Universidad Zaragoza
1542

Master's Thesis

Multisolution Bayesian optimization for robotic manipulation tasks

Autor

IGNACIO HERRERA SEARA

Supervisor

RUBÉN MARTÍNEZ CANTÍN

Escuela de Ingeniería y Arquitectura

2022

Multisolution Bayesian optimization for robotic manipulation tasks

ABSTRACT

Humans have acquired over thousands of years an incredible ability to grasp and manipulate objects, both known and unknown, of different shapes, sizes and materials. This capability to interact with objects is of great interest in robotics. The use of robotic arms with hand-type grippers allows the automation of tasks previously performed by humans. This gives them greater flexibility when manipulating and grasping objects compared to traditional grippers used in the industry. However, replicating this dexterity is not a simple task and, despite recent advances, it is still far from matching human level.

One of the main challenges facing this field of research is the grasping of previously unknown objects, which in practice leads to a high overhead in costs due to the required reconfiguration and reprogramming the optimal grasp for each new object. Determining the optimal grasp through a brute force approach is unfeasible due to the cost of operating the real robot. Furthermore, despite the existence of simulators that allow to efficiently evaluate and verify grasps, automating the identification of the optimum remains a complex task.

On the other hand, depending on the task to be performed or the existence of occlusions in the environment, it is possible that the optimal grasp may prevent certain manipulations with the object. Therefore, it is necessary to find multiple alternative grasps, sufficiently good and diverse, that allow the correct manipulation of the object.

In this work we focus on the problem of automatically obtaining multiple grasps for previously unknown objects with robotic hands. To obtain the grasps, only tactile exploration of the object was used, taking advantage of the sensors in the robotic hands. For this purpose, in this work we have designed and implemented a 3D grasp evaluation environment in simulation with Simox simulator. The iCub robot, designed for research tasks, has been used to carry out experiments.

Finally, obtaining multiple grasps has been approached as an optimization problem, where the function to be optimized is a quality metric of a given relative hand-object position. To address this problem we have used Bayesian optimization, an efficient global optimization method that does not require prior information about the problem and works sequentially with a trial-and-error approach to identify the optimum. However, this algorithm is focused on finding a single optimal solution, so it has been necessary to design and implement a variant that identifies multiple solutions (optimal or suboptimal), known as multisolution Bayesian optimization.

Optimización bayesiana multisolución paara tareas de manipulación robóticas

RESUMEN

Los humanos han desarrollado durante miles de años una capacidad increíble para agarrar y manipular objetos, tanto conocidos como desconocidos, de diferentes formas, tamaños y materiales. Esta habilidad para interactuar con los objetos resulta de gran interés en robótica. El uso de brazos robóticos con herramientas de tipo mano permiten la automatización de tareas previamente realizadas por humanos. Gracias a esto les permite tener una mayor flexibilidad a la hora de manipular y agarrar objetos frente a las pinzas tradicionales utilizadas en la industria. Sin embargo, replicar esta destreza no es una tarea sencilla y, a pesar de los últimos avances, aún dista mucho de igualar a la humana.

Uno de los desafíos a los que se enfrenta esta área de investigación es el agarre de objetos previamente desconocidos, el cual en la práctica conlleva altos sobrecostes al necesitar reprogramar y reconfigurar el agarre óptimo para cada nuevo objeto. Obtener el agarre óptimo a través de métodos de fuerza bruta resulta inviable debido al coste que supone operar el robot. Además, a pesar de la existencia de simuladores que permiten de manera eficiente evaluar y verificar agarres, automatizar la identificación del óptimo sigue siendo una tarea compleja.

Por otro lado, dependiendo de la tarea a realizar o de la existencia de oclusiones en el entorno, es posible que el agarre óptimo impida llevar a cabo ciertas manipulaciones con el objeto. Por tanto, es necesario encontrar múltiples agarres alternativos lo suficientemente buenos y diversos que nos permitan la correcta manipulación del objeto.

En este trabajo nos centramos en el problema de la obtención automática de múltiples agarres para objetos previamente desconocidos con manos robóticas. Para la obtención de los agarres se ha utilizado únicamente la exploración táctil del objeto aprovechando los sensores con los que cuentan las manos robóticas. Para ello, en este trabajo se ha diseñado e implementado un entorno de evaluación de agarres 3D en simulación haciendo uso del simulador Simox. Para la realización de experimentos se ha utilizado el robot iCub, pensado para tareas de investigación.

Finalmente, la obtención de múltiples agarres se enfoca como un problema de optimización, en el que la función a optimizar es una métrica de calidad de una determinada posición relativa mano-objeto. Para abordar este problema se ha utilizado la optimización bayesiana, un método eficiente que no requiere de información sobre el problema y que funciona de manera secuencial mediante una aproximación de prueba y error para encontrar el óptimo. Sin embargo, este

algoritmo se enfoca en la obtención de una única solución óptima, por lo que ha sido necesario diseñar e implementar una variante del mismo que identifique múltiples soluciones, tanto óptimas como subóptimas, conocida como optimización bayesiana multisolución.

Index

Chapter 1

Introduction

1.1 Introduction

The ability of the human hand to grasp and manipulate (known and unknown) objects of different sizes, shapes, and materials is unbeatable. This dexterity to interact with objects has been a research field extensively studied in robotics for years. The use of robotic arms with hand-type gripper allows the automation of tasks previously performed by humans, minimizing the need to condition the environment for the robotic arm. Its application is obvious: from enabling a higher degree of automation in industrial processes to increasing the motor skills of people with disabilities through assistive systems. Figure 1.1 shows some popular robotic hands used in research.

Figure 1.1: Examples of robotic hands: Shadow hand (left), iCub hand (middle) and Barrett hand (right).

However, replicating this skill is a very complicated task and, although advances have been made in recent years, the dexterity of robotic hands is still far from human. One of the main challenges facing robotic manipulation is the grasping of previously unknown objects, which in practice leads to a high overhead in costs due to the required reconfiguration and reprogramming of the robot's grasps for each new object. A large part of this challenge is based on correctly identifying the optimal grasp (position in which to orient the end-effector and its actu-

ators) that allows for subsequent manipulation. In addition, thanks to the use of robotic hands, we can also use their posture and the tactile information they provide to get a better grasp, which is an advantage over traditional grippers. Determining the optimal grasp through a brute force approach is not viable due to the cost of operating the real robot. Moreover, despite the existence of simulators that allow to efficiently evaluate and verify that the configuration is valid for a given grasp, automating the identification of the optimal grasp remains a complex task. Therefore, it is necessary to use methods that try to identify the grasp efficiently. One of the most efficient methods for finding an optimal solution is Bayesian optimization (BO).

BO is a global optimization method that does not require information about the problem to be optimized, it works sequentially with a trial and error approach where it evaluates different parameters, in this case grasp configurations, to identify the optimal one. The fact that it does not need prior information about the problem makes it easy to learn the grasp of any object without the need to make changes to the system. There are few previous works that are based on BO for grasp calculation. One of the first works used BO in combination with mean-shift to efficiently explore an object for grasping, using an external camera to evaluate the quality of the grasp [1]. While most algorithms based on BO use a Gaussian process as a stochastic model, the work of Montesano and Lopes uses beta processes to represent a grasp as a binary value [2]. Another advantage of BO is the ability to use the stochastic model to make other types of reasoning about optimization, as in the case of Nogueira et al.[3] where they analyze the grasp reproducibility to obtain safe and reliable solutions. This work was later extended to incorporate collisions and occlusions [4]. There are also several works that address both the task of tactile exploration with BO and object perception with point clouds [5]. Finally, Englert and Toussaint [6] use BO with reinforcement learning for manipulation tasks.

However, there are situations in which the optimal solution may not be feasible a posteriori or it is necessary to have multiple solutions that meet unique business and research objectives. In the problem of robot grasping, it is necessary to emphasize that during the grasp learning process, it is not feasible to model all possible manipulation scenarios. In practice, it may happen that an optimal grasp makes it impossible to perform certain tasks and manipulations. For example, to manipulate an object and place it on its base, a grasp that covers the base may fail in the task. That is why it is of interest that the system be able to offer multiple grasp alternatives. This problem of finding multiple grasp solutions is addressed from a computer vision perspective in the work of Ghazaei et al. [7]. As a preliminary work to this thesis, we address this problem using some existing multisolution Bayesian optimization (MBO) methods [8]. Although it must be emphasized that the state-of-the-art in MBO is practically non-existent [9, 10, 11].

Consequently, given the lack of research on MBO and the good reception of our preliminary work at the conference it was presented, this master's thesis proposes several methods based on Bayesian optimization to address the problem of finding multiple grasp solutions. Therefore, this thesis has a strong research component that is intended to serve as a basis for future work. In addition, the methods to be developed can also be used in other problems, such as the tuning of hyperparameters of a neural network or the development of new drugs.

1.2 Objectives and scope

The aim of this work is to **design and develop some Bayesian optimization based methods to obtain multiple solutions** to a given problem. In addition, the implemented methods will be used to obtain **multiple grasp configurations on previously unknown objects with robotic hands in simulation.** Among the main problems to be addressed to achieve these objectives are: the modification of the classical BO algorithm, which is sequential and focused on finding a single solution; the implementation of the new algorithm, for which it will be necessary to study and use a BO library; the study and programming of grasping tasks with robots and, finally, the design and execution of different experiments and their subsequent analysis.

Therefore, in order to overcome the main challenges and achieve the proposed objectives, the following tasks have been identified and will be carried out in the order indicated:

1. First, a study of state-of-the-art methods on Bayesian optimization and multisolution optimization (both BO-based and non-BO methods).

2. Design of a multisolution Bayesian optimization system.

3. Implementation of the proposed method in the Bayesian optimization library BayesOpt [12], which will require a study and understanding of how it works internally.

4. Analysis and comparison of the performance of the proposed methods with others already existing in synthetic multimodal functions. This step will allow us to check the correct performance of the methods and how well they behave, in order to determine if any modification is needed before moving to the next step, since the execution of experiments with robots is more expensive.

5. Implementation of a grasp simulation environment (GSE) in the Simox robotic simulator [13].

6. Design and execution of experiments for the humanoid robot iCub, which has a robotic hand as end-effector; with objects of different size and shape. Finally, an analysis and

comparison of the diversity and quality of the grasps returned as a solution with the proposed methods versus other existing ones.

This work has been carried out in the robotics, perception, and real-time (RoPeRT) research group as part of an internship at the *Instituto de Investigación en Ingeniería de Aragón* (I3A). In addition, preliminary results were published at *XLIII Jornadas de Automática* [8].

1.3 Methodology and tools

As mentioned in the previous section, the proposed methods will be implemented in the free and open source BayesOpt library. This library is an efficient implementation of the Bayesian optimization methodology for nonlinear optimization, experimental design and hyperparameter tunning. The library has been developed in standard C++, which allows it to be extremely efficient as well as portable and flexible. It also includes an interface to other languages (Python, Matlab and Octave). The library offers several surrogate models with different distributions (Gaussian processes, Student-t processes,...), as well as various kernels and mean functions. It also implements a wide range of acquisition functions (such as EI, UCB, PI,...), allowing, in addition, to establish combinations of them. Currently, it supports the optimization of continuous, discrete and categorical functions. Thanks to all this, the library can be used in any type of problem involving bounded optimization, stochastic bandits, active learning for regression, etc.

The manipulation tasks will be carried out in a virtual environment created in the Simox simulator. It is fully implemented in C++ and uses modern language features to achieve both efficiency and robustness. Furthermore, unlike other simulators, it is a lightweight and platform independent library, this is mainly due to the low number of dependencies required to compile and run and the possibility to disable certain features (such as visualization). Simox consists of three main libraries: *VirtualRobot*, provides functionalities for robot description (3D models, kinematic chains, end-effectors,...); *Saba*, implements sampling-based motion planning algorithms; and *GraspStudio*; which adds grasp planning and grasp quality evaluation functions. In addition, it is quite flexible by allowing the user to easily add new components to the simulator. Therefore, it will allow us to perform experiments in a simple way and with less execution time than using other more popular simulators (such as ROS).

Finally, an experiment execution system will be designed and implemented to allow modularly adding new experiments and optimization methods, among other things. It will also allow to save the results of each experiment for later evaluation. This system will be implemented in Python 3, since it allows us to perform a simple and fast development.

