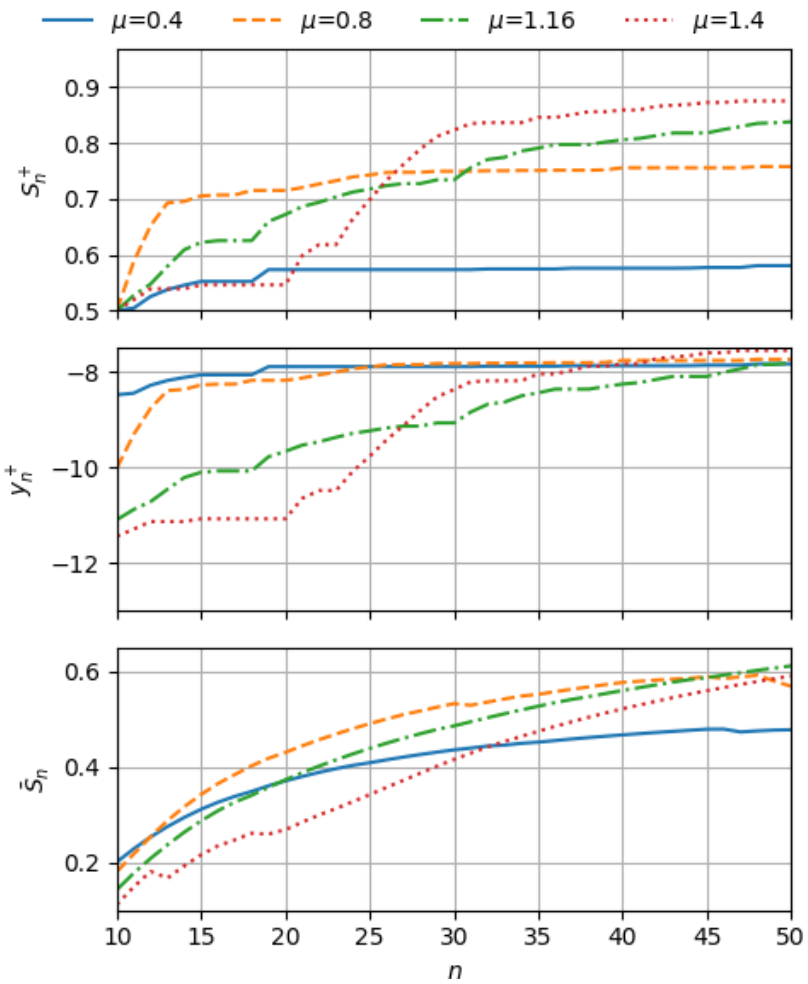


Figure 4.6: Learning curves of the 2D circle problems.

Remark The curves of the average of all received values of the target function are similar for all learning tasks in this section and hence these curves are not shown again for the other tasks below.

Circle 4D Learning Results

Figure 4.7 shows the learning rates (top) and the corresponding values of the maximum seen rewards (bottom) of the circle 4D problems with different friction coefficient values. It can be seen that the improvement is generally faster than in the 2D case. Most of the improvement happened in the first 15 Bayesian optimization iterations after the 10 initial observations. However, similar to the 2D case, the amount of improvement increases as the friction coefficient increases.

Looking to the bottom graph in Fig. 4.7, it can be confirmed that, like in the 2D case, the reason for the improvement trend is that the initial observations produce better rewards for smoother surfaces. At the end of the 40 Bayesian optimization iterations, the maximum found reward converges to almost the same value of -7.6 for all values of μ. This is slightly higher than what was reached in the 2D case. The reason is that, with more degrees of freedom in the 4D case, the agent was able find impedance parameters that produce higher accuracy as will be shown further below.

The circles drawn by the drone at different stages of learning in the 4D problem with $\mu = 1.4$ are shown in Fig. 4.8. It can be seen that after training (40 iterations of Bayesian optimization),

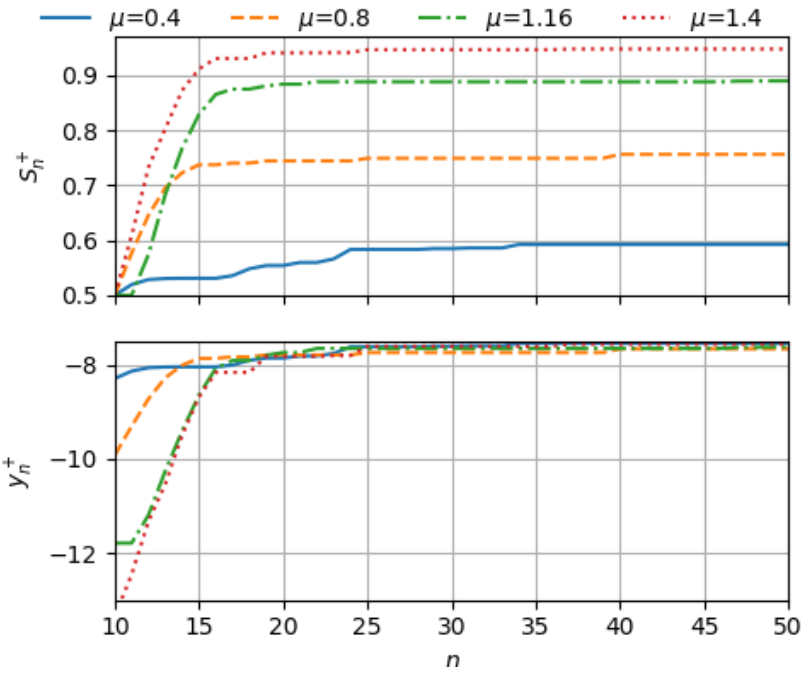


Figure 4.7: Learning curves of the circle 4D problems.

the agent found the impedance parameters allowing the drone to draw a very accurate circle in such a rough sliding task.

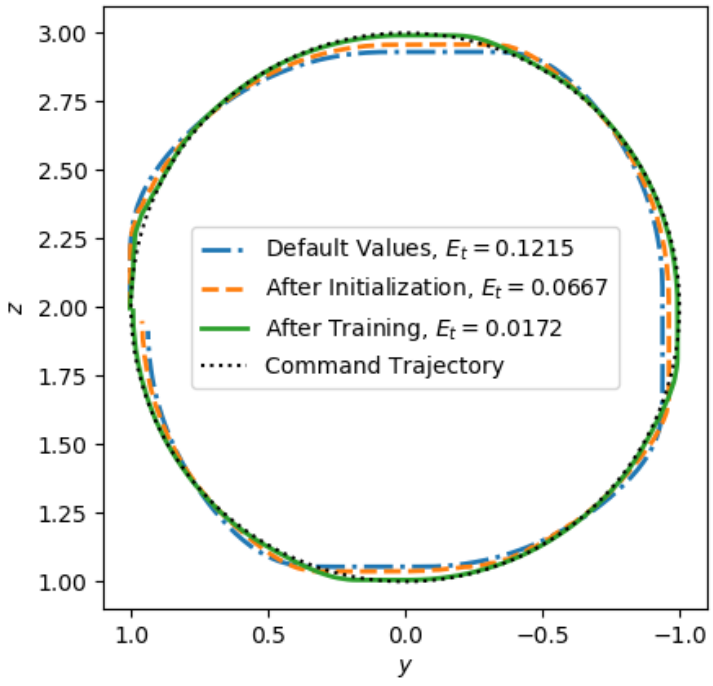


Figure 4.8: The circles drawn by the drone at different stages of learning in the 4D problem with $\mu = 1.4$.

Deeper Look at the 2D and 4D Learning at Different Stages

Table 4.3 gives deeper insights into how the different parts of the reward function change at different learning stages of the circle 2D and 4D problems. It shows the maximum reward (denoted y^+) seen for the default values in Eq. (4.11), within the initial observations, and among all training iterations. For each y^+, the table shows the corresponding value of the translational

error E_t (multiplied by 100 to be easier to show) and the average of the controlled impedance parameters values $\bar{\mathbf{x}} = \text{sum}(\mathbf{x})/\dim(\mathbf{x})$.

Remark Note that the rotational error E_r is not included in the table as it did not improve throughout training, which is logical because the rotational stiffness and damping impedance parameters were fixed at the default values in the tasks presented in this subsection. Including the rotational error E_r in the reward function was necessary, however, to penalize the cases where selection of certain values for translational stiffness and damping impedance parameters leads to an oscillating behavior during interaction as mentioned in section 4.2.1.

Table 4.3: Elements of the rewards at different learning stages for the 2D and 4D circle problems.

		$\mu = 0.40$			$\mu = 0.80$			$\mu = 1.16$			$\mu = 1.40$		
		y^+	$100E_t$	$\bar{\mathbf{x}}$	y^+	$100E_t$	$\bar{\mathbf{x}}$	y^+	$100E_t$	$\bar{\mathbf{x}}$	y^+	$100E_t$	$\bar{\mathbf{x}}$
Default	2D	-8.83	3.36	6.00	-12.08	6.58	6.00	-15.39	9.88	6.00	-17.71	12.15	6.00
Values	4D	-8.98	3.36	7.00	-12.24	6.58	7.00	-15.56	9.88	7.00	-17.80	12.15	7.00
Initial	2D	-8.48	1.84	13.50	-10.02	3.34	13.50	-11.10	1.78	30.00	-11.45	2.13	30.00
Points	4D	-8.29	1.72	13.00	-9.92	3.31	13.00	-11.79	5.19	13.00	-13.34	6.67	13.00
After	2D	-7.83	2.16	7.30	-7.74	1.96	7.95	-7.81	1.82	9.31	-7.55	2.14	5.59
Training	4D	-7.54	1.83	7.89	-7.66	1.74	8.70	-7.62	1.73	8.30	-7.56	1.72	7.98

Considering the rewards resulting from using the default values of impedance parameters in Eq. (4.11) (the two first rows), it can be seen that they get lower as the friction coefficient μ gets higher. The reason, as mentioned before, is that a drone with a stable free-flight performance does not need significant changes to its impedance parameters values to perform drawing with low friction. While the translational error for each friction value is the same for both the 2D and 4D problems (because the default values are the same regardless of the dimension of the problem), the rewards of the 4D tasks are slightly lower. This is because, for 4D tasks, the average of the controlled impedance parameters values is calculated for four parameters rather than two, making it higher (the default translational damping parameters are higher than the stiffness ones).