1.4 Thesis structure

The remaining of this Thesis is structured as follows. First, Chapter 2 exposes the problem of grasping objects with robotic hands. A comprehensive frame of reference on BO is explained for the unexperienced reader in Chapter 3. Then, a more precise literature review on multisolution Bayesian optimization is presented in Chapter 4. Chapter 5 explains in detail the proposed multisolution methods, as well as their implementation in the BayesOpt library. The experiment execution system, the multimodal function benchmark and the implementation of the grasp simulation environment are explained in Chapter 6. Chapter 7 discusses the results obtained from both the synthetic functions and the grasp simulation environment. Finally, Chapter 8 offers a conclusion and proposes future directions for next work.

Chapter 2

Robot grasping

First, the grasping problem to be addressed in this work is described in Section 2.1. The following section (2.2) reviews the state of the art of different grasp quality metrics and presents the metric to be used. Finally, Section 2.3 presents the robot that will be used in this work.

2.1 Grasping with robotic hands

To perform manipulation tasks correctly with robotic arms it is necessary that the grasp on the object is good enough. In addition, if the object is unknown, getting the best grasp is even more difficult. Many works rely on the use of RGB and/or depth cameras to address this problem and the use of neural networks to obtain the solution [14]. Robotic hands often include tactile sensors that provide a lot of information about the hand-object interaction, as in the case of the iCub robot. This can be exploited to address the problem of grasping objects, and without the need to use more sensors or train complex neural networks. Therefore, in this work we will exclusively use the robotic hand with tactile sensors to find the grasp on unknown objects.

Formally we can define the problem of finding the optimal grasp as finding the position and orientation (pose) of the Tool Center Point (TCP) that maximizes a grasp quality metric $\mathcal{Q}$:

$$w^* = \arg\max_{w \in W} \mathcal{Q}(w),$$

where w is the TCP pose and W is the reachable space. We can define the position of the TCP in Cartesian coordinates, spherical coordinates or joint coordinates, among others. In practice when working at a high level with manipulator robots it is more common to use Cartesian coordinates (X, Y, Z) since they facilitate the work, so they will be used in this thesis. While the orientation will be defined with (*roll*, *pitch*, *yaw*) (RPY). Therefore, we can define a pose as $w \in W \subseteq \mathbb{R}^6$.

However, the ability to grasp and manipulate objects it is also related to hand posture, which refers to the use of the intrinsic DOFs of the hand. Therefore, if we use both TCP pose and hand posture, we can define the problem as follows:

$$w^*, p^* = \arg \max_{w \in W,\, p \in P} \mathcal{Q}(w, p),\ P \subseteq \mathbb{R}^d,$$

where p is the hand posture, P is the reachable posture space and d the number of intrinsic DOFs, in the iCub $d = 9$. Optimizing hand posture is a complex task due to its high dimensionality, and increases as the robotic hand becomes more human-like. Nevertheless, there are studies that indicate that the use of subspaces (synergies) of the hand postures are enough to represent almost all possible postures [15]. But due to the lack of public data on synergies, the use of hand posture optimization has been discarded in this work. The explanation of the synergies can be found in Appendix A.

Finally, as explained in the Introduction (Section 1.1), in this work we are interested in finding a set of (optimal or suboptimal) grasps covering different regions of the object. Such regions are not differentiated by their possible utility/affordability, e.g. grasping the handle of a hammer to drive a nail, but by grasp areas. To simplify the problem, in this thesis we only consider one-handed grasps. For example, Figure 2.1 shows a picture of a watering can highlighting the different grasp regions.

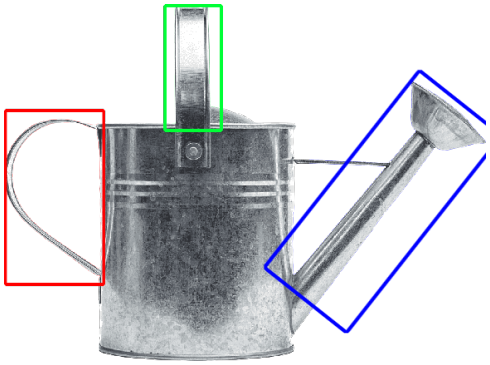


Figure 2.1: Picture of a watering can. Each colorize rectangle indicates a possible one-handed grasp region.

2.2 Grasp metrics

2.2.1 Introduction to grasp metrics

In the problem presented in the previous section, the objective function is the quality of a given relative hand-object position. There are numerous grasp quality metrics defined in the literature

[16], usually based on contact points, torques and forces applied to the object. A grasp quality metric can take into account: contact points, positions, forces and velocities. The forces acting at the point of contact are exclusively in opposition to the object. The different types of contact between the fingertips and the object can be defined as:

- ***Punctual contact without friction*:** the applied force is always normal to the contact boundary.
- ***Punctual contact with friction (hard contact)*:** the applied force has a normal component in relation to the contact boundary and may also have a tangential component.
- ***Soft contact*:** same forces as the hard contact, in addition to a torque around the direction that is perpendicular to the contact boundary.

The forces and velocities associated with the object, hand and contact points must satisfy certain relationships [17]. This relationships are summarized in Figure 2.2. The relationship between forces f and velocities ν at the fingertips and torques T and velocities $\dot{\theta}$ at the finger joints is described by the hand Jacobian J_h. On the other hand, the grasp matrix G gives the relationship between velocities ν at the contact points and the twist $\dot{x}$ (described through the translational velocity v and the rotational velocity w), and also to the relation between forces f at the fingertips and the total wrench ω applied on the object. A wrench ω is a vector defined from the force vector F and the torque vector τ acting on the object.

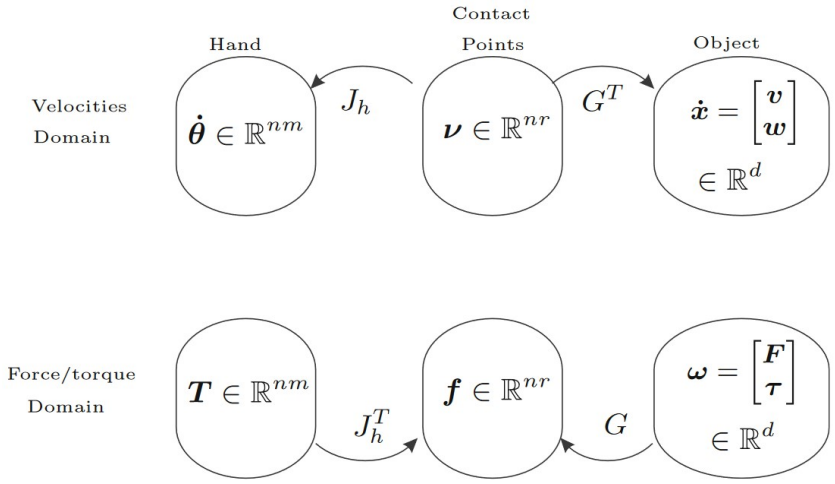


Figure 2.2: Relationship between grasp force and velocity domains [16].

The different grasp quality measures can be grouped intro three main groups [16]. The first one, *metrics associated with the position of contact points*, includes those that only take into account the properties (shape, size, weight) of the object, friction constraints and form

and force closure conditions. For example, in this group, we can find the Grasp isotropy index metric [18], with this measure we try to obtain an isotropic grasp where each contact force contributes to the internal forces of the object in a similar way. This quality measure is defined as:

$$\mathcal{Q}_{GII} = \frac{\sigma_{\min}(G)}{\sigma_{\max}(G)},$$

where $\sigma_{\min}/\sigma_{\max}$ being the minimum/maximum singular values of G. Therefore, the utility of this metric is in its ability to indicate whether a grasp is direction-independent, which is useful for general-purpose grasps.

The second one is *metrics associated with hand configuration*, which includes those that consider hand configuration to estimate the grasp quality. The basic idea from the previous group can be extending to consider the hand-object Jacobian H [19], which allows direct transformation in the velocity domain from the hand joint space $\dot{\theta}$ to object space $\dot{x}$. In this group we can find the Uniformity of transformation metric, which is conceptually equivalent to $\mathcal{Q}_{GII}$. If the contribution of each joint velocity is the same in all the components of $\dot{x}$, then the transformation in the velocity domain from the finger joints to the object is uniform. This means that the hand can move the object in any direction with the same gain and, therefore, a good manipulation. The measure is defined as:

$$\mathcal{Q}_{UOT} = \frac{\sigma_{\min}(H)}{\sigma_{\max}(H)}.$$

However, it may happen that, for example, ignoring the hand's actual geometry when selecting optimal contact points on the object surface using any criteria from the first group could result in contact locations that are not physically feasible for the real hand, and vice versa: an optimal hand configuration could generate a weak grasp in the presence of small perturbations. Therefore, the third group includes *approaches that combine different quality measures*, allowing to capture different aspects of prehension [20], such as geometric constraint, ability to resist forces, manipulability or comfort. A simple method uses the algebraic sum of the results of each individual measure [21, 22]

2.2.2 Epsilon metric

After the review of the state-of-the-art on grasp quality measures, and since we are only going to use tactile information for grasping and the hand posture is not going to be used, it has been decided to use the *epsilon* metric [23], which is also a popular metric in research.

This metric belongs to the first group and is based on the convex hull of the *Grasp Wrench space* (GWS), specifically the radius, ϵ, of the largest 6D ball centered at the origin, that can be

enclosed in the hull. The vector from the origin of the GWS to the point where the ball contacts the hull boundary is the smallest maximum wrench that can be applied by the grasp. The closer ϵ is to 1 the more efficient the grasp is. In addition, in case of collision with the object or no contact with its surface, the ϵ will take the value 0.

To compute the GWS this metric assumes a constraint in the finger forces, so the sum of the force modules applied by n contact points is limited, corresponding to a limited common power source for all fingers. Therefore, the total force f is defined as:

$$f = \sum_{i=1}^{n} ||f_i|| \leq 1$$

Now, assuming a Coulomb friction model, the total force f, acting on the object at a contact point must lie within a cone that has a vertex at the contact point, an axis along the contact normal and a half angle of $\tan^{-1} \mu_s$, being μ_s the coefficient of static friction. To find the total GWS we will need a finite set of basis vectors. To achieve that it is necessary to approximate this cone with a pyramid of m edges. Therefore, the force f_i applied by the i-th contact point can be expressed as a positive linear combination of unitary forces $f_{ij}, 1 \leq j \leq m$ along the pyramid edges. Hence f can be rewritten as:

$$f = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} f_{ij},$$

$$and \;\; \alpha_{ij} \geq 0, \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} \leq 1.$$

The total wrench ω acting on the object can be defined in the same way as f:

$$\omega = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} \omega_{ij}.$$

and, finally, the set of all possible wrenches is obtained by the convex hull of all ω_{ij}:

$$W = CH\left(\cup_{i=1}^{n} \{\omega_{i1}, ..., \omega_{im}\}\right).$$

Castanheira et al. [4] propose a variation of the *epsilon* metric, adding a Collision Penalty (CP) that accounts for the level of penetration of the hand in the object and is used only as a heuristic to improve the convergence speed of the optimization. CP is defined as:

$$CP = 1 - e^{-\lambda n},$$

where λ is a tuning parameter used to smooth the penalty and n is the number of joints in the robot's hand that collide with the object. Therefore, the final metric is:

$$\mathcal{Q} = \mathcal{Q}_\epsilon + CP.$$

Although this variation will not be used, it is explained in this work since it may be of interest for future work.

2.3 iCub robot

In this thesis we will work with the research-grade humanoid robot iCub [24], which is designed to support research in embodied AI. The iCub is distributed as open source. Natale et al. [25] reviewed the history and evolution of the iCub humanoid platform. Figure 2.3 shows a picture of the robot.