Looking at the maximum rewards in the ten initial observations (the two middle rows), it can be seen that the same trend of reward values persists. As the friction increases, the maximum seen reward in the initial observation decreases. We see also that the translational error in the 4D case is slightly lower than the 2D case for the two smaller friction coefficients and much higher for the larger friction coefficients. Notice also that, in the 2D case, the point that produced the highest reward among the initial points had much higher $\bar{\mathbf{x}}$ for the highest two friction coefficient values than for the lower two values.

After completing an additional 40 Bayesian optimization iterations (the two bottom rows), the maximum found reward significantly increased especially for higher friction coefficient values. Generally, the rewards achieved in the 4D problems are higher than those achieved in the 2D problems. This is not due to finding values with lower $\bar{\mathbf{x}}$, but because the agent was able to tune the addition parameters in the 4D problems to achieve significantly lower translational errors.

Analyzing the Learned Parameters Values in the 4D Problem

The parameters values that resulted in the highest rewards during the 4D circle training tasks are shown in Table 4.4. It can be seen that, in general, the results confirm the intuition that the impedance should be compliant in the direction normal to the wall (low values of $K_t^{1,1}$) and stiff in the directions parallel to it (higher values of $K_t^{2,2}$ and $K_t^{3,3}$) as discussed in section 4.1.1. It is noted also that the damping in the direction normal to the wall is selected to be almost equal to zero for all μ values. This is because, as the drone *presses* against the wall throughout performing the task, almost no damping is need in that direction. On the other hand, we see that the selected damping in the directions parallel to the wall decreases for higher values of μ.

This is due to the fact that more natural damping is provided by the environment as the friction increases. So, less damping needs to be produced by the controller.

Table 4.4: Learned parameters values for the circle 4D problem with different friction coefficients.

μ	$K_t^{1,1}$	$K_t^{2,2} = K_t^{3,3}$	$K_d^{4,4}$	$K_d^{5,5} = K_d^{6,6}$
0.40	4.955	11.302	0.010	15.279
0.80	5.481	22.921	0.019	6.374
1.16	3.642	22.828	0.020	6.701
1.40	3.317	22.747	0.060	5.811

Table 4.5 shows the rewards obtained by using the parameters values learned for a circle 4D problem with a specific friction coefficient μ value in circle 4D problems with other μ values. It can be seen (looking to each row) that using the values for a task with a certain μ on a different task produced suboptimal results with lower rewards and higher translational errors. Further, the reward decreases and the translational error increases as the difference in μ increases, indicating that learned parameters for a specific environment perform better in similar environments than in different ones.

Table 4.5: Results of using the learned parameters in the 4D circle problem with different friction coefficients.

Used for	$\mu = 0.40$		$\mu = 0.80$		$\mu = 1.16$		$\mu = 1.40$	
Learned for	y^+	$100E_t$	y^+	$100E_t$	y^+	$100E_t$	y^+	$100E_t$
$\mu = 0.40$	-7.52	1.83	-8.14	2.34	-9.30	3.53	-10.09	4.28
$\mu = 0.80$	-8.08	2.16	-7.66	1.74	-7.86	1.89	-8.21	2.33
$\mu = 1.16$	-8.23	2.37	-7.82	1.96	-7.62	1.73	-7.60	1.71
$\mu = 1.40$	-8.24	2.43	-7.83	2.02	-7.63	1.80	-7.56	1.72

Looking at each column, it can be also seen that the values that produced the best results for a task are those learned by training on the task itself. This confirms one of the biggest underline assumptions of this thesis that the optimal parameters values change depending on the environment. The table shows that, given the same initial knowledge, the proposed learning framework was able to automatically find the parameters values suitable for each task.

4.4.2 Generalizing Knowledge with a Square

Having trained the drone to find suitable impedance parameters values for drawing circles with different frictions, it is worth investigating if the learned parameters values are able to provide satisfactory performance for tasks involving drawing a different shape. In this subsection, the task of drawing a square is considered. All results shown here are for drawing a square with a half-side $r = 1m$, speed $v = 0.2m/s$ and $depth = 0.05m$. These settings are the same as the ones used for the circle tasks except that the perimeter of a square with half-side r is greater than the circumference of a circle with a radius r. We consider here the square 4D problems with the controlled and linked impedance parameters being the same as the ones mentioned in section 4.3.2.

Using the Parameters Values Learned for the Circle

Before training on the square tasks, the impedance parameters values found in the circle 4D learning (table 4.4) were applied to the square tasks (denoted by □) with the respective friction coefficient values as in Table 4.6. The previously shown results of using these parameters values in the circle tasks (denoted by ○) are also included in the table for comparability. It can be seen that the values produced lower rewards and higher translational errors in square tasks than in circle tasks. Note that a higher translational error (and hence a lower reward) is expected for

square tasks because of the involved discontinuities in the trajectory. However, it is still unclear form that table if the difference in rewards is completely due to the added difficulty or due to the fact that square tasks need completely different values for the impedance parameters.

Table 4.6: Learned parameters values for the circle 4D tasks applied to the square 4D tasks.

μ	Shape	y^+	$100E_t$	$\bar{\mathbf{x}}$
0.40	○	-7.52	1.83	7.89
	□	-8.74	2.47	
0.80	○	-7.66	1.74	8.70
	□	-8.96	2.23	
1.16	○	-7.62	1.73	8.30
	□	-8.91	2.24	
1.40	○	-7.56	1.72	7.98
	□	-8.90	2.28	

Square 4D Learning Results

To reach a conclusion about the aforementioned point, Bayesian optimization was run to find the suitable impedance parameters for the square 4D problems. The same ten locations of the initial observations used in the circle 4D tasks were used here. Figure 4.9 shows the learning curves. It can be seen from the top graph (showing the maximum seen value of the target function S_n^+) that the amounts of improvement exhibit a similar trend to these seen in the circle 4D case. However, the amounts of improvements to the initial knowledge are generally lower than the circle 4D counterparts.

Looking to the bottom graph in Fig. 4.9, it can be seen also that the maximum seen reward within the initial observations is higher for the two highest values of μ and lower for the two lowest values. At the end, the maximum found reward converges to almost the same value of around -8.80 for all values of μ. This is more than one unit lower than the maximum rewards reached in the circle 4D tasks. These differences between the square 4D learning and the circle

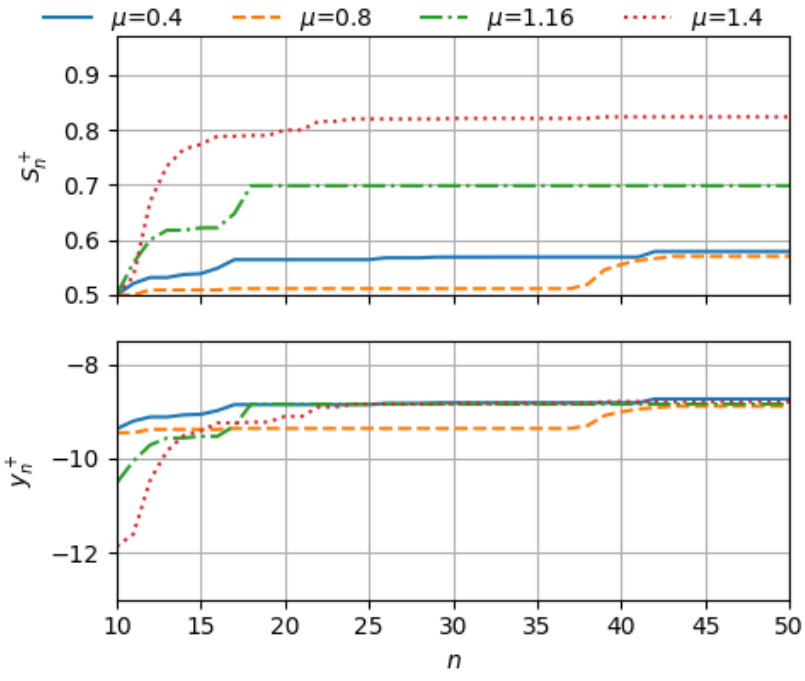


Figure 4.9: Learning curves of the square 4D problems.

4D learning can be attributed to the increased difficulty in the square tasks (due to discontinuities) decreasing the room for improvement and limiting the maximum reward that can be found.

The squares drawn by the drone at different stages of learning in the 4D problem with $\mu = 1.4$ are shown in Fig. 4.10. It can be seen that after training, the agent was able to draw a square more accurately. However, there are still (seemingly unavoidable) errors around the discontinuous parts of the trajectory.

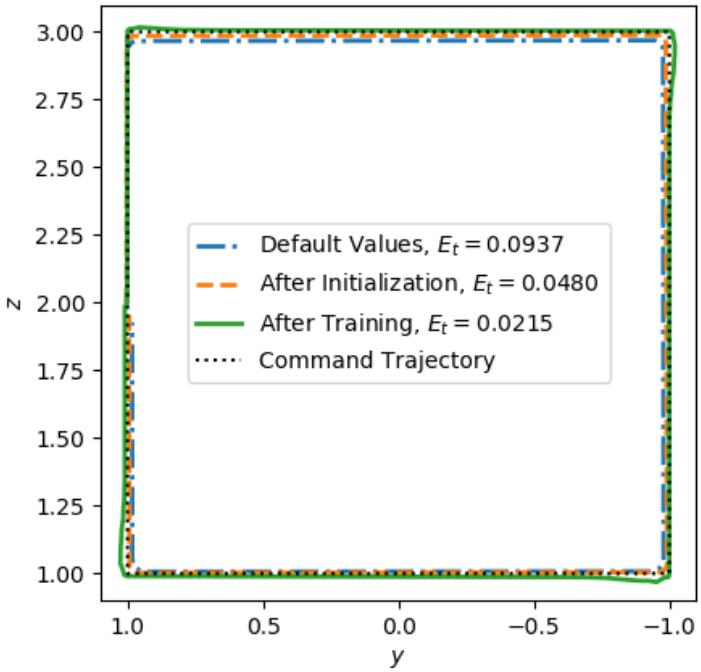


Figure 4.10: The squares drawn by the drone at different stages of learning in a 4D problem ($\mu = 1.4$).

Analyzing the Learned Parameters Values

Table 4.7 shows the parameters values that resulted in the highest rewards during the square 4D training tasks. Comparing these to the ones resulted from the circle 4D tasks (Table 4.4), it can be noticed that the values of the translational stiffness parameters ($K_t^{1,1}$, $K_t^{2,2}$ and $K_t^{3,3}$) are very similar, with the exception that the selected values of the stiffness in the direction normal to the wall $K_t^{1,1}$ are a little higher in square tasks than the corresponding ones in the circle tasks.

Table 4.7: Learned parameters values for the square 4D problem.

μ	$K_t^{1,1}$	$K_t^{2,2} = K_t^{3,3}$	$K_d^{4,4}$	$K_d^{5,5} = K_d^{6,6}$
0.40	7.342	13.151	3.283	14.389
0.80	5.222	20.031	3.258	8.396
1.16	5.660	21.033	2.080	10.869
1.40	4.290	22.856	0.040	8.665

The trends among the selected damping parameters values are also a little different. It can be seen that as μ decrease, higher values for the damping in the direction normal to the wall $K_d^{4,4}$ are selected, compared to almost zero for all values of μ in the circle tasks. For the damping in the directions parallel to the wall, the selected values are very similar to the ones selected for the circle tasks, yet slightly higher for higher values of μ. These changes are the way the learning agent adapted the impedance parameters to the square tasks trying to minimize the transla-

tional error around the discontinuous parts of the trajectory. Minimizing this error required higher stiffness normal to the wall and higher damping.

Using the Parameters Values Learned for the Square

To see how the performance obtained by the impedance parameters values learned for the square tasks differs from that of the values learned for the circle tasks, table 4.8 shows the results of using the learned parameters values of the square 4D tasks on the circle (denoted by ○) and square (denoted by □) 4D tasks. Comparing these results with the ones shown in Table 4.6, important conclusions can be drawn. Generally, it can be seen that the rewards resulting from impedance values obtained by training on a certain shape are greater than those resulting from impedance values obtained by training on the other shape. However, the difference between the two rewards is mostly small. This means that training for drawing one shape might be enough to find parameters values suitable for drawing different shapes.

Table 4.8: Learned parameters values for the square 4D tasks applied to the circle and square 4D tasks.

μ	Shape	y^+	$100E_t$	$\bar{\mathbf{x}}$
0.40	○	-7.84	1.80	9.54
	□	-8.73	2.20	
0.80	○	-7.71	1.72	9.23
	□	-8.89	2.19	
1.16	○	-8.09	2.00	9.91
	□	-8.84	2.19	
1.40	○	-7.70	1.72	8.96
	□	-8.79	2.15	

It can be said though that, if the interest is in generalizing knowledge, it is more beneficial to use the results of training for the harder task. For example, it can be seen that the parameters obtained from the square training (a harder task because of the discontinuities) were able to provide less translational errors for both the circle and square tasks than those obtained by circle training. The reason these parameters values produce lower rewards for the circle tasks although they are providing less translational errors is that they have higher average values $\bar{\mathbf{x}}$ that did not, according to the reward function, justify the decrease in the translational errors.

The results in tables 4.6 and 4.8 show one of the key advantages of the problem formulation suggested in the presented work over other formulations in literature (mentioned in section 1.1) that concern learning a *trajectory* of impedance parameters values (learning impedance parameters values as a function of time) for a certain task. A learned impedance values trajectory for drawing a square, for example, cannot be used for drawing a circle. With the suggested method, training does not have to be repeated for similar tasks provided that the environment is the same.

4.4.3 Learning without the Proposed Modifications

All the simulations presented in this section so far were done with the two novel modifications proposed in sections 3.4 and 4.2.2. In this subsection, results are shown confirming the significance of these modifications when learning impedance parameters for robotic drone systems.

The Effect of Not Using the Logistic Function

Figure 4.11 compares the learning curves for the circle and square 4D tasks without using a logistic function to the already seen learning curves obtained with using a logistic function as suggested in Algorithm 2.

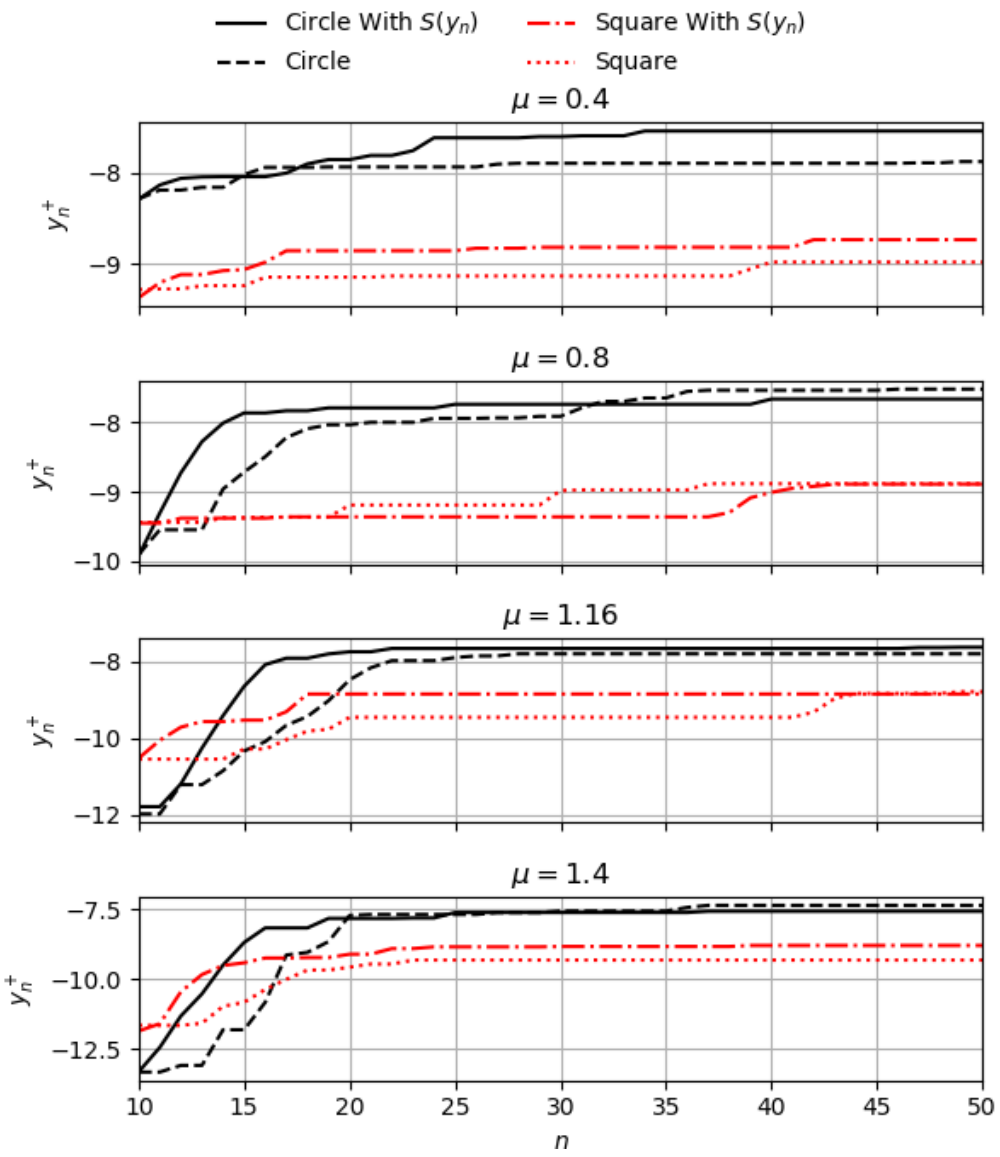


Figure 4.11: Learning rates of the circle and square 4D problems with and without a logistic function.

In the simulations done without using the logistic function modification, the same locations for the initial observations were provided and the same learning settings were used as described in section 4.3.2 except that Bayesian optimization was performed directly on the received rewards without inputting them to a logistic function as described in section 4.2.2.

It can be seen form Fig. 4.11 that, in almost all cases, using the logistic function modification resulted in reaching higher rewards in lower numbers of iterations confirming the statements made in section 4.2.2. Few exceptions exist. For the circle tasks with $\mu = 0.8$ and $\mu = 1.16$, using the logistic function modification resulted in getting higher rewards quicker. However, towards the end of the training (40 Bayesian optimization iterations), the learning curves without using the logistic function could reach a slightly higher point in a way that doesn't affect the general trend. Also, for the square task with $\mu = 0.8$ the learning curve is faster without using the logistic function, although reaching the same point at the end of the training. The reason why the learning curve was faster in this only case is that the range of the reward function in that case was suitable for the selected learning settings. However, as the range of the reward function changes from one situation to another, Bayesian optimization without using the logistic function was not able to maintain the same performance.

These results clearly show that the suggested logistic function modification allowed Bayesian optimization to generalize the meaning of learning settings among black-box reward functions of different natures and to concentrate the sensitivity of the target function around the maximum seen reward in the initial observations helping finding points with higher rewards quickly.

The Effect of Global Optimization

Figure 4.12 compares the learning rates (y_n^+ curves) and the average received rewards ($\bar{y}_n$ curves) of the circle 4D problems with a globally optimized acquisition function to the already seen results with a locally optimized acquisition function as suggested in section 3.4.