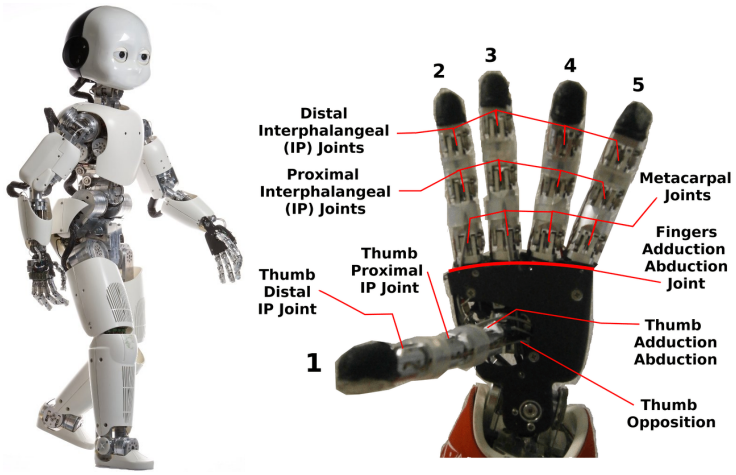


Figure 2.3: A picture of the iCub robot and its hand.

The overall size (104cm tall and weighs 30kg) and general kinematics of the iCub is similar to those a five-year-old child. It can crawl on all fours, walk and manipulate objects. It has full-body sensitive skin (more than 3000 tactile sensors) to deal with physical interaction with the environment and its hands have been designed to perform sophisticated manipulation tasks. In addition, the robot is equipped with cameras, microphones, inertial sensors, joint sensors and force/torque sensors. Figure 2.4 summarizes the degrees of freedom of the iCub.

Component	# of degrees of freedom	Notes
Eyes	3	Independent vergence and common tilt
Head	3	The neck has three degrees of freedom to tilt, swing and pan
Chest	3	The torso can also tilt, swing and pan
Arms	7 (each)	The shoulder has 3 DoF, 1 in the elbow and three in the wrist
Hands	9	The hand has 19 joints coupled in various combinations: the thumb, index and middle finger are independent (coupled distal phalanxes), the ring and little finger are coupled. The thumb can additionally rotate over the palm.
Legs	6 (each)	6 DoF are sufficient to walk.

Figure 2.4: Degrees of freedom of the iCub robot [24].

Chapter 3

Bayesian optimization

This chapter presents a summary of the theoretical foundations on which this work is based. Most of this chapter is based on two tutorials on Bayesian optimization [26, 27] and the book *Gaussian Processes for Machine Learning* [28], among others.

3.1 Overview

Formally, Bayesian optimization tries to find the global optimum (x^*) of an unknown and costly function $f : X \to \mathbb{R}$, where X is a compact space, $X \subset \mathbb{R}^d$, and $d \geq 1$ its dimension,

$$x^* = \arg\max_{x \in X} f(x),$$

performing sequentially evaluations/observations on f, having as limit a maximum of T evaluations. Moreover, such observations may be noisy. It is typically assumed that the black box function f does not have a simple closed-form, but can be evaluated at any point $x \in X$.

BO methods are some of the most efficient approaches in terms of the number of function evaluations required to find the global optima. The efficiency of BO is largely due to its ability to incorporate prior belief about the problem to guide sampling and to trade-off exploration and exploitation of the search space. BO takes its name because it uses **Bayes' Theorem**, which states that the *posterior* probability of a model (or theory, or hypothesis) M given some evidence (or data, or observations) E is equal to the *likelihood* of E given M multiplied by the *prior* probability of M:

$$P(M|E) = \frac{P(E|M)P(M)}{P(E)}.$$

In BO, M is the objective function f to be optimized and E is the set of observations or samples obtained so far, that is $\mathcal{D}_{1:t} = \{(x_i, f(x_i))_{i=1}^t\}$. Then, the *prior* $P(f)$ represents our belief about the space of possible objective functions and the *likelihood* $P(\mathcal{D}_{1:t}|f)$ represents how plausible the data we have sampled are given our *prior* knowledge about the model. Now,

we can combine these to obtain the *posterior* distribution:

$$P(f|\mathcal{D}_{1:t}) = \frac{P(\mathcal{D}_{1:t}|f)P(f)}{P(\mathcal{D}_{1:t})}.$$

The posterior distribution captures the updated belief about the unknown function, which helps us to discern which functions are most likely to correspond to the objective function given previous observations. Another interpretation is the estimation of the objective function by means of a ***surrogate (probabilistic) model***, that learns the properties of f given previous observations and, with each new observation, the model is sequentially refined by Bayesian posterior updating. Gaussian processes (GP) are a common choice as a *surrogate model* in BO, explained in Section 3.2.

To guide the search of global optimum, BO obtains the next point to be evaluated $x_{t+1} \in X$ by maximizing an **acquisition function** $\alpha(\cdot)$. These kind of functions are heuristics that trade-off *exploration*, evaluating points where the uncertainty of the *surrogate model* is high; and *exploitation*, evaluating points where the *surrogate model* expects high values. This balance between exploitation and exploration is key to finding the global optima efficiently. This concept is explained in detail in Section 3.3.

Summarizing, BO can be divided into 2 stages:

- **Learning stage:** BO learns the *surrogate model* from available information.
- **Decision stage:** BO decides the next sample from the *surrogate model* by maximizing the acquisition function.

Both stages are executed in loop for T iterations. At the end of the process, the observed point with the highest value is returned as the optimal solution.

3.2 Gaussian process

A Gaussian process is a probabilistic model used to define a distribution over random functions. This means that a GP is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution [26]. A GP is fully specified by its mean function $\mu : X \to \mathbb{R}$, which determines the center of the distribution; and kernel or covariance function $k : X \times X \to \mathbb{R}$, which determines the level of similarity between different input points. Therefore, we can write the objective function f as follows:

$$f(x) \sim \mathcal{GP}\left(\mu(x), k(x, x')\right),$$

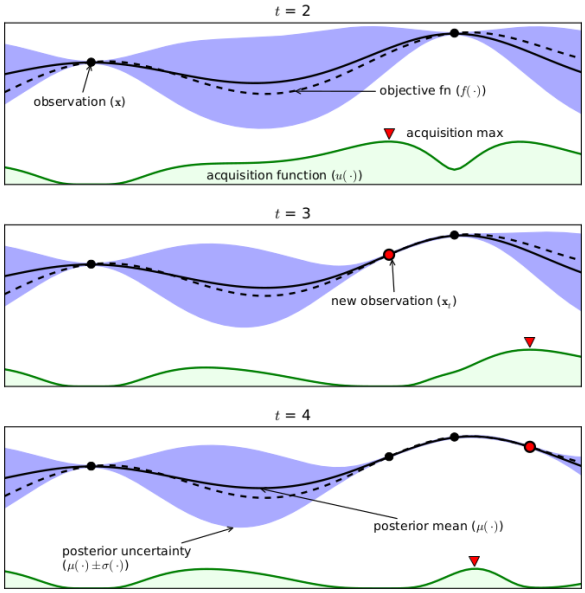


Figure 3.1: Example of BO on a toy 1D problem [26]. The figure shows the objective function (dashed line), the acquisition function at the bottom and the GP model, with the mean (solid line) and the uncertainty (shaded area), in 3 iterations. The acquisition function is high where the GP predicts a high value (exploitation) and where the uncertainty is high (exploration), areas with both attributes are sampled first.

then when evaluating any $x \in X$ in the GP, we obtain the mean and variance of a normal distribution over the possible values of f at x. A more in-depth explanation about GP and its applications in machine learning can be found in [28].

For convenience, it is assumed that the prior mean is the zero function $\mu(x) = 0$, although there are different alternatives.

The covariance function plays a key role in the GP, as it determines the smoothness of the functions that can be drawn from it. A popular choice for the kernel is the *squared exponential function*, however sample functions with this kernel are unrealistically smooth for practical optimization problems. Another important kernel in machine learning, as suggested by Sonek et al. [29], is the 5th order *Matérn ARD*, which allows greater flexibility in modelling functions. The "ARD" stands for "Automatic Relevance Determination", i.e. the kernel uses independent parameters for each dimension of the problem, allowing to find the relevance of the each feature in the input space.

$$k_{v=5/2}(x, x') = \theta_0 \left(1 + \sqrt{5r^2(x, x')} + \frac{5}{3}r^2(x, x')\right) \exp\left(-\sqrt{5r^2(x, x')}\right),$$

where θ_0 is the covariance amplitude and $r^2(x, x') = (x - x')^T M(x - x')$. Finally, M is a positive semidefinite diagonal matrix where each value is a different length-scale for each dimension.

The hyperparameters have to be learned based on the evidence using techniques such as *maximum likelihood* (ML) or *Markov Chain Monte Carlo* (MCMC). In BO, with each iteration we obtain new evidence, therefore it is necessary to re-learn the hyperparameters in order to best fit the updated information. In comparison between the two techniques: MCMC is more robust because it works with several of the possible hyperparameter values, i.e. it uses n GPs, so it is much slower. Whereas ML uses only the most likely one for the estimation of the hyperparameters, but the computation overhead is minimal.

To obtain the next sample to be evaluated in the objective function x_{t+1}, we need to obtain the predictive distribution based on the posterior distribution in the current iteration t, which will allow us to discuss the value of any x in $t+1$. Assuming that the following data $\mathcal{D}_t = \{(x_i, y_i)_{i=1}^t\}$, where $y_i = f(x_i)$, has been evaluated based on independent observations $y_i \sim \mathcal{N}(f(x_i), \sigma^2)$, and as a result of the Bayes rule, we obtain [28]:

$$P(f_{t+1}|\mathcal{D}_{1:t}, x_{t+1}) = \mathcal{N}\left(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})\right),$$

where

$$\mu_t(x_{t+1}) = \mu(x_{t+1}) + k^T K^{-1} f(x_{t+1})$$
$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - k^T K^{-1} k,$$

and $k = [k(x_{t+1}, x_i)]_{x_i \in \mathcal{D}_t}$, the correlation vector of size t; and $K = [k(x_i, x_j)]_{x_i, x_j \in \mathcal{D}_t}$, the kernel matrix of size $t \times t$. Therefore, from $\mu_t(\cdot)$ and $\sigma_t^2(\cdot)$, we can obtain the mean and variance of the predictive distribution at each point in the space, which will be used by the acquisition functions (Section 3.3).

3.3 Acquisition function

Using the posterior distribution of the *surrogate model*, we sequentially induce the acquisition functions $\alpha_t : X \to \mathbb{R}$, to evaluate the utility of the candidate points for the next evaluation of f. Where t indicates the implicit dependence of the available data ($\mathcal{D}_{1:t}$). Therefore, the next point to evaluate is the one that maximizes its utility:

$$x_{t+1} = \arg\max_{x \in X} \alpha_t \left(x, P(f|D_{1:t})\right).$$

Unlike the original objective function f, $\alpha(\cdot)$ is known and inexpensive so it can be sampled efficiently. We can use different algorithms to optimize $\alpha(\cdot)$, for example: *DIRECT* [30], *Monte Carlo* [31] or *Multistart* [32]. In the case of BayesOpt, it uses a combination of a global search (*DIRECT*) and a posterior local refinement (*BOBYQA* [33]), both derivative-free.

Many acquisition functions have been proposed in the literature, all attempting to balance exploitation and exploration. Shahriari et al. group them into three types [27]:

- ***Improvement-based policies*:** these functions favour points that are likely to improve upon an incumbent target τ. For example, the function *probability of improvement* (PI) [34] tries to maximize the probability of improvement over the current best solution ($\tau = y_t^* = \max(y_1, ..., y_t)$):

$$\alpha_{PI}(x; \mathcal{D}_{1:t}) := P(y \geq y_t^*) = \Phi\left(\frac{\mu_t(x) - y_t^*}{\sigma_t(x)}\right),$$

where Φ is the standard normal cumulative distribution function. However, this strategy is pure exploitation and, in addition, all improvements are treated equal and PI simply accumulates the posterior probability mass above τ at x. A better acquisition function is *expected improvement* (EI) [35]. This function incorporates the amount of improvement over y_t^* or zero otherwise:

$$\alpha_{EI}(x; \mathcal{D}_{1:t}) := \mathbb{E}[\max(0, y - y_n^*)] = (\mu_t(x) - y_t^*)\Phi\left(\frac{\mu_t(x) - y_t^*}{\sigma_t(x)}\right) + \sigma_t(x)\phi\left(\frac{\mu_t(x) - y_t^*}{\sigma_t(x)}\right),$$

when $\sigma_t(x) > 0$ and where ϕ is the standard normal probability density function.