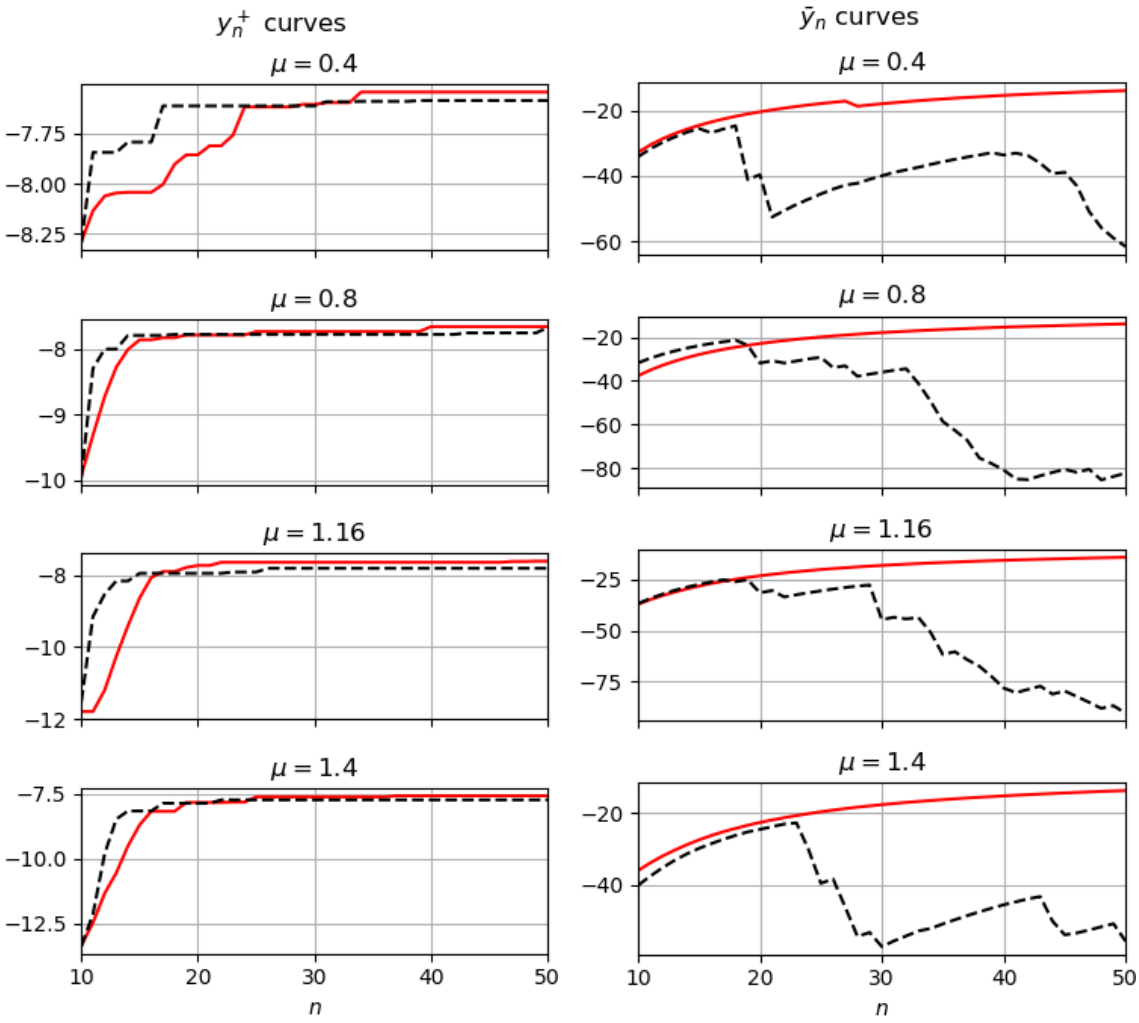


Figure 4.12: Learning rates and average received rewards of the circle 4D problems with locally optimized (solid) and globally optimized (dashed) acquisition function.

Again, for the simulations done with a globally optimized acquisition function, the same settings and initial points offered for the other 4D simulations, as discussed in section 4.3.2, were used (including using a logistic function) except that PI was optimized globally with the same default global optimization method mentioned in section 3.2.2.

Looking at the learning rates (y_n^+ curves), it can be seen that learning with a globally optimized PI was able to reach high rewards quicker than it is with a local optimization. However, in all learning problems, local optimization of the acquisition function was able to find higher rewards at the end. This is because, for a globally optimized PI, setting the hyperparameter ξ to 0.05 results in a more explorative behavior allowing quicker finding of places with higher

rewards. However, this increased exploration resulted in the search being unable to fine-tune and reach the locations of the highest rewards. Local optimization, while still benefiting from the exploration by setting ξ to 0.05, is able reach higher rewards slightly slower than global optimization.

What is more significant about local optimization is not its ability to find higher rewards, but its ability to make the learning much safer. Observing the averages of the received rewards during the different learning problems ($\bar{y}_n$ curves), it can be clearly seen that the search with local optimization almost always obtains rewards that are equal or higher than the already seen rewards ($\bar{y}_n$ are always increasing except for a minor exception for the learning problem with $\mu = 0.4$). This indicates that the search is safe as it doesn't fall in areas in the impedance parameters space with low rewards (corresponding to very unsatisfactory and dangerous performance or instability). This is because the search with local optimization is able to receive *early warnings* along paths leading to low rewards instead of jumping straight to them as discussed in section 3.4.2. On the other hand, the search with global optimization falls into areas with worse rewards than already obtained most of the time of the training, which indicates that the learning is very unsafe. Unless there is a very fast way of detecting failures, this training is disastrous if conducted with a physical drone in the real world.

4.5 Conclusion

In this chapter, the framework proposed in section 1.3 was validated through several simulations. The concentration was on sliding interaction tasks. It was seen that, even for this very simple task, it is difficult to tune the impedance parameters using mere human intuition or expert knowledge as the suitable parameters values depend on many unpredictable factors.

A reward function was designed to assess the performance of the drone's interaction under a set of selected impedance parameters values. It was shown that, often, the reward function tries to achieve contradicting goals and it should be designed to prioritize and manage the conflict between the subgoals of learning.

An issue was faced with the reward function that the range of its output cannot be predicted and changes from one situation (environment and task) to another. This makes it difficult to unify the used learning settings. To remedy this, a novel method was proposed to *condition* the reward function. The proposed method exploits the provided initial observations to seek improvement over them.

In describing the procedures followed during learning as well as the initial learning requirements needed to obtain effective learning results, it was emphasized that it is important to make the training episodes as independent as possible and to select the relevant impedance parameters for learning based on the task.

Finally, the chapter presented and discussed the simulation results. The results represent an empirical proof that the proposed learning framework is able to automatically find the suitable impedance parameters for different tasks and environments. Also, the results showed that the framework is able to generalize the learned parameters values to similar tasks. Additionally, the significance of the proposed modifications on Bayesian optimization was witness. It was seen that conditioning the reward signal with a logistic function made it possible to maintain a consistent performance using the same learning settings across different learning problems. It was also seen that locally optimizing acquisition functions significantly increased the safety of the learning.

5 Reinforcement Learning Point of View

This chapter presents another way of looking to the problem formulation in section 1.2 as a reinforcement learning problem, and to the proposed method in section 1.3 as a reinforcement learning technique. The importance of this different perspective is that it opens doors for benefiting from the vast literature of using reinforcement learning with robotics allowing the proposed framework to be extended in different ways in future research work.

The chapter begins with a brief background about reinforcement learning in section 5.1. Then, section 5.2 shows that the problem formulated in section 1.2 can be reformulated as a reduced version of the full reinforcement learning problem. Finally, section 5.3 argues that the proposed learning framework can be categorized as a reinforcement learning technique.

5.1 Background

In general, machine learning techniques can be classified into three broad categories: supervised learning, unsupervised learning and reinforcement learning [29]. In supervised learning, the learning *agent* is presented with a data-set of *labeled* examples, each contains a description of a situation as well as the correct classification of this situation or action to take when confronted by it. The objective of the learning is then to extrapolate this training data-set and be able to determine the correct classification or action to take for unseen situations. On the other hand, in unsupervised learning, the agent is given an unlabeled data-set and is required to find some hidden structure in it.

Reinforcement learning is different from both previously described categories. Instead of dealing with readily made data-sets, reinforcement learning is the learning by interactions with the environment. It is formalized by what is known as Markov Decision Process (MDP). A simple diagram showing the main components of an MDP is shown in Fig. 5.1. Based on the current state S_t of the system, the agent selects an action A_t. This action results in a *reward* R_{t+1} indicating how good the action was. The action also causes the system to move to a new state S_{t+1}. Both the new state and the reward are fed-back from the environment to the agent. The goal of the agent is to maximize the (long-term) reward received over the course of its life by finding a *policy* that determines what action should be taken given the state of the system.

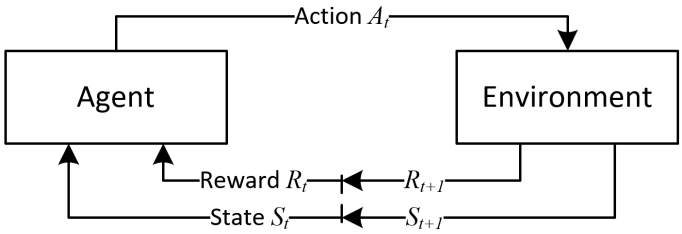


Figure 5.1: Main components of Reinforcement learning.

Note that, in the full reinforcement learning problem, maximizing the reward is not as simple as selecting action A_t that leads to the maximum immediate reward R_{t+1} since this action might lead to a state S_{t+1} where the probabilities of getting future rewards are very small. This is referred to in the literature as maximization of the *delayed* reward.

A reinforcement learning agent is not given any indications about which actions to select other than the reward signal. It has to efficiently perform a trial-and-error process to quickly find the optimal action per state. Undergoing this process, the agent has to balance exploration (trying actions with large uncertainty about their return) and exploitation (selecting actions with known (delayed) rewards depending on the already gained experience). Relying solely on

any of the two will result in a definite failure of the task. This exploration/exploitation trade-off problem is unique to reinforcement learning and cannot be found in any other form of learning [29].

The reward signal is the mean by which the goal of the learning is specified for the agent. It is a designed application-specific function that, given the action of the agent and the state of the system, returns a single real number indicating how good or bad that action was. It corresponds to pleasure and pain in biological systems. In that sense, reinforcement learning is often used as a mathematical model of how animals learn [29].

There exist also reduced versions of the reinforcement learning problem. Among the most important ones are *Multi-armed bandits* and *contextual bandits* [29]. In multi-armed bandits, the agent is always in one state and tries to find the action corresponding to the maximum reward while minimizing the *regret* (the number of times suboptimal actions were chosen). Selecting an action A_t, in this case, only affects the immediately received reward R_{t+1} and does not move the system to another state S_{t+1}.

In contextual bandits, on the other hand, the agent moves from one state to another, yet this move is not controlled by the actions of the agent but by an external input (context). The agent then tries to find the best action per state (context). So, a contextual bandit problem can be considered as a group of several Multi-armed bandits each given a distinctive label called the context.

Due to the interactive nature of reinforcement learning, it is especially suitable for robotic applications, where it comes naturally to a large variety of problems in the field. There is a rich and wide literature for applying reinforcement learning to different aspects in robotics allowing robots to autonomously learn hard-to-engineer behaviors. For a relatively recent review see [15].

For UAVs, reinforcement learning has been employed on different levels. Form learning low-level autonomous control policies for helicopters [20] and multi-rotors [13], to learning higher level tasks like helicopter aerobatic maneuvers [2], multi-rotors target following [17] and navigation through cluttered environments [22].

However, for robotic learning agents, whether they are UAVs, manipulators or any other robotic system, reinforcement learning is more challenging than it is for other general learning agents that do not interact with the real world. Some of the challenges are pointed out to in [15]. They are briefly summarized here:

- Continuous and high-dimensional spaces: the state and action spaces of robotic systems are inherently continuous and of high number of dimensions. This is due to the many degrees of freedom (DoFs) robotic systems often have. This induces what is known as the *curse of dimensionality*, which refers to the problem that, as the number of dimensions grows, exponentially more data and computations are needed to cover the complete state-action space.

- Real-world samples: unlike other reinforcement learning agents (like a one the learns how to play a computer game for instance), robotic agents inherently interact with the physical real world. This makes the trial-and-error learning process costly as robots need power to operate and also suffer from wear and tear. In addition, during learning, robots may produce unstable behaviors while exploring parts of the action space resulting in damages to the robot and the surrounding environment. That is why, for robotics, safe exploration is a key issue in reinforcement learning. A one that is often neglected in the general reinforcement learning community [15].

  Other aspects also of the real world makes robotic reinforcement learning more challenging. Among the most import is the complex dynamics of the robot and the environment.

These dynamics can change due to many external factors like temperature and wear. This is especially true for mult-rotors where aerodynamics is very hard to model and make the repeatability of some actions very difficult.

- Model uncertainty: since it is very difficult to accurately model many robotic systems. It unfeasible in many cases to rely on simulation in the training process.
- Reward Shaping: designing a good reward signal for a robotic reinforcement learning task can be challenging in different ways. This part of reinforcement learning (reward designing or shaping) is considered an art rather than a well-established science [29]. In robotic systems, the reward function often depends on sensors readings which need calibration and sensor fusion techniques. Also, in many cases, there is a trade-off between the complexity of the reward function and that of the learning problem such that, in order to make the robotic learning feasible, a complex sophisticated reward function is required [15].

5.2 Problem Formulation as a Continuum Bandit

As mentioned in section 5.1, in bandit problems, the agent stays in one state throughout its life and the goal is to maximize the received rewards by finding actions that lead to higher values of them. Multi-dimensional continuum bandits are bandits that have a continuous multi-dimensional action space.

Figure 5.2 shows a simple block diagram of the bandit problem. The agent iteratively selects an action and observe the reward resulting from it. The reward signal is application-specific and is designed to reflect the goal of the learning. The agent aims to maximize the reward it gets over time.

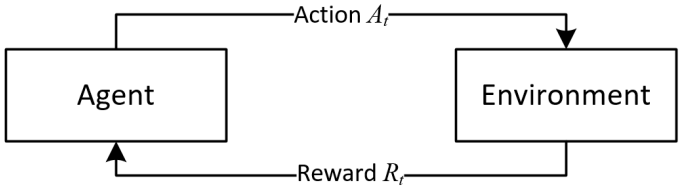


Figure 5.2: Simple block diagram of the bandit problem.

As mentioned in section 1.2, the focus of this thesis is on learning impedance parameters values for tasks requiring a fixed set of them throughout operation. The problem can then be framed as a multi-dimensional continuum bandit with the following elements:

1. **Action Space**
   The action space is equal to the impedance controller's parameters space, i.e. an action is equal to assigning certain values $\mathbf{x}$ to the set of learned parameters. This action space is multi-dimensional and continuous.
2. **Reward Signal**
   The reward corresponds to the performance measure. It is identical to the black-box reward function explained in section 1.2. Given certain impedance parameters values $\mathbf{x}$, it calculates the performance of the impedance controller by observing the drone as it performs a predetermined task.
3. **State**
   The word 'state' can mean different things in different contexts. It can mean for example the configuration of the robot. However, in the bandit problem context, a state is the circumstance that determines the optimal action. Moving to another state means that the action space and/or the optimal action have changed.

Per our specification, as the interest is in a fixed set of parameters values throughout an interaction task, the optimal parameters values depend only on the environment and the task. Consequently, as long as the environment and the task is the same, the agent stays in one state and hence the problem can be considered as a continuum bandit problem. If the state changed (environment or task changed), then a new learning has to be done.

Since the agent stays in one state throughout learning, there is actually no need to formulate, estimate or detect the state, which is one of the main differences between the bandit problem and the full reinforcement learning problem.

Continuum bandit problems have been investigated in literature. However, most of the proposed approaches try to adapt solutions of the traditional discrete multi-armed bandit problem to continuum bandits [16]. This results in discretization of the action space and gives rise to the curse of dimensionality (see section 5.1) as the number of dimensions increases.

5.3 Bayesian Optimization and Reinforcement Learning

Bayesian optimization provides a solution for the continuum bandit problem. Its goal is to search for the action that will produce the largest reward assuming that the state doesn't change. In other words, it tries to find the location of the global maximum of the reward function.

Bayesian optimization, as a solution to the continuum bandit problem described in the previous section, can be then considered a reinforcement learning technique [24]. In fact, Bayesian optimization can be also used to solve the discrete multi-armed bandit problem [24]. In contrast with the classical solutions of the continuum bandit problem, Bayesian optimization doesn't discretize the action space and hence is able to escape the curse of dimensionality.

The differences between Bayesian optimization and other classical reinforcement learning techniques (like Q-learning for instance) come from the fact that Bayesian optimization solves a reduced version of the MDP (bandit problem) where the agent stays in only one state throughout learning. However, Bayesian optimization, as included in the proposed learning framework in section 1.3, has several similarities with classical reinforcement learning methods. They are summarized here:

- **Active Trial-and-Error Learning**
  Just like all reinforcement learning agents, Bayesian optimization in the proposed framework learns by interacting with the environment. It is not given a dataset of situations linked to the correct actions and asked to extrapolate it as the case in supervised learning approaches. Instead, the agent decides what actions to try in an intelligent way.

- **Reward Signal**
  In the proposed learning framework, the goal of the learning is encoded in a reward signal as it is the case in reinforcement learning. This reward is evaluated by direct interaction with the environment. This is why, when designing the reward signal for a task in the proposed learning framework, one can rely on the literature of robotics reinforcement learning.

- **Exploration/Exploitation Trade-Off**
  Bayesian optimization faces the exploration/exploitation dilemma that is unique to reinforcement learning approaches [29]. Further, as exploration in the proposed learning framework is directly connected to robotics dynamic behavior, it imposes all the potential dangers exploration imposes on robotics in other reinforcement learning approaches. That is why all challenges of using reinforcement learning with robotics described in section 5.1 are relevant to the proposed learning framework.

For all of these reasons, before the name 'Bayesian optimization ' becomes popular, some works in literature used the name reinforcement learning to describe it. For instance, a paper that used Bayesian optimization with the UCB acquisition function to learn robot grasping parameters was titled *Active exploration for robot parameter selection in episodic reinforcement learning.* [16].

Conclusion and Future Work

The presented research work started with the aim to:

> "Design and apply a suitable model-free approach to allow a drone, equipped with an impedance controller, to autonomously learn suitable impedance parameters for a certain interaction task."

Concentrating on interaction tasks requiring a single set of impedance parameters values throughout operation allowed formulating the problem as an optimization of an expensive black-box function. As opposed to supervised approaches, this formulation does not require accurate modeling of the environment or building large datasets. As opposed to formulations meant for learning trajectories of impedance parameters values, this formulation allows easier generalization of knowledge to similar interaction tasks.

Three features were prioritized when looking for a solution: **sample-efficiency**, **scalability to higher dimensions** and **safety**. A learning framework was proposed to solve the problem. The framework relies on Bayesian optimization and episodic reward calculation requiring the drone to repeatedly perform a predetermined task in the environment actively searching in the impedance parameters space for the suitable values.

Proposing this framework, two research questions (section 1.4) arose. The first one was:

> "How to adapt Bayesian optimization to the requirements and limitations faced in learning impedance parameters values for a drone."

Investigating the different elements of Bayesian optimization (chapter 2), choices were made with the three priorities mentioned above in mind. Most significantly, for a sample-efficient learning, a Gaussian process was used as a surrogate model combined with the RBF kernel. Although the made choices impose several assumptions on the black-box functions to be encountered during learning, it was shown that these assumptions are justifiable.

One of the Bayesian optimization choices that required a deeper study was the selection of a suitable acquisition function and its hyperparameter value. For that, simulations were done on randomly generated functions of different numbers of dimensions (chapter 3). Out of the three considered acquisition functions it was found that EI is not suitable for sample-critical problems, PI is the most sample-efficient one and UCB is less likely to fall into areas of low rewards and hence provides safer learning.

A novel method for optimizing acquisition functions was proposed to make the learning safer, computationally cheaper and abler to scale to higher dimensions. The method relies on optimizing the local areas around the already seen observations instead of global optimization. This allows the agent to get early warnings along paths leading to low rewards. The simulations done on the randomly generated functions as well as those done on the simulated drone confirm the significance of this modification in achieving safer learning at the cost of being slightly slower in reaching areas of high rewards.

The other research question tackled in this work was:

> "How to apply the proposed learning framework to find impedance parameters values for an interaction task and verify its validity in a simulated environment?"

Attention was given to sliding tasks. It was seen that, even for this very simple task, it is difficult for an expert human to tune the impedance parameters as the suitable values depend on many unknown factors.

A reward function was designed to reflect the goals of learning in the sliding task. The function tries to minimize the tracking errors and the used impedance parameters values.

With an unpredictable range that changes from one situation (environment and task) to another, it is difficult to unify the used learning settings for the used reward function over different learning problems. To remedy this, a novel method was proposed to *condition* the reward function, exploiting the provided initial observations to seek improvement over them.

Great emphasis was put on providing meaningful initial knowledge to the learning. It was seen from the simulation done in the randomly generated functions (chapter 3) that as the number of dimensions of the search problem increases, it is inevitable to provide meaningful initial knowledge to achieve satisfactory results. Providing knowledge depending on random sampling or more systematic sampling approaches (like Latin hyper-cubes) proved useless. Instead, the proposed framework provides two main means for embedding human intuition in the learning:

- Selecting the relevant impedance parameters to learn depending on the task.
- Selecting the locations of the initial observations.