- ***Optimistic policies*:** the *upper confidence bound* criterion (UCB) [36] is a popular way of trade-off exploration and exploitation, often with provable cumulative regret bounds. The principle of this type of strategy is to be optimistic in the face of uncertainty, in the sense of considering the best scenario for a given probability value. Srinivas et al. [37] proposed the *Gaussian process upper confidence bound* (GP-UCB) algorithm:

$$\alpha_{UCB}(x; \mathcal{D}_{1:t}) := \mu_t(x) + \beta\sigma_t(x),$$

where β is used to adjust the exploitation-exploration, although it may be difficult to find an appropriate value.

- ***Information-based policies*:** Unlike the other acquisition functions discussed above, information-based policies take into account the posterior distribution over the optimal value x^*, denoted $P_*(x|D_{1:t})$. The posterior over objective functions f implicitly determines this distribution. There are two subpolicies: Thompson sampling (TS) [38] and entropy search (ES). The first one is a randomized strategy which samples a reward function from the posterior and selects the arm with the highest simulated reward. TS was extended to continuous search spaces [39] and can be formulated as:

$$\alpha_{TS}(x; \mathcal{D}_{1:t}) := f^{(t)}(x)$$

$$\text{with } f^{(t)} \overset{s.s.}{\sim} GP(\mu_0, k|\mathcal{D}_{1:t}),$$

where $\overset{s.s.}{\sim}$ indicates approximate simulation via spectral sampling [40, 41, 42]. Instead

of sampling the distribution P_*, ES methods select the points that maximize the information gain about the location of x^*, i.e., they select the point that is expected to cause the largest reduction in entropy of the distribution P_* [43, 44, 45]. However, this function is intractable for continuous spaces, so approximations must be made, but this approximation methods are computationally expensive.

Chapter 4

Related work

This chapter first presents an overview of multisolution Bayesian optimization. Then, the following sections briefly summarize each of the multisolution Bayesian optimization methods that have been found in the literature, in chronological order.

4.1 Multisolution Bayesian optimization

The multimodal optimization problem (for maximization) can be defined as follows: We would like to find all global and local maximums of a function $f : X \to \mathbb{R}$ in a single run. Therefore, the function f will have:

- Local maximums: a local maximum $\hat{x_l} \in X$ of a function f is an input element with $f(\hat{x_l}) \geq f(x)$ for all x neighboring $\hat{x_l}$. If $X \in \mathbb{R}^d$, we can write:

$$\forall\ \hat{x_l}\ \exists\ \epsilon > 0 : f(\hat{x_l}) \geq f(x)\ \forall x \in X,\ |x - \hat{x_l}| < \epsilon.$$

- Global maximums: a global maximum $\hat{x_g} \in X$ of a function f is an input element with $f(\hat{x_g}) \geq f(x),\ \forall x \in X$.

Formally, these functions f are called multimodal, which are non-convex and have one or more global and local optima.

This type of problem has been extensively studied in evolutionary algorithms (EAs), a review of the state of the art can be found in [46, 47]. So popular are these methods that a benchmark was even created for a EAs competition [48], which will also be used in this work. However, these methods require many function evaluations and are therefore less efficient than BO methods.

On the other hand, the multisolution problem has barely been researched in the field of BO. During the state-of-the-art research stage of this work, only three works about multisolution

Bayesian optimization (MBO) have been found and their are very recent [9, 10, 11], which will be discussed in the following sections 4.2, 4.3 and 4.4, respectively. In this thesis we are going to focus on finding diverse solutions, which can be both global and local optimums, as is done in MBO related works. Figure 4.1 shows an example of a representation of an optimal multisolution in a grasping problem.

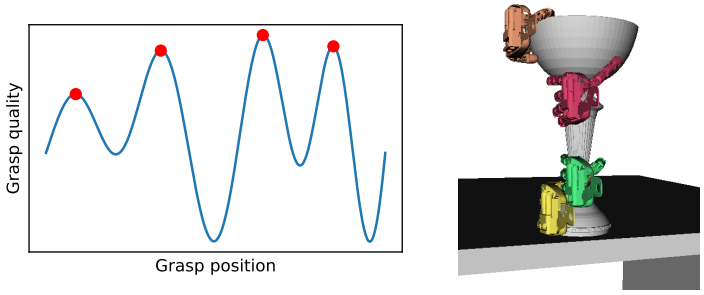


Figure 4.1: Optimal multisolution, with both local and global optimums, where the objective is to find the multiple grasps that obtain high values of quality metrics.

By default, BO only finds a single solution, so using the M best evaluations during optimization does not guarantee sufficient diversity. It is also useless to select M evaluations at different points in space, as they do not guarantee a good metric. It is necessary to use a variant of BO that takes into account multisolution, i.e. actively balances finding solutions with high metric values and evaluating sufficiently diverse points.

In the state-of-the-art we can find variants of BO that allow working in "parallel", i.e. they evaluate a batch of points in each iteration. To do this, a local repulsion is introduced around an already chosen point to force the next point to be different, when the batch is being sampled. In addition to trying to ensure that the points to be evaluated will improve the current optimum. This can be achieved by replacing the value of an already chosen point with a low constant value [49] or by calculating a penalty around already chosen points from the *Lipschitz constant* [50]. This active search for diversity is similar to what we want to achieve in MBO and therefore parallel BO can be a good starting point.

4.2 BO approaches for massively multimodal problems

Roman et al. [9] presented in 2019 three different BO methods to solve the multimodal problem in the CEC2013 Benchmark [48] (CEC13). They propose methods based on combining BO with local search techniques and all of them use the UCB acquisition function:

1. **Sequential BO:** this algorithm is inspired in the "forgetting" Subset of Data approach [51]. Compared to the original BO method, this algorithm applies a local search at each

point proposed by BO, in order to exploit each possible solution. Then, the *surrogate model* is updated with the point suggested by BO and the best point achieved by the local search. Finally, it also limits the size of the dataset, i.e. when the dataset reaches the maximum number of points (N), the oldest point is discarded when updating the *surrogate model.* This is the simplest, but also the most computationally inexpensive of the three algorithms.

2. **Adaptive BO:** this second algorithm avoids unmanageable growth of the covariance matrix by splitting the current set of solutions into different clusters. It is based on the previous one, but it aims to take more advantage of the knowledge acquired during the optimization process. This approach divides the dataset into several subsets using clustering techniques. This way, to obtain the conditional distribution of the GP given some solution, only the closest solutions will be taken into account, reducing the size of the covariance matrix. It can be seen as a single *surrogate model* that is composed by several GPs, each one with an exclusive subset of the dataset.

3. **Batch BO:** this algorithm is a modified version of "Batch BO via Local Penalization" [50]. It is similar to the first one but is intended to be evaluated in parallel. This technique iteratively selects a batch of solutions by optimizing an acquisition function that penalizes the previous selections in the batch. First, a solution is selected according to UCB. Then a batch of solutions is obtained iteratively, selecting a solution and penalizing the acquisition function according to this selection. To adapt the acquisition function, a penalization ball is applied for each solution already selected for the batch.

Each of the methods is run on functions of different dimensionality and running the maximum number of evaluations allowed, as indicated in CEC13. The paper shows how the results obtained by the three methods are similar, none of them standing out from the rest.

However, its use as a comparative method with the one to be developed in this thesis is ruled out. This is because their code is not available and there is no time to implement the three methods in another library. Furthermore, due to lack of resources, in this thesis it is not possible to run the maximum number of iterations indicated by the benchmark, so the metrics in the paper are not useful.

4.3 SigOpt: Multisolution experiment

SigOpt [52] is an optimization-as-a-service platform offering different types of experiments. Among these experiments are two that return multiple solutions: *Multisolution* [10] (SO-MS)

and *All Constraints* [53] (SO-CAS). These two methods were already used in our preliminary work about MBO [8]. But for this thesis we have discarded the use of SO-CAS because from the results obtained in that work it was seen that this method simply returned all solutions above the set threshold value, regardless of the diversity.

Although it is not possible to know how SO-MS works, their blog hints that they use an ensemble of strategies to achieve diversity and quality, including a constraint active search approach similar to that done by Malkomes et al. [54]. This method requires to specify the number of solutions to find.

4.4 Discovering Many Diverse Solutions with BO

Recently, in October 2022, Maus et al. [11] presented the Rank-Ordered Bayesian Optimization with Trust-regions (ROBOT) method, which aims to find a set of diverse and quality solutions in high-dimensional functions.

This method is based on multiple trust regions, where each region represents a possible solution, with the same length adjustment dynamics as in Eriksson et al. [55]. To measure diversity, they use a symmetric, user-supplied function $\delta(\cdot)$ defined on pairs of points in the search space X. Two pairs of points are considered diverse if $\delta(x_1, x_2) \geq \tau$, where τ is set by the user. Formally, they seek a sequence $S^* = \{x_1^*, ..., x_M^*\}$ so that:

$$x_1 = \arg\max_{x \in X} f(x)$$

$$x_i = \arg\max_{x \in X} f(x) \; s.t. \; \delta(x_1, x_2) \geq \tau \; for \; 1 \leq j < i.$$

Therefore, the multiple optima x_i^* in S^* are defined hierarchically. In order to find a set of M solutions, ROBOT maintains M simultaneous local optimization runs using M individual non-stationary trust regions, but using only one global *surrogate model*. Each local run i aims to find a single diverse solution x_i^*, which together form the desired set S^*. Each solution x_i^* is only constrained with respect to prior solutions ($j < i$), therefore ROBOT assigns a hierarchical rank-ordering to the M trust regions.

In each iteration of optimization ROBOT select candidates $\hat{x}_i$ from each trust region to improve over its local incumbent (x_i^+, y_i^+) using a similarly hierarchically-constrained acquisition function. Finally, a modified Thompson sampling procedure is used to sample points from the acquisition function. The maximum number of points to be evaluated per solution in each iteration can be set with the parameter BS.

ROBOT has been evaluated in different high-dimensional tasks with varying numbers of solutions. The results show a good performance, achieving both good quality and diversity.

Chapter 5

Multisolution Explorative Bayesian Optimization

5.1 Overview

Looking at the MBO methods explained in Chapter 4 we can see how the first one does not require specifying neither the number of solutions to find nor a minimum level of diversity. The method focuses on exploring enough the space to find the maximum number of solutions. Then, the SO-MS method uses a parameter to determine the number of solutions to find, just like ROBOT. In addition, ROBOT requires specifying a minimum level of diversity between solutions and a diversity function to compare two points in the space.

Based on this, it has been decided to first implement a base model, *Multisolution Explorative Bayesian Optimization* (MEBO) (Section 5.2), which requires neither the number of solutions nor the minimum level of diversity. This method is intended for the case in which the user does not know the number of different solutions needed or how to determine the diversity among them. Furthermore, unlike methods that seek M solutions it may be able to provide many more solutions allowing the user to choose between them as requirements change in the future. Although, there will be many solutions that do not reach the optimal value.

On the other hand, an extension of MEBO has been implemented that focuses on finding M solutions with a minimum level of diversity and using clustering algorithms, *Cluster MEBO* (CL-MEBO) (Section 5.3).

The implementation of these methods is explained in Section 5.4. In addition, they are open source and are available in a fork of BayesOpt repository [56].

Also add that the first proposal was to start working from Local Penalization method [50] but due to problems with the implementation in BayesOpt that could not be solved, its use was discarded. Although the code is still available in the repository.

5.2 Base model

Many acquisition functions are designed to trade-off exploration and exploitation, as mentioned in Section 3.3. However, this trade-off is based on the assumption that the surrogate model is good enough to predict both the expected function value and its uncertainty and can be a problem, for example in the first iterations the model is too inaccurate to give correct predictions. For inaccurate models, the acquisition function is still able to provide some information, but by greedily selecting a single value we are wasting some of that information, there may be various modes that are also of interest but which are ineligible. This can be seen in Figure 5.1, which shows an example of how the greedy policy only cares about the central mode of the acquisition function, while the other two on the left are almost equally interesting to explore. This is very important for our objective since it will allow us to further explore the search space, thus obtaining potential areas with solutions.