The simulations shown in chapter 4 presented a practical example of using these two forms of initial knowledge in sliding tasks.

The simulated sliding tasks were varied by changing the friction and the trajectory of the sliding. The results showed that the proposed framework is able to automatically find suitable impedance parameters values in different situations given the same initial knowledge. The results also confirmed that the proposed learning method makes it possible to generalize the learned impedance parameters values to similar tasks. Further, it was shown that the two proposed modifications to Bayesian optimization (local optimization of acquisition functions and conditioning the reward signal with a logistic function) successfully allowed maintaining a consistent learning performance across different learning tasks and increased the sample-efficiency and safety. In most of the performed simulations, most of the improvement in the received rewards happened only within 30 training episodes (around 20 minutes of operation time).

The results of this research work, along with the proposed modifications to Bayesian optimization that make the technique safer and more sample-efficient for robotic systems are being prepared to be published as a journal paper.

Future Work

The presented research can be extended in different directions as detailed here.

Experimental Validation

The proposed learning framework provided satisfactory simulation results as shown in chapter 4. It is very important to validate the conclusions reached by the simulation results through practical experiments in the real world where additional challenges are present. This includes uncertainty sources like noise in the received sensory data as well as aerodynamics.

Another significant challenge worthy of highlighting is the training time. Considering that the simulation results showed that 30 training episodes were enough in most cases to achieve satisfactory results, it is important to investigate if this number of episodes will be enough in a practical situation.

Automatic Failure Detection

The current work provided failure handling procedures. However, failure detection was left to the user. Adding reliable automatic failure detection techniques to the learning will significantly reduce its cost and increase its safety as mentioned in section 4.3.1.

More Advanced Reward Functions

Although the used reward function described in section 4.2 is able to achieve good learning results, it only considers the translational and rotational errors and the average value of the controlled impedance parameters. Different other factors can be added to the reward function (e.g. force measurement). The effects and value of adding such additional factors should be investigated.

More Complex Interaction Tasks

The simulations done in this research concentrated on sliding tasks. It is beneficial to investigate if the same framework can be used with other interaction tasks (e.g. peg-in-hole).

Deeper Look into Local Optimization

The empirical experiments, whether on the randomly generated functions in chapter 3 or on the simulated drone in chapter 4, show that local optimization has a great potential to make learning sample-efficient, safer and able to scale to higher dimensions. However, a rigorous formal investigation has to be done to find out what are the conditions for the success of this technique and whether it is possible to offer theoretical convergence bounds on the convergence rates when using it.

Moving to Contextual Bandits

The most significant limitation of the proposed learning framework is that it is only suitable for interaction tasks requiring a fixed set of impedance parameters values throughout operation. For more complex interaction tasks requiring variable impedance parameters values, the used formulation is not suitable. Because of the linking of proposed framework to continuum bandits as seen in chapter 5, the idea can be extended to contextual bandits to be valid for tasks requiring changing the impedance parameters values during operation. A simple example would be a sliding task where the surface has different materials on different areas.

A Impedance Control

A brief overview of the used interaction control is included here. A through discussion can be found in [31]. As mentioned before, active interaction control schemes can be divided to two main categories: direct and indirect force control. In direct force control, the contact force and moment are explicitly controlled with a force feedback loop. This requires reasonably precise 6D force measurements. Instead, in this thesis, we work with indirect force control. One of the well known and widely used schemes in this category is impedance control, where the deviation of the end-effector motion from the desired motion due to the interaction with the environment is related to the contact force through a mechanical impedance/admittance.

In many cases, the words impedance and admittance are used interchangeably. However, a distinction should be made between impedance and admittance control schemes. The control behaves as an impedance if the robot reacts to the motion deviation by generating forces, while it corresponds to an admittance if the robot reacts to interaction forces by imposing a deviation from the desired motion [31].

As an initial phase in this research, both admittance and impedance control schemes were implemented. Admittance control has the nice feature of decoupling between position control and interaction control. To be able to do this, it needs force measurements at the end-effector. It turned out that Gazebo simulator is not able to produce a reliable and consistent force measurement (see appendix 1). In addition to that, in a practical situation, force measurements require either complex processing to get force estimation (beyond the scope of the thesis) or a costly sensor. So, it was decided to apply the learning to impedance control.

An impedance controller is simply a PD controller where the wrench applied by the drone in the end-effector frame $\mathbf{W}^E$ is:

$$\mathbf{W}^E = \mathbf{W}_p + \mathbf{W}_d + \mathbf{W}_{ff} \tag{A.1}$$

where $\mathbf{W}_p$ is the spring wrench (proportional term), $\mathbf{W}_d$ is the damper wrench (derivative term) and $\mathbf{W}_{ff}$ is the feedforward wrench that compensates gravity all in the end-effector frame.

The spring wrench $\mathbf{W}_p$ has a rather lengthy expression, but is parameterized by the positive semi-definite 6×6 matrix $\mathbf{K}_p$:

$$\mathbf{K}_p = \begin{pmatrix} \mathbf{K}_r & \mathbf{K}_c \\ \mathbf{K}_c^\top & \mathbf{K}_t \end{pmatrix} \tag{A.2}$$

where $\mathbf{K}_r$ and $\mathbf{K}_t$ are the symmetric 3×3 translational and rotational spring gains matrices respectively and $\mathbf{K}_c$ is the 3×3 coupling spring gains matrix.

The damper wrench has a simple expression:

$$\mathbf{W}_d = -\mathbf{K}_d(\boldsymbol{\tau}_E^{E,I} - \boldsymbol{\tau}_C^{E,I}) \tag{A.3}$$

where $\boldsymbol{\tau}_C^{E,I}$ is the twist between the end-effector frame E and the inertial frame I expressed in the end-effector frame, $\boldsymbol{\tau}_C^{E,I}$ is similarly the twist between the command (set) point frame C and the inertial frame I expressed in the end-effector frame, and $\mathbf{K}_d$ is a positive diagonal matrix representing the damping (derivative) gains. The first three elements control the rotational damping and the last three ones control the translational damping.

Tuning the parameters (or the gains) $\mathbf{W}_p$ and $\mathbf{K}_d$, one can control the compliance and stiffness of the drone when it interacts with the environment. The complete number of parameters that need to be tuned, and consequently the maximum number of dimensions of the learning problem, is then 27.[1]

[1] 6 from $\mathbf{K}_d$, 6 from each of $\mathbf{K}_t$ and $\mathbf{K}_r$ because they are symmetric, and 9 from $\mathbf{K}_c$.

B Simulation Software Implementation Details

Simulation Setup

Impedance control was implemented for the betaX hexarotor drone. In addition to that, a GUI was developed to allow users to easily manage the learning process of a set of impedance parameters. The software was built over `ROS` (robotic operating system) in a way that makes it possible to be used either with a simulated version of the drone using `Gazebo` and `RotorS` simulators [11] or with the real physical drone. A simplified UML-like block diagram of the main components of the software is shown in Fig. B.1.

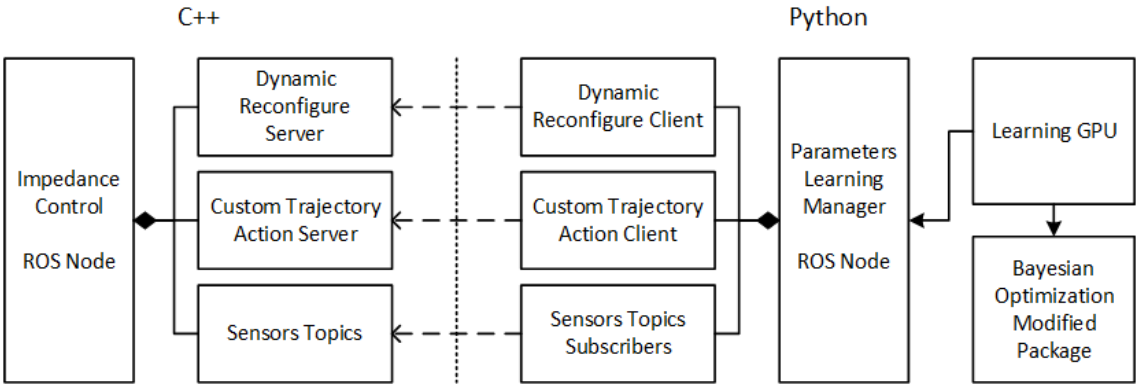


Figure B.1: A simplified UML-like block diagram of the simulation software.

The impedance controller side (left) is written in C++ and can be used with a real drone or with a simulated version of it. The controller has a `ROS` *dynamic reconfigure server*. This simply allows changing the impedance parameters in run-time. This server can be also contacted by a *client* and inquired about the current values of the parameters or asked to change (some of) them through the `ROS dynamic reconfigure API`. In the current implementation, the impedance parameters that can be controlled via the dynamic reconfigure server are the diagonal elements of both $\mathbf{K}_t$ and $\mathbf{K}_r$ as well as the elements of $\mathbf{K}_d$. All other parameters, including the spring coupling gains matrix $\mathbf{K}_c$ are fixed at zero.

Moreover, the controller has a custom trajectory `ROS` *Action Server*. An application-specific custom trajectory has to be designed to assess the performance of the impedance controller given a set of parameters values as mentioned in section 1.3. An *action client* can contact the server and request the controller to follow this custom trajectory. To ensure the independence of each training episode (or iteration), the action server is designed to follow the procedures described in section 4.3.1.

Finally, the controller side has also a number of sensors topics where sensor data is published. This includes the current position of the drone as well as the rotors speed and other measurements depending on the sensor put in (the simulated version of) the drone.

The learning management side (right) is completely written in Python to be able to use of the `BayesianOptimization` package written in python. It has three main components. The first one is the Parameter Learning Manager which is implemented as a `ROS` node and is responsible for direct communication with the controller node as well as sensor data storage and reward calculation. It has a `ROS` dynamic reconfigure client that allows it to directly access the current values of the impedance parameters and change them while the drone is flying. In addition to that, the learning manager has a `ROS` action client for the custom trajectory action server mentioned above. Via this client, it is possible to request the start of the custom trajec-

tory, receive messages about the initialization and progress and also stop the custom trajectory if something wrong happened. The manager node also subscribes to the sensor topics to store sensor data during a training episode and use it to calculate the reward.

Because the parameter learning manager has direct access to the sensor data, can control the impedance parameters, and start and stop the custom trajectory, it is possible to equip it with a failure detection capability as described in section 4.3.1. As failure detection techniques are beyond the scope of this thesis, it is currently left to the human user who can press 'FAILURE' button in the learning GUI (described in section B) to trigger the same failure handling procedures mentioned in section 4.3.1.

The other two components of the learning management side is the Learning GUI and the `BayesianOptimization` package equipped with the local optimization of acquisition functions as discussed in section 3.4. The learning GUI is described in more details in section B. Its main purpose is to allow the user to select the parameters to be learned and to monitor the learning episodes. It also abstracts the whole process of performing a custom trajectory and calculating the reward based on the sensory data into a simple black-box function that takes the parameters values and returns the reward. This abstracted black-box function is the one passed to the `BayesianOptimization` package, which is also managed by the GUI. The user can select the acquisition function to use and set its hyperparameters.

To successfully carry impedance parameters learning for a certain task, the user has several responsibilities. One of the design issues of the software architecture shown in Fig. B.1 is that one of the learning responsibilities mentioned in that section (designing the custom trajectory) is done in the controller side, while all others are done in the learning management side. However, for better modularity, all learning-related software should be kept in the same place. The current implementation has this issue because it a lot easier to code the custom trajectory directly in the controller, otherwise the trajectory has to published from the learning manger node in a `ROS` topic and then received by the controller when the action server is called. Given the tight time frame, these complexities are left for future work.

Simulation Environment

The simulations in this report are all done using `Gazebo` and `RotorS`. They are concentrated on the task of writing or drawing a certain shape on a wall. For that purpose, a `Gazebo` *world* was created as was seen in Fig. 4.1.

The world is very simple containing only a 4×4 m^2 wall that is positioned one meter away from the origin along the x-axis. There were multiple trials to add a virtual force sensor either to the wall or to the end-effector of betaX and then use it for force feedback in admittance control or as a part of the reward calculations. However, it was found that the two available `Gazebo` force plugins produce inconsistent and unrepeatable measurements. It was not possible given the scope and time limitations to design a custom force plugin.

An end-effector with zero DoFs was attached to the drone through a fixed joint. The tip of the end-effector is ball shaped to simulate the tip of a maker pen. Through setting the friction parameters of the wall and the tip of the end-effector, one can simulate drawing with different tools on different kinds of surfaces and hence different learning situations as described in section 4.1.3.

Learning GUI

The learning GUI (shown in Fig. B.2) was designed to allow the user to control the different stages of Bayesian optimization. It is based on the `Tkinter` library, the standard Python choice for building GUIs. The different parts of the GUI are walked through here via a typical use case.

The user first runs the (simulated) drone impedance controller ROS node and then opens the GUI as in Fig B.2. The GUI runs an instance of the parameters learning manager (described above), which automatically connects to the dynamic reconfigure server, the action server and the sensors topics. The user can then select the impedance parameters (obtained from the dynamic reconfigure server) to learn in the 'Dynamic Reconfigure Settings' section in the GUI.

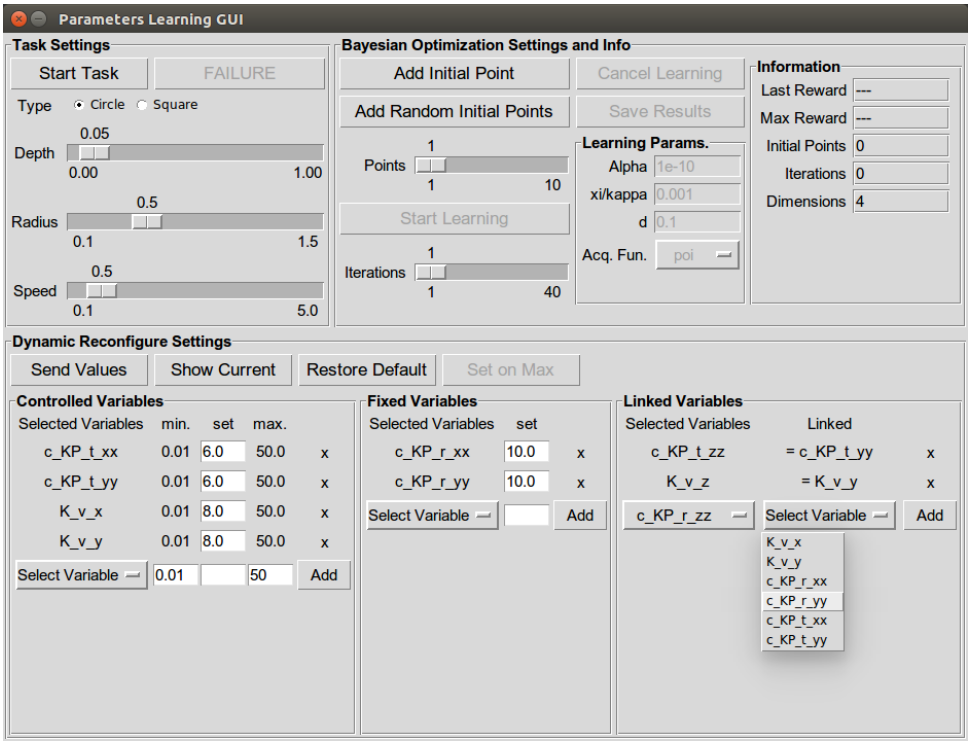


Figure B.2: Selecting impedance parameters to learn in the GUI.

Parameters can be added to the three different categories mentioned in section 4.3.2. They can be *controlled parameters*, in which case the user has to provide the lower and upper bounds (minimum and maximum) of the selected parameters. The user can also optionally specify the set value of the added parameters (otherwise the default value automatically appears after adding the parameter). Controlled parameters are the parameters Bayesian optimization controls as inputs to the black-box reward function during learning. By providing the bounds of these parameters, the user effectively specifies the domain of the black-box reward function.

The user can also add *fixed parameters*, these are the parameters, the user wishes to keep with specific values throughout the learning. Note that this is only useful if the user wants to change the default values of these parameters. All other unselected parameters are kept fixed on their default values throughout learning. Finally, the user can also *link* some parameters to selected controlled or fixed parameters. The linked parameter will always have the same value as the one it is linked to.

As long as the learning process has not started (no initial points were added) The user can freely add and remove parameters and change their values by writing in the white entry boxes and pressing 'Send Values'. The user can also verify the current values of the parameters (by getting their current values from the dynamic reconfigure server) by pressing 'Show Current' and can restore the default values of all parameters by pressing 'Restore Default'. The 'Set on Max' button is only activated after learning has run. It sets the parameters to the values that produced the maximum seen reward.

In the 'Task Settings' section of the GUI, the user can control different aspects related to the custom trajectory. The 'Type' setting allow the user to select between drawing a circle or a square.

The 'Depth' setting determines how *deep* (in meters) should the set point be placed behind the wall (corresponds to Δ in section 4.1.1). The 'Radius' setting determines the size of the drawing (in meters) and corresponds to r in the circle and square trajectories in equations (4.2) and (4.3). Finally the 'Speed' setting control the lateral speed of the drawing (in meters per second) and corresponds to v in (4.2) and (4.3). These settings can be freely changed by the user as long as the learning hasn't started. After the learning has started (by adding initial points) their control is deactivated and the last settings are fixed.

The user can then press 'Start Task'. This will make the drone perform one episode. When an episode runs, either initiated by the user by pressing this button or as an iteration in learning (discussed below), the same custom trajectory initialization procedures detailed in section 4.3.1 are performed. The manager then starts storing all sensory data coming from the drone as the impedance controller proceeds in following the custom trajectory. While initializing or following the custom trajectory, all controls in the GUI are deactivated and the only thing the user can control is the 'FAILURE' button (as in Fig. B.3) which should be pressed when the user notices that the drone is unstable or having a dangerous performance.

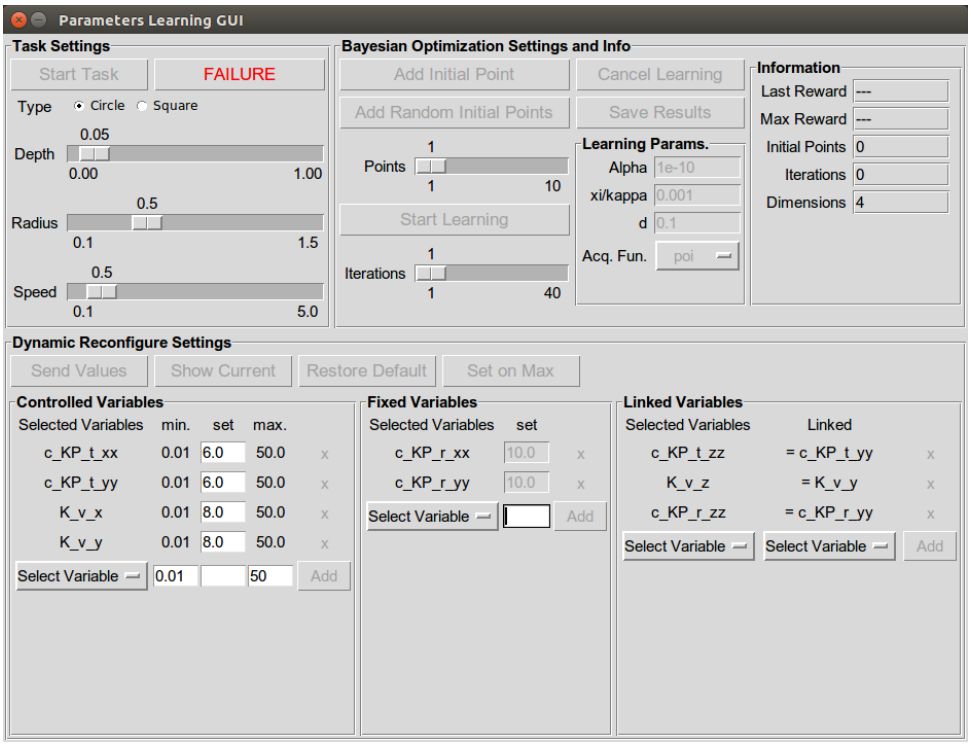


Figure B.3: The GUI while the drone is performing the custom trajectory.