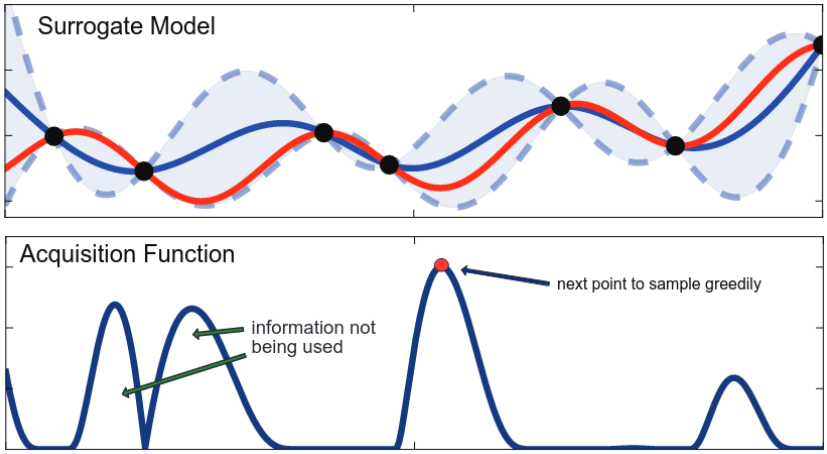


Figure 5.1: Surrogate model (blue) and target function (red). The greedy policy selects the maximum of the acquisition function (red dot) and ignores the rest of the regions which are almost as valuable [57].

In this context, Garcia-Barcos and Martinez-Cantin [57] consider BO as a *partially observable Markov decision process* (POMDP) and, introduce the idea of stochastic policies, specifically the Boltzmann policy, for BO with the objective to perform fully distributed BO. In addition, this concept was also used to achieve robust policy search for robot navigation [58]. As will be explained later, this method will allow us to solve the above problem by performing an active search for diverse but optimal (or suboptimal) points. Therefore, it has been decided to use this method as the base model for MEBO.

First, the connections between BO framework and POMDP framework can be found in Appendix B. Second, we replace the greedy policy for selecting the next query in the acquisition function by the stochastic Boltzmann policy:

$$p(x_{t+1}|y_{1:t}, x_{1:t}) = \frac{e^{\beta_t \alpha(x_{t+1}, p(f|y_{1:t}, x_{1:t}))}}{\int_{x \in \mathcal{X}} e^{\beta_t \alpha(x_{t+1}, p(f|y_{1:t}, x_{1:t}))} dx}$$

This policy defines a probability distribution for the next query or action. Thus, the actual next query is selected by sampling that distribution $x_{t+1} \sim p(x_{t+1}|y_{1:t}, x_{1:t})$. This policy allows exploration even if the model is completely biased or overconfident, thus allowing us to obtain points in different modes of the distribution (acquisition function). This becomes even more important in MEBO since searching for multiple optima requires a deeper exploration of the search space than in standard BO. In addition, this approach can be applied to any acquisition function or surrogate model that can be found in the literature. Since it is based on sampling from a Boltzmann policy, we will refer to this method of query selection as *Boltzmann selection*.

However, sampling from this distribution is not trivial. The Metropolis-Hastings algorithm was used for this purpose, which is a Markov chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution from which direct sampling is difficult. Although this algorithm allows us to sample efficiently, most acquisition functions are highly multimodal with many large areas of low probability between modes. In addition, some acquisition functions tend to have larger areas of low probability as optimization progresses, as is the case of EI, which makes the situation worse. This is a problem because in order to have variety in the batch it is necessary to skip these large zones, this means that to obtain each query it is necessary to discard many samples, which increases the cost in time. In addition, as the number of dimensions increases, the number of discarded samples must increase. The number of samples to be discarded per dimension, called *burning samples*, can be adjusted by the user.

In addition, sampling using this Boltzmann selection ensures diverse queries with random numbers, allowing us to run the optimization process in parallel and fully distributed [57]. Thus, a centralized node is not required and all computation can be done in each node, unlike many parallel or batch BO methods. In terms of communication, the nodes only need to broadcast their latest evaluated query and observation value, requiring minimal communication bandwidth. This communication can be asynchronous and it is robust to delays or failures in the network, as the order of the queries and observations is irrelevant. Therefore, we could deploy nodes more exploitative or more explorative.

In MEBO, neither distributed nor asynchronous optimization will be used, but a batch sampling approach. Although, it will take advantage of the ability to use any acquisition function to divide the batch into more exploitative and more explorative points, this is explained in Section 7.2 as it is an experimental setup. In addition, since Boltzmann selection allows the use of any surrogate model, we have decided to approximate the posterior distribution of the model to learn the kernel hyperparameters using an empirical Bayesian approach, which allows us to save computation time at the cost of being less robust. Furthermore, a standard zero mean Gaussian process with noisy observations has been used as a surrogate model. This GP will use a 5th order *Matérn ARD* kernel, which is a popular choice as mentioned in Section 3.2.

Finally, another version of MEBO has been designed but it is more explorative, E-MEBO. Here *explorative sampling* is used, i.e. for each point in the batch, T points of the same acquisition function are sampled with the Boltzmann selection, thus obtaining a large set of possible candidates. Then, the final batch will be composed by the B points of the sampled set that maximize the Euclidean distance between them. E-MEBO uses this technique in each iteration until 40% of the total available function evaluations are reached, then it is only applied in 1 out of 3 iterations. This allows for a more exhaustive exploration of the space.

5.3 Clustering approach

In the case where the user only needs M solutions, it is necessary to focus the optimization process on finding those solutions, performing local refinements. To do this it is necessary to divide the dataset $\mathcal{D}_{1:t}$ into M regions, using a clustering algorithm, and sample points from them. The clustering algorithm needs that the size of $\mathcal{D}_{1:t}$ is sufficiently large and diverse, so a similar strategy to E-MEBO is followed, using explorative sampling until 40% of the total number of function evaluations allowed is reached and another 10% with Boltzmann selection. Subsequently, local refinement is carried out with *Cluster batch sampling* (Section 5.3.1), interspersed with iterations with Boltzmann selection. As more evaluations of the function have been consumed, the *Cluster batch sampling* is applied in more iterations. Finally, the set of M solutions is chosen from the total set of evaluated points $\mathcal{D}$ (Section 5.3.2). The CL-MEBO algorithm in pseudocode can be found in Appendix C.

5.3.1 Cluster batch sampling

This process can be divided into three stages: *cluster computation*, *cluster selection* and finally *cluster sampling*. Figure 5.2 summarizes the cluster batch sampling procedure.

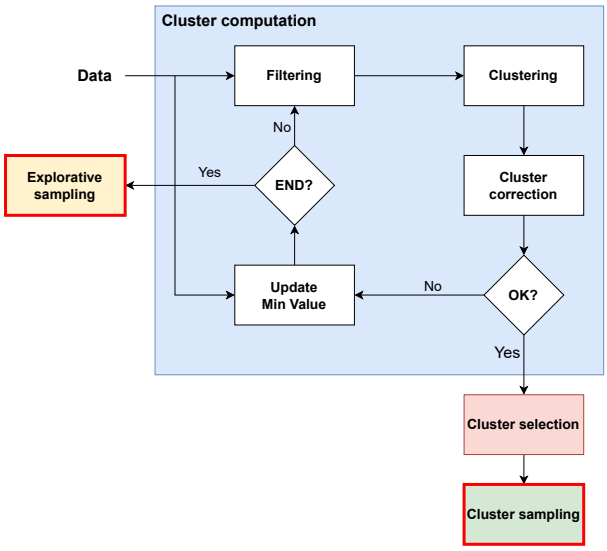


Figure 5.2: Cluster batch sampling procedure. Each colored rectangle represents a different stage of this procedure. The two rectangles with a red border indicate the end of the process.

Cluster computation

Since all the points of $\mathcal{D}_{1:t}$ are not of interest due to their low value, the first thing to do is to remove those points whose value is less than a limit L. This is the *Filtering* step. To obtain L we use the distribution of values of $\mathcal{D}_{1:t}$, $Y = \{y \in (x, y) \ \forall \ (x, y) \in \mathcal{D}_{1:t}\}$, and then sort Y in ascending order and divide it into 10 equal parts, i.e. we calculate the deciles, D_i with $1 \leq i \leq 9$. Since we are interested in solutions with the highest possible value, the procedure starts with the ninth decile, so $L = D_9$.

Once the size of $\mathcal{D}_{1:t}$ was reduced, we applied a clustering algorithm to divide the set into M clusters, each one representing a possible solution area. For this purpose, the *kmeans++* algorithm [59] is used to obtain a first solution and subsequently refined using the *xmeans* algorithm [60] with *Bayesian information criterion* as stop criteria. The distance between points is measured with the Euclidean distance, having previously normalized its dimensions between $[0, 1]$.

The objective of step *Cluster correction* is to validate that the clusters have enough diversity. Two clusters are considered distinct if the distance between their centroids is greater than or equal to the minimum level of diversity. Then, the cluster sets that do not meet the diversity condition are merged. This step is detailed in Appendix C. Figure 5.3 shows an example of cluster correction.

Finally, if the number of clusters, after correction, is equal to M, then we proceed to the *Cluster selection* stage. Otherwise, the process is repeated using the next decile, $L = D_{i-1}$. If $L == D_1$ and $3M/4 \leq |clusters| < M$, then we select the set of clusters with the largest size

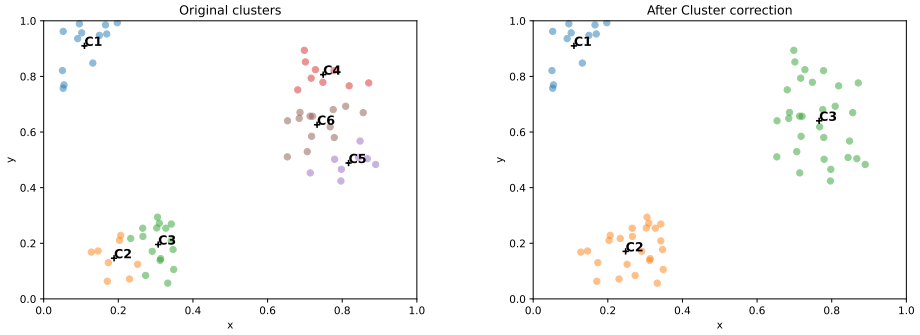


Figure 5.3: Cluster correction in an example problem where the objective is to find 6 solutions with a minimum level of diversity of 0.2. Cluster 1 (C1) does not change since it meets the requirements. C2 and C3 merge because their centroids are not enough diverse. On the other hand, the C4-C6 and C6-C5 pairs have low diversity and, although C4 and C5 are diverse, all three merge into one since they share a lack of diversity with C6.

and from the largest decile, else further exploration is needed and explorative sampling is used.

Cluster selection

Since the batch size is limited, and M is likely to be larger than B, it is necessary to select a subset of clusters to sample. In addition, as each cluster may have different characteristics, such as a smaller size or a lower maximum value, it is necessary to set a priority order when selecting the clusters. This prioritization is based on a scoring system that takes into account the features of the clusters.

These features are independent of each other Therefore, a scoring function, $S : \mathcal{C} \to \mathbb{R}$ with $\mathcal{C}$ the set of clusters, can be defined for each feature, where the value returned by S is normalized. Hence, the final score of a cluster $C \in \mathcal{C}$ can be defined as:

$$CS(C) = \sum_{i=1}^{F} S_i(C),$$

where F is the number of features. The following characteristics were used to compute the score:

- ***Value score*** **(V):** if a cluster has a smaller maximum value than the global optimum found in any cluster ($G_{\max}$) or the mean of the maximum value of the clusters (G_μ), then it should have a higher priority, since it implies that it may be *under-exploited*. Therefore, we can define V as the distance to G_μ and then normalize it, i.e. we use mean normalization:

$$V(C) = \frac{G^C_{\max} - G_\mu}{G_{\max} - G_{\min}},$$

where $G^C_{\max}$ is the maximum value of C and $G_{\min}$ is the minimum of the maximum of the clusters.

- ***Size score* (W):** if one cluster has a smaller size, a smaller number of points, than another, this may imply that the region is *under-explored* and should therefore have higher priority. This function is defined, as in V, with the mean normalization:

$$W(C) = \frac{L^C_{\max} - L_\mu}{L_{\max} - L_{\min}},$$

where L indicates the cluster size.

Since mean normalization has been used, the clusters we are interested in sampling first will have lower and negative scores. Thus, we transform the scores to the $\mathbb{N}$ domain,

$$CS'_C = CS(C) + |\min(0, \min\{CS(C)\ \forall\ C \in \mathcal{C}\})|,$$

and then flip them so that the highest scores are the lowest, so the final score will be:

$$CS_C = \max\{CS'_C\ \forall\ C \in \mathcal{C}\} - CS'_C.$$

Finally, the clusters are selected in order of priority until the batch is completed. In addition, when a cluster is selected, its score is updated and penalized:

$$CS_C = CS_C(1 - 0.1B_C),$$

where CS_C is the current score of C and B_C is the number of times C was selected.

Cluster sampling

In this last stage, we sample points for each selected cluster with the acquisition function *Cluster EI* (CL-EI), which is a modified version of the EI. For each selected cluster, if B_C is greater than 1, then explorative sampling with CL-EI is used; otherwise, only one point is sampled directly.

CL-EI acts as a local EI, i.e. it measures the amount of improvement of each point with respect to the cluster being sampled C_s. For this purpose, the function needs as input the C_s and for each cluster: the centroid and its maximum value. Then, CL-EI is defined as:

$$CL - EI(x) = EI(x, G^{C_s}_{\max}) \frac{\min\{||x - C||_2\ \forall\ C \in \mathcal{C}\}}{||x - C_s||_2},$$

where $G^{C_s}_{\max}$ is the maximum value of C_s and the fraction is a factor to penalize points that are closer to another cluster. This acquisition function requires setting the limits of the search space to the bounding box built from the points of C_s.

5.3.2 Solution set selection process

This is the final process of CL-MEBO, here we are interested in finding a set of M solutions that maximize diversity and quality given the dataset $\mathcal{D}$. The set of possible solutions is $S = \{X \in \mathcal{D}^M | (x_i, y_i) \neq (x_j, y_j)\ \forall\ (x_i, y_i), (x_j, y_j) \in X\}$. Now, given a function $QD : X \to \mathbb{R}$ that evaluates the quality and diversity of a set of points X, we look for the optimal set that maximizes this function, $S^* = \arg\max_{X \in S} QD(X)$.

Performing a brute force approximation on $\mathcal{D}$ to find S^* is not feasible, since as the size of $\mathcal{D}$ increases the size of S increases considerably, namely it would be $\frac{|\mathcal{D}|!}{M!(|\mathcal{D}|-M)!}$. Therefore, we use the same clustering approach as in cluster batch sampling to divide $\mathcal{D}$ into M regions, although in this case we do not merge clusters considered non-diverse. Then, we evaluate combinations of size M by taking one point from each cluster. In this way we reduce the number of combinations significantly, being the worst case when the whole $\mathcal{D}$ is used and divided into M equal parts, $(\frac{|\mathcal{D}|}{M})^M$, although for much larger datasets than those used in this work it also becomes computationally expensive.

Defining the QD function is not a simple task and may depend on the problem being addressed. In the problem of finding multiple grasps, using $d_{\min}(x, X) = \min\{||x - x'||_2 | x' \in \{X - x\}\}$, we can define *diversity* as the average minimum Euclidean distance between solutions ($\overline{d}_{\min}$) plus to a factor that penalizes short distances. On the other hand, we can define *quality* as the mean grasp quality of X ($\overline{y}$) plus the minimum value ($y_{\min}$). Therefore, QD is defined as:

$$QD(X) = M_d(\overline{d}_{\min} + \frac{(d^{\min}_{\min})^2}{d^{\max}_{\min}})(y^X_{\min} + \overline{y}),$$

where M_d is the number of diverse solutions in X, $d^{\min}_{\min}$ is the minimum $d_{\min}$ in X and $d^{\max}_{\min}$ is the maximum $d_{\min}$ in X.

5.4 Implementation in BayesOpt

To implement the algorithms in the BayesOpt library, a new fork was created to avoid conflicts and errors in the original repository. Subsequently, the batch BO via Local Penalization method was implemented, for which it was necessary to read the original paper and review its implementation in Python [61] in order to port it to C++ in our library. In BayesOpt there was already the possibility of batch sampling in each iteration, but it was simulated, i.e., each point was sampled and evaluated sequentially and, therefore, it is not possible to run the points in parallel. Therefore, the first task performed was to change this to actually allow running the batch in parallel. For this purpose, internal modifications have been made to the library, and so

that this new feature can also be used by other languages, two new interfaces have been created, one for C and one for Python.

Second, the acquisition function Local Penalization was implemented and can be found in the file *criteria_lp.hpp*. This acquisition function, among other things, estimates the Lipschitz constant, in the original implementation this was done by taking maximizing the norm of the expectation of the gradient of the surrogate model. Since the gradient of the surrogate model was not yet implemented in BayesOpt and required many changes to be made, we first proceeded to use a brute force approximation to estimate the gradient. Several tests of the implementation were made and everything worked correctly, but the problems appeared when including it in BayesOpt. Here the Lipschitz values were not calculated correctly causing the batch points to be practically equal, this can be seen in Appendix D. After a long time of unsuccessful attempts to fix it, we proceeded to continue with the approach of Section 5.2.

The explorative sampling has also been implemented in the library. First, we obtain all possible combinations of points without repetition, and then obtain the one that maximizes the Euclidean distance, if the number of points were very large it could be implemented in parallel but so far it has not been necessary.

On the other hand, Boltzmann selection was already implemented in BayesOpt. First, the starting point for this selection process is obtained with DIRECT-BOBYQA optimization, which starts with a random point in the search space. Although for CL-MEBO it was necessary to modify the code to limit the search space, both for Boltzmann selection and DIRECT-BOBYQA, for the bounding box of the corresponding cluster. In addition, the starting point of DIRECT-BOBYQA in CL-MEBO is the centroid of the cluster to be sampled.

To implement CL-MEBO, we first made a preliminary version in Python, since development is much faster and also allowed us to test variations of the algorithm quickly. This version was tested with data collected from several executions of MEBO and E-MEBO. Subsequently, once its correct performance was verified, we proceeded to implement it in C++ in BayesOpt. Everything was implemented from scratch, except the kmeans++ and xmeans algorithms, since it would have required a lot of time and there are already implemented and very efficient libraries. Therefore, we decided to use the *PyClustering* library [62] for both algorithms, moreover it is the same as the one used for the Python version.

Finally, a new criteria, *BatchCombinedCriteria*, had to be implemented to allow the use of different acquisition functions in the same batch. This criteria takes as input the number of acquisition functions to be used and for each one: the function parameters, if any; and the number of points to be sampled with it. Therefore, this criteria is responsible for initializing and configuring the different acquisition functions, and selecting the correct one to sample the

next point in the batch. In addition, it was also necessary to implement the CL-EI criteria, which, as explained in Section 5.3.1, takes as input the cluster centroids and the current cluster to be sampled.

Chapter 6

Experiment execution system

This chapter describes the Experiment execution system (EES) developed in this thesis to carry out different types of experiments. Section 6.1 explains the modular architecture implemented, Section 6.2 summarizes the implementation of the CEC13 benchmark in the EES. Finally, the implementation of the grasp simulation environment (GSE) is explained in Section 6.3. All the work developed for this system is available in an open source repository on Github [63]. To facilitate the use of this system, the repository includes a *requirements* file, with the Python dependencies; and a tutorial, explaining step by step how to install, compile and use the EES.

6.1 Architecture

Figure 6.1 shows an overview of the different components that constitute the EES and a brief communication flow diagram showing the modularity of the EES. In the following figures, the dashed lines represents that a package (or module, class...) uses another package (or module, class...). Whereas continuous lines with a white head represent that one class is a subclass of another. Text in italics and underlined represent abstract classes.

This section explains the *OptSystem* package, which contains all the necessary components to run and save any experiment, this package and its modules are purely implemented in Python 3. Specifically, this section explains the *Core* and *Optimizers* modules, while the *CEC13* and *GraspOpt* modules are explained in Section 6.2 and 6.3, respectively. Finally, the *Third-party components* is a representation of all external services or libraries used in the EES.

6.1.1 Core module

The *Core* module, as its name suggests, is the core of the package and allows to create experiments in a flexible way by combining different metrics, objective functions and optimizers. Figure 6.2 shows the different classes and their relationships that constitute the *Core* module.

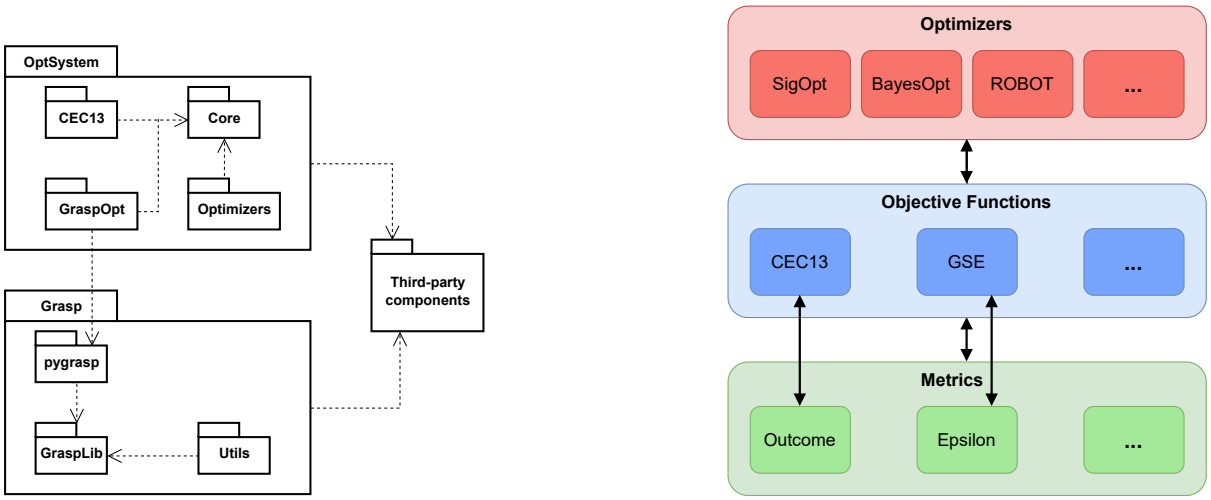


Figure 6.1: EES overview. Component diagram on the left. Brief communication flow diagram on the right.

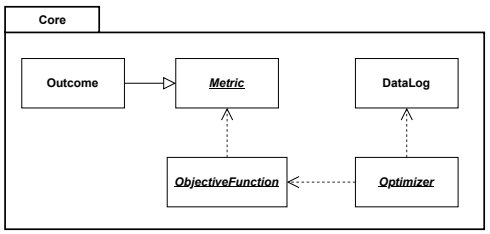


Figure 6.2: Class diagram of the *Core* module.

The *Metric* class represents any type of metric of any type of problem and when is instantiated (in a subclass) represents the result of an evaluation of the objective function. Classes that inherit from it must implement a method to pass from the result of the objective function to the float type, which is the one supported by the optimizers. In addition, it allows to add metadata to the result, for example the execution time or the grasp position error. The metric *Outcome* represents the most basic metric, it only receives a float and return the same float and does not allow to add metadata.

The *ObjectiveFunction* class represents the objective function to be optimized. It takes as input a *Metric* and optionally a dictionary with the parameters of the objective function. The classes that inherit from it must implement the abstract method *evaluate* that will execute the function to be optimized. It also allows to execute queries one at a time or in parallel, in the case of batch BO.

The class *Datalog* is in charge of saving, in a *json* file, the progress, parameters and additional information of an experiment. It also allows loading the contents of a file to analyze the results or, if allowed by the optimizer, to resume an experiment that has not been completed.

Finally, the *Optimizer* class represents an optimization method, such as BayesOpt. Its main

function is to act as a bridge between the objective function and the optimizer. It also sends the results, parameters and other information to the *DataLog*, if saving is enabled. The subclasses must be responsible for initializing the optimization method and communicate correctly with it.

6.1.2 Optimizers module

This module contains the wrapper classes that connect the different optimization methods to the EES. The following have been used in this thesis: BayesOpt, SigOpt and ROBOT. Figure 6.3 shows the class diagram of this module.