After finishing the "task" (episode), the calculated reward (by the parameters learning manager) is shown to the user in the 'Bayesian Optimization Settings and Info' section of the GUI. The user can then add the last run as an initial point for learning by pressing 'Add Initial Point'. The user can also add a number of random initial points by specifying the number of points and pressing 'Add Random Initial Points'. In this case, a number of consecutive episodes will be run, during which the user sees the selected random values for the controlled parameters in their respective white entry boxes and the received rewards in the information section.

Figure B.4 shows the GUI after adding 10 random initial points. After adding the points, the learning is considered to have started. The learning settings are activated and it is not possible to add or remove parameters or change the task settings anymore except if the user pressed 'Cancel Learning', in which case all the added points, whether initial points or training episodes, are removed and the user gets back control over the task and parameters settings.

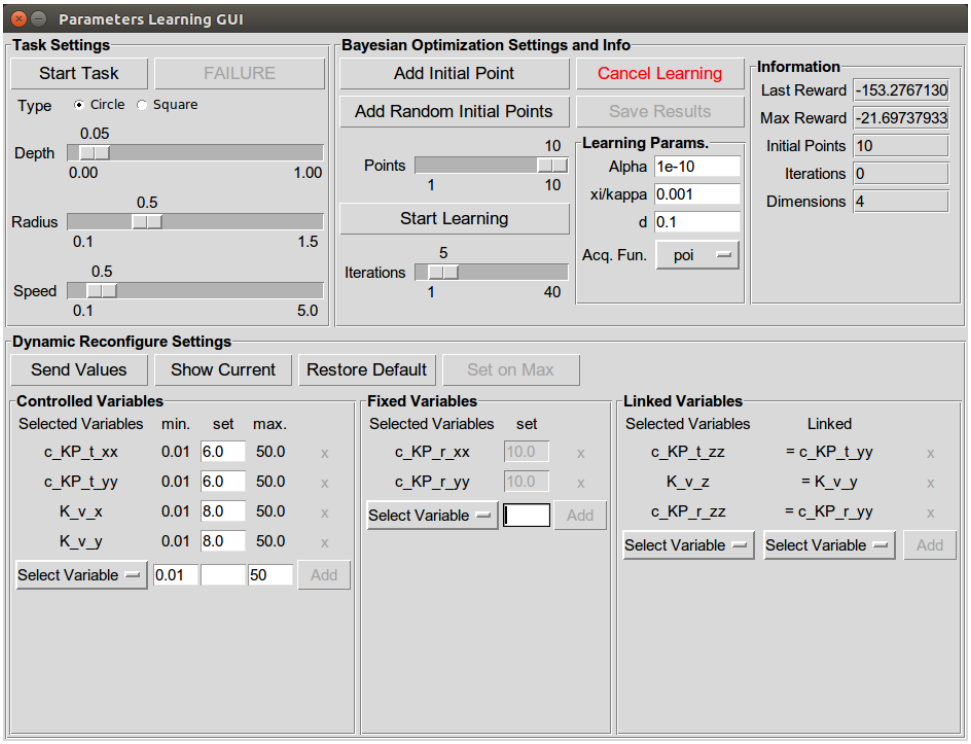


Figure B.4: The GUI after adding initial points.

The user can select the number of learning iterations to do as well as the acquisition function and other learning hyperparamters. 'alpha' is the noise hyperparameter corresponding to σ^2 in equations (2.4), (2.6) and (2.7). It has to be set to a proper value depending on the anticipated noise in the observations of the black-box reward function as mentioned in section 3.2.2, 'xi/kappa' is the hyperparameter of the selected acquisition function, and 'd' is the distance hyperparameter of the local optimization of the acquisition functions as presented in section 3.4.2. After pressing 'Start Learning' the selected number of iterations will be done, during which the information regarding the received rewards and the selected controlled parameters values are shown in GUI in real-time.

After finishing the selected number of iterations (Fig. B.5), the user can use the button 'Set on Max' to set the parameters on the values that produced the maximum seen reward so far. The user can also resume learning for an additional number of iterations, optionally changing the settings of the learning which allows creating acquisition functions *profiles* as discussed in section 2.4. In addition, the user can also 'Save Results'. This will write all the learning data, including settings to a file using Python's `pickle` library. This makes it easy to analyze the results later.

Various design and programming challenges were faced while creating this GUI. Due to the limited space of this report, they cannot be fully described here. One of them, for example, was the need to use threading to allow information to be shown in real-time and to give the user the possibility to use the 'Failure' button while a training episode is being done. The different design choices are explained in the documented code.

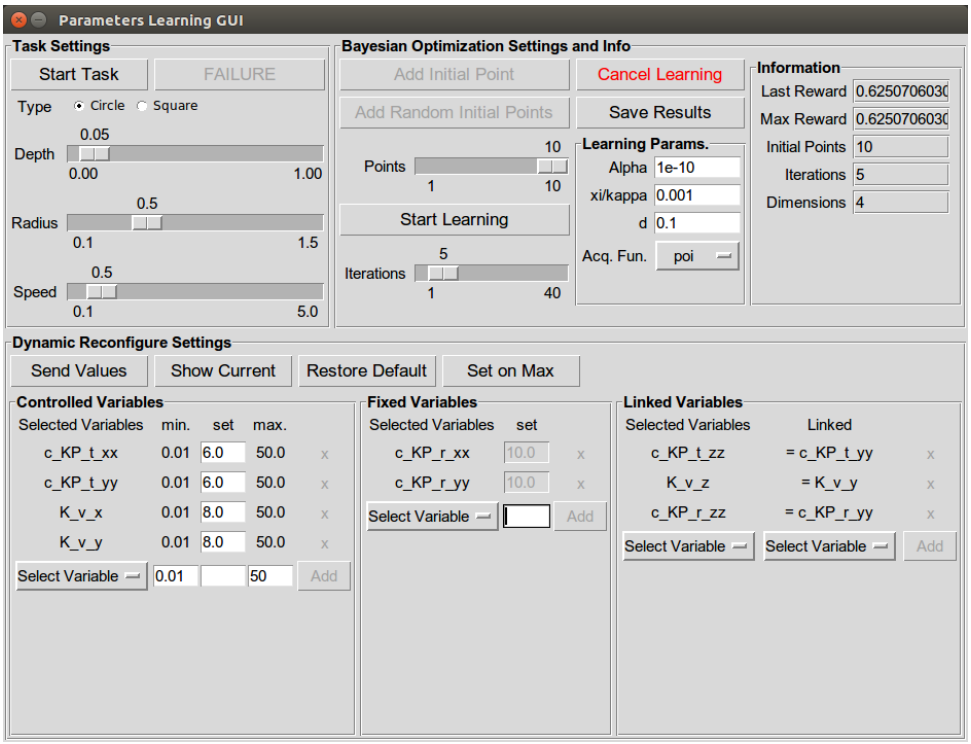


Figure B.5: The GUI after doing multiple Bayesian optimization iterations.

Bibliography

[1] Reference tables - coefficient of friction. http://www.engineershandbook.com/Tables/frictioncoefficients.htm, 2006. [Online; accessed 6-July-2018].

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[3] Navid Aghasadeghi, Huihua Zhao, Levi J Hargrove, Aaron D Ames, Eric J Perreault, and Timothy Bretl. Learning impedance controller parameters for lower-limb prostheses. In *IEEE/RSJ international conference on Intelligent robots and systems (IROS)*, pages 4268–4274. IEEE, 2013.

[4] The GPyOpt authors. Gpyopt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016. [Online; accessed 6-June-2018].

[5] E. Brochu, V. M. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *ArXiv e-prints*, December 2010.

[6] Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Learning variable impedance control. *The International Journal of Robotics Research (ijrr)*, 30(7):820–833, 2011.

[7] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

[8] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503, 2015.

[9] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[10] Fernando Nogueira. Bayesianoptimization, a python implementation of global optimization with gaussian processes. https://github.com/fmfn/BayesianOptimization, 2018. [Online; accessed 6-June-2018].

[11] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors - a modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[12] Matthew D Hoffman, Eric Brochu, and Nando de Freitas. Portfolio allocation for bayesian optimization. In *UAI*, pages 327–336. Citeseer, 2011.

[13] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[14] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

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

[16] Oliver Kroemer and Jan Peters. Active exploration for robot parameter selection in episodic reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pages 25–31. IEEE, 2011.

[17] Siyi Li, Tianbo Liu, Chi Zhang, Dit-Yan Yeung, and Shaojie Shen. Learning unmanned aerial vehicle control for autonomous target following. *arXiv preprint arXiv:1709.08233*, 2017.

[18] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, volume 3, pages 321–328, 2007.

[19] Djordje Mitrovic, Stefan Klanke, Matthew Howard, and Sethu Vijayakumar. Exploiting sensorimotor stochasticity for learning control of variable impedance actuators. In *Humanoids*, pages 536–541. Citeseer, 2010.

[20] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[21] José Nogueira, Ruben Martinez-Cantin, Alexandre Bernardino, and Lorenzo Jamone. Unscented bayesian optimization for safe robot grasping. In *IEEE/RSJ international conference on Intelligent robots and systems (IROS)*, pages 1967–1972. IEEE, 2016.

[22] Stephane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[23] Markus Ryll, Giuseppe Muscio, Francesco Pierri, Elisabetta Cataldi, Gianluca Antonelli, Fabrizio Caccavale, and Antonio Franchi. 6d physical interaction with a fully actuated aerial robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5190–5195. IEEE, 2017.

[24] 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*, 104(1):148–175, Jan 2016.

[25] The sklearn authors. Gaussian processes. http://scikit-learn.org/stable/modules/gaussian_process.html, 2017. [Online; accessed 6-June-2018].

[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2951–2959, 2012.

[27] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*, pages 2171–2180, 2015.

[28] Freek Stulp, Jonas Buchli, Alice Ellmer, Michael Mistry, Evangelos A Theodorou, and Stefan Schaal. Model-free reinforcement learning of impedance control in stochastic environments. *IEEE Transactions on Autonomous Mental Development*, 4(4):330–341, 2012.

[29] Richard Sutton and Andrew Barto. *Reinforcement learning: An introduction (Complete Draft of the Second Edition)*. MIT press, 2017.

[30] Boxin Tang. Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

[31] Luigi Villani and Joris De Schutter. Force control. In *Springer Handbook of Robotics*, pages 161–185. Springer, second edition, 2016.