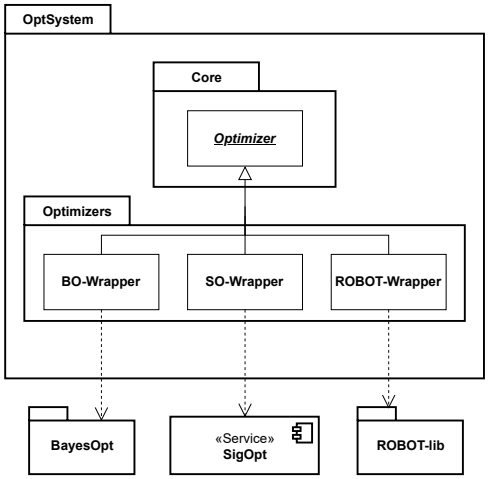


Figure 6.3: Class diagram of the *Optimizers* module.

BO-Wrapper works directly with the Python interface provided by the BayesOpt library itself, which communicates directly with the C++ compiled library. This interface only requires the dependency on the popular NumPy library.

To work with the SigOpt service it is necessary to use its API, for which they offer two options: to call the API from scratch or to use directly its library, which can be found on their website. The first option is designed for companies that want more flexibility to integrate the service into their systems. Whereas the second option includes everything needed to use the service at the highest level and is therefore easier to use. For this reason, the *SO-Wrapper* works with the second option.

The ROBOT method is available in a public repository [64]. The implementation is written in Python using BoTorch and GPyTorch libraries. It was necessary to make some changes to the repository because several bugs were found during the execution of test experiments. These changes do not affect the logic of the ROBOT method itself. The modified ROBOT implementation can be found in the EES repository.

All optimizers can be customized through an input parameter, of dictionary type, which contains the necessary attributes to configure each one. In addition, all optimizers were thoroughly tested for correct operation before running the final experiments.

6.2 CEC2013 Benchmark

This section first presents a brief summary of the CEC13 benchmark [48] and then describes how it has been implemented in the EES. This benchmark

Although the state-of-the-art in EAs for multimodal optimization has been extensively studied, researchers encountered several obstacles when comparing algorithms: many studies focused on low-dimensional functions, some methods introduced new parameters that were difficult to configure, or different functions or modifications of existing functions were used. Therefore, the objective of this benchmark is to create a unifying framework for evaluating algorithms in multimodal problems. For this purpose, the benchmark collects numerous functions with different characteristics and dimensions. This benchmark is used in this work as test functions before moving on to experiments in the GSE. Finally, the CEC13 metrics and functions to be used in this work are explained in Section 7.1.

The benchmark can be found in a public repository, referenced in the original paper, and includes the implementation in different languages, such as Python and C++. As EES is implemented in Python, the same language was selected for CEC13. In addition, the original repository has been simplified, removing all unnecessary languages and data, and added to the EES repository.

Figure 6.4 shows the class diagram of the *CEC13* module. The *CEC13-Wrapper* class acts as a bridge between the modified CEC13 repository (*CEC13-lib*) and EES. The benchmark function to be used in an experiment is set by an input parameter in the wrapper class. As metric it uses the basic one provided by the *Core* module, since no additional information needs to be stored and the result returned by the *CEC13-lib* is a float.

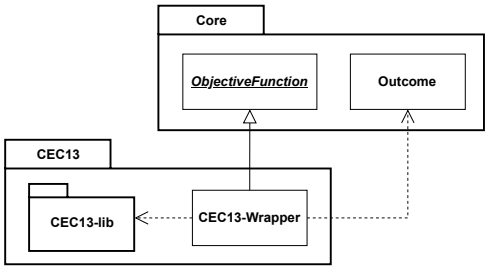


Figure 6.4: Class diagram of the *CEC13* module.

6.3 Grasp simulation environment

The GSE, whose class diagram can be seen in Figure 6.5, has been developed in Simox. Recently, the Simox project has been migrated to a new repository [65], so the one referenced in the original article [13] is no longer maintained. Therefore, in order to correctly develop the simulation environment, it is first necessary to review the Simox documentation. Additionally, the simulator includes a variety of tutorials, with source code available, on different topics, such as trajectory planning.

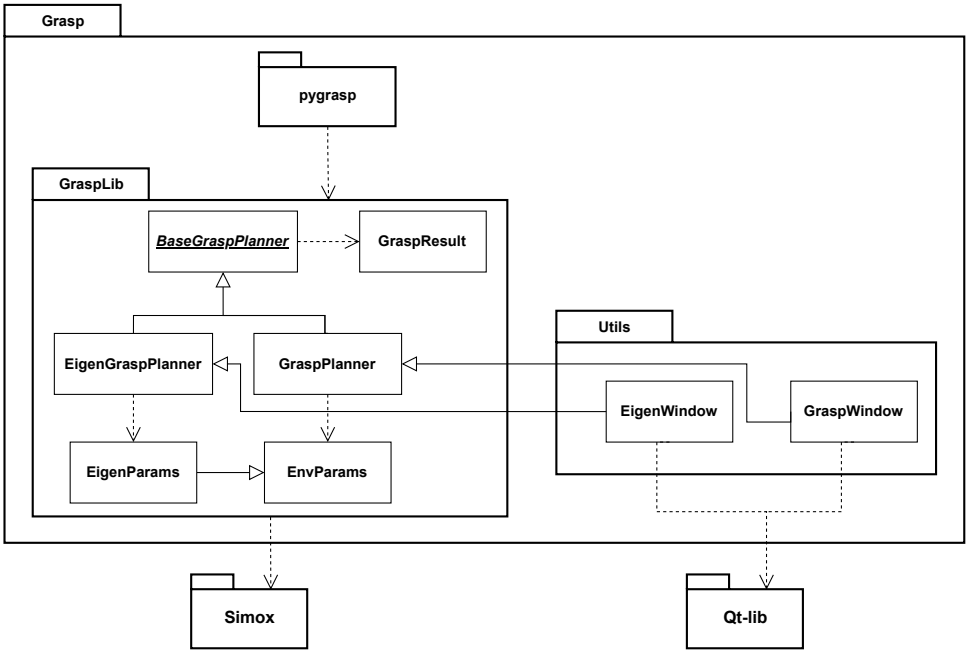


Figure 6.5: Class diagram of the *Grasp simulation environment*.

As in this work we are only interested in grasping with robotic hands using tactile sensors and in order to simplify the simulation, only the robot's hand will be used. In addition, only the target object to be grasped will be in the simulation.

Two planners have been implemented, but the basic grasping process (implemented in *BaseGraspPlanner*) is the same for both. The grasp is performed when the end-effector is placed in the desired final position. If in this position the end-effector is in collision with the object, then the process terminates and the epsilon metric is 0. Otherwise, the fingers of the robotic hand are closed synchronously and each of them stops when it comes into contact with the object surface or when it reaches its joint limit. When the finger movement is finished, the grasp quality metric is computed. The class *GraspResult* stores the epsilon metric in addition to the volume metric [23], whether the grasp is force closure and whether the grasp produces collision, among other things.

The *GraspPlanner* planner receives as input the global position, in Cartesian coordinates; and the orientation, in radians in (roll, pitch, yaw) format; of the TCP. Each time a new target position is received, the relative position of the end-effector is calculated and then the end-effector is placed at the new location. On the other hand, the *EigenGraspPlanner* planner acts in the same way as *GraspPlanner*, but differs from it in that it also receives as input the amplitude vector of the eigengrasps (explained in Appendix A). In addition, before closing the hand, the preshape is calculated from the amplitude vector and the eigengrasps, then the fingers are set in the corresponding preshape position.

In order to view the simulation environments, in addition to being able to load the results of the experiments, a viewer has been created for each planner that makes use of Qt, which is a UI library. Both are located in *Utils*. Figure 6.6 show these viewers with the iCub hand and a bottle object.

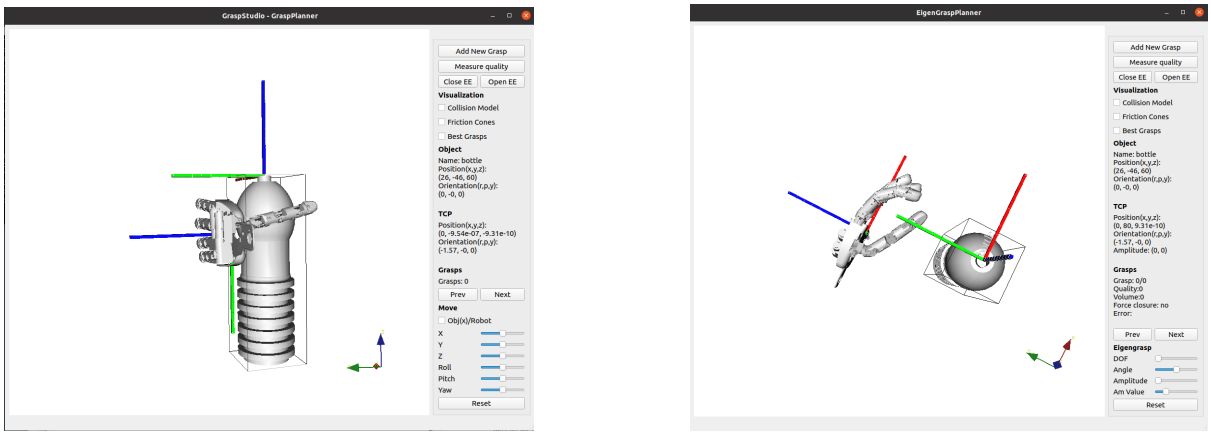


Figure 6.6: **(Left)** *GraspPlanner* viewer. **(Right)** *EigenGraspPlanner* viewer. Both with the iCub hand and a bottle object.

Finally, in order to perform experiments with the planners, it was necessary to create an interface in Python, *pygrasp*, with the SWIG tool, which allows to connect programs written in C/C++ to other languages. This interface is used by the *GraspOpt* module (Figure 6.7), which acts as an intermediary between the grasp planners and the *OptSystem*. The *EpsilonMetric* class parse the grasp result (*GraspResult* type) received from the planner to float, and adds the metadata it stores.

The *GraspPlannerWrapper* class is responsible for initializing and configuring the grasp planners, as well as sending and receiving information correctly. It accepts as input parameter the name of the planner to be used: “GP” for *GraspPlanner* and “EGP” for *EigenGraspPlanner*, as well as a path to the file with the planner parameters.

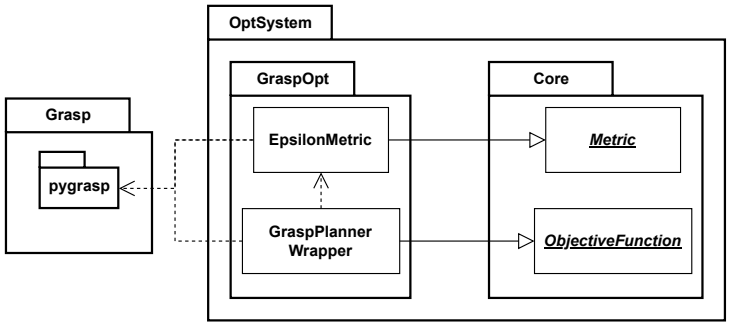


Figure 6.7: Class diagram of the *GraspOpt* module.

Chapter 7

Experiments and results

This chapter first presents in Section 7.1 the CEC13 metrics and functions used in this work. Section 7.2 explains the setup of the MEBO-based methods to be used in the experiments. Finally, the results obtained with the MBO methods in CEC13 and GSE are shown in Sections 7.3 and 7.4, respectively. All the experiments and results are also available in the EES repository [63].

7.1 CEC13 Benchmark

This benchmark has a total of 12 multimodal functions (from 1 to 20 dimensions). In addition, some of these have different dimensional configurations, each dimension having a different number of global optima, so the total number of functions is 20.

The objective is to find the maximum number of global optima of each function, so local optima are not taken into account, although for some problems it would be useful to locate them, such as in grasping. Figure 7.1 shows two functions of this benchmark and Table 7.1 shows the characteristics of the functions used in this thesis.

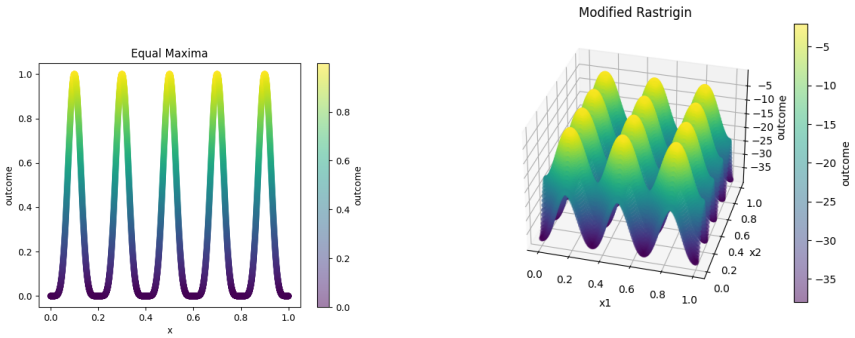


Figure 7.1: Two functions of the CEC13 benchmark. Left function: 1D with 5 global optima. Right function: 2D with 12 global optima.

Function	#Dim	Bounds	Global optima	# global optima	# local optima
Uneven Maxima	1	$x \in [0, 1]$	1.0	1	4
Five-Uneven-Peak Trap	1	$x \in [0, 30]$	200.0	2	3
Equal Maxima	1	$x \in [0, 1]$	1.0	5	0
Himmelblau	2	$x, y \in [-6, 6]$	200.0	4	0
Modified Rastrigin	2	$x, y \in [0, 1]$	-2.0	12	0
Vincent	2	$x, y \in [0.25, 10]$	1.0	36	0

Table 7.1: Characteristics of the CEC13 functions used in this work.

The CEC13 sets a maximum number of function evaluations and defines three metrics to evaluate the performance of the algorithms:

- **Peak ratio (PK):** measures the average of all known global optima found over multiple runs.

$$PR = \frac{\sum_{i=1}^{NR} NPF_i}{NKP_f * NR},$$

  where NPF_i denotes the number of global optima found in the end of the i-th run, NKP_f the number of known global optima of function f, and NR the number of runs.

- **Success rate (SR):** measures the percentage of successful runs (a successful run is defined as a run where all known global optima are found) out of all runs.

$$SR = \frac{NSR}{NR},$$

  where NSR is the number of successful runs.

- **Convergence speed (CS):** average number of function evaluations (over the total allowed) required to locate all known global optima.

$$CS = \frac{\sum_{i=1}^{NR} \frac{FE_i}{MaxFE_f}}{NR},$$

  being FE_i the number of function evaluations used in the i-th run and $MaxFE_f$ the maximum number of function evaluations allowed for function f. If the algorithm cannot locate all the global optima in the i-th run then $FE_i = MaxFE_f$.

As in this work we will use far fewer function evaluations than those specified in CEC13 (>50k), due to the computational cost involved in BO, a global optimum is considered to be reached if the value of the nearest point is at most 5% less than the optimum. Therefore, in addition to the previous metrics, the *Mean of global optima found* (MGOs) will also be used.

7.2 Experimental setup

The Boltzmann selection used in MEBO allows the use of any acquisition function, but choosing the ideal acquisition function for this problem is not an easy task. The improvement-based acquisition functions allow us to obtain points where the optimum is expected to improve, but they fail when trying to explore the space, which is a problem because we have less knowledge of the objective function and therefore it is difficult to find new optima. On the other hand, optimistic acquisition functions try to balance more exploration and exploitation but may nevertheless fail to obtain the best results.

As this method uses a batch of B points in each iteration, we can address the exploration-exploitation problem using different acquisition functions for each point or set of points. Therefore, we can divide the batch into *exploitative points*, more focused on improving the solution candidates; and *explorative points*, more focused on exploring the space. A good choice for exploitative points is to use EI, and UCB for explorative points, with $\beta \geq 2$ to allow more exploration. Since in multisolution the search space exploration is a bit more important than obtaining the best solution and UCB also has exploitation, a good way to distribute the points is to use about 70% in the explorative set.

The improvement of using this strategy versus using a single acquisition function (EI or UCB) will be analyzed below. All parameters are the same for each model. The number of burning samples has been set to 250. The surrogate model is relearned every 5 iterations. The β parameter of UCB has been set to 2. All methods will perform a total of 25 initial evaluations, sampled with Latin Hypercube sampling (LHS), and 225 evaluations in batches of $B = 3$. This configuration will also be used in the final experiments. In the case of the combined strategy, one point is sampled with EI and the others with UCB. The models will be compared using the functions and metrics of the previous section. Tables [7.2, 7.3] show the results for one-dimensional and two-dimensional multimodal functions in 10 runs, respectively.

	Uneven Maxima				Five-Uneven-Peak Trap				Equal Maxima			
	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
EI	0.700	0.700	0.819	**1.000**	0.750	0.500	0.806	**198.62**	**1.000**	**1.000**	0.938	**1.000**
UCB	**1.000**	**1.000**	**0.485**	0.999	**0.900**	**0.800**	0.886	197.09	**1.000**	**1.000**	**0.857**	0.998
EI-UCB	**1.000**	**1.000**	0.622	**1.000**	0.850	0.700	**0.800**	198.60	**1.000**	**1.000**	0.904	**1.000**

Table 7.2: Results in 1D CEC13 functions with the different acquisition functions.

	Himmelblau				Modified Rastrigin				Vincent			
	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
EI	0.325	0.000	1.000	198.53	0.241	0.000	1.000	**-2.0139**	**0.225**	0.000	1.000	**0.981**
UCB	0.875	0.500	**0.855**	196.43	0.058	0.000	1.000	-2.0366	0.113	0.000	1.000	0.974
EI-UCB	**0.925**	**0.800**	0.923	**198.81**	**0.491**	0.000	1.000	-2.0173	0.150	0.000	1.000	0.978

Table 7.3: Results in 1D CEC13 functions with the different acquisition functions.

As can be seen in Table 7.2 UCB achieves good results, although it fails slightly to obtain solutions close to the optimum. On the other hand, EI fails in the first two functions by not being able to find all the solutions and, in addition, it is the slowest in locating all the optima. However, it finds solutions closer to the optimum. Now if looking at EI-UCB we see how it takes advantage of the best of both acquisition functions: it is able to find the solutions approximately as well as UCB and obtaining the best optima as EI.

This behavior can also be seen in Table 7.3, where EI-UCB achieves the best results in 2 of the 3 cases. However, in this case UCB does not perform as well as in the one-dimensional functions, especially in the case of Modified Rastrigin.

If we now analyze the batch diversity in different functions, Table 7.4, we will see how UCB-based acquisition functions achieve higher diversity and thus exploration. This table shows the mean variance per dimension between the batch points.

	Himmelblau		Vincent		Modified Rastrigin	
	x	y	x	y	x	y
EI	0.411	0.284	5.131	5.131	0.013	0.012
UCB	7.575	6.965	5.431	5.173	0.055	0.052
EI-UCB	7.394	6.235	5.009	5.479	0.057	0.063

Table 7.4: Mean of the batch variance per dimension in 2D-functions.

7.3 Results on CEC13 Benchmark

For this comparison, the same functions and metrics from Section 7.1 have been used. The number of solutions M to be found has been set to the number of global optima of each function for SO-MS, ROBOT and CL-MEBO. Although for the *Vincent* function this number has been reduced to 20 for the last two, due to problems with the methods when optimizing the function. In addition, as SO-MS and CL-MEBO requires that $M > 1$, then for *Uneven Maxima* function $M = 2$. On the other hand, the minimum level of diversity between solutions for ROBOT and CL-MEBO is specified in Table 7.5. Finally, the diversity function of ROBOT is $\delta(\cdot) = ||\cdot||_2$, and $BS = 2$ for 1D functions and $BS = 1$ for 2D functions, because M increases significantly.

Function	τ
Uneven Maxima	0.15
Five-Uneven-Peak Trap	15
Equal Maxima	0.15
Himmelblau	2.0
Modified Rastrigin	0.25
Vincent	2.0

Table 7.5: Minimum diversity level between solutions for CEC13 functions.

All methods will execute a total of 250 function evaluations, of which 25 correspond to initial points except for SO-MS which does not require this parameter. Each experiment was run a total of 10 times. Table 7.6 shows the results obtained in 1D functions, while Table 7.7 shows the results in 2D functions.

	Uneven Maxima				Five-Uneven-Peak Trap				Equal Maxima			
	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
MEBO	**1.000**	**1.000**	0.708	**1.000**	0.900	0.800	0.774	198.045	**1.000**	**1.000**	0.923	**1.000**
E-MEBO	**1.000**	**1.000**	0.693	**1.000**	**1.000**	**1.000**	0.753	199.481	**1.000**	**1.000**	0.903	**1.000**
CL-MEBO	**1.000**	**1.000**	0.511	**1.000**	**1.000**	**1.000**	0.821	**199.996**	**1.000**	**1.000**	0.926	**1.000**
SO-MS	**1.000**	**1.000**	**0.356**	0.996	**1.000**	**1.000**	**0.163**	199.864	**1.000**	**1.000**	**0.725**	0.997
ROBOT	0.900	0.900	0.763	**1.000**	0.800	0.600	0.805	196.893	0.940	0.700	0.848	0.993

Table 7.6: Results in 1D CEC13 functions with the different MBO methods.

	Himmelblau				Modified Rastrigin				Vincent			
	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
MEBO	**1.000**	**1.000**	0.936	199.94	0.916	0.500	0.948	-2.006	0.180	0.000	1.000	0.981
E-MEBO	0.975	0.900	0.965	199.43	0.783	0.100	0.993	-2.016	0.147	0.000	1.000	0.975
CL-MEBO	**1.000**	**1.000**	0.985	**199.99**	**1.000**	**1.000**	**0.740**	**-2.0006**	0.155	0.000	1.000	0.976
SO-MS	**1.000**	**1.000**	**0.341**	199.98	0.658	0.000	1.000	-2.041	**0.222**	0.000	1.000	**0.991**
ROBOT	0.800	0.300	0.956	196.28	0.025	0.000	1.000	-2.036	0.089	0.000	1.000	0.977

Table 7.7: Results in 2D CEC13 functions with the different MBO methods.

From Table 7.6 we can observe how all optimizers achieve a similar MGOs. SO-MS is the fastest optimizer, as evidenced by its low CS, having in the worst case a $1.16\times$ speedup (SO-MS vs ROBOT in *Equal Maxima*). In addition, both SO-MS and E/CL-MEBO are always able to find all global optima. Moreover, MEBO manages to find all the optima in 2/3 cases, but fails in the second function, which is difficult to optimize due to its peaks and where it is likely that more exploration is needed to avoid getting trapped. Finally, ROBOT is unable to find all the optima in any case, but the results it obtains are similar in all three functions.

In Table 7.7 we can see how the average performance of the optimizers decays as M increases, this is more noticeable in ROBOT. This poor performance of ROBOT may be due to the fact that it is designed for high dimensionality functions and therefore more iterations are used. Moreover, in each iteration it evaluates points of each solution region and, therefore, as the number of solutions increases and when the maximum number of function evaluations is small, it reaches this maximum earlier and is not able to learn the surrogate model as well . The best performing method on average is CL-MEBO, which is able to find all optima in the first 2 functions. Surprisingly, the base model MEBO performs quite well in the first two functions, obtaining results close to CL-MEBO. Moreover, it is the MEBO-based method that finds more optima in the last function. E-MEBO also performs well in these two functions, however it is not able to find all the optimums, this may be due to the fact that it is more explorative and, therefore, to the lack of refinement on the solutions found. On the other hand, the results of *Vincent* show that optimizers require a larger budget to locate all its global optima.

7.4 Results on Grasp simulation environment

To compare the MBO methods, the following experiments were carried out. *TCP position* in Cartesian coordinates (3D) optimization, keeping the orientation of the hand pointed and parallel to the surface of the object, i.e. perpendicular to the Z-axis (height). The limits of the search space for X (width) and Z axes are set to the object dimensions, while for the Y-axis (approximation) the limits extend from the object surface to the plane in which there is no contact between the end-effector and the object. *TCP pose* (6D) optimization, the limits for RPY are set to plus/minus $[45°, 90°]$ with respect to the orientation perpendicular to the surface. For both 3D and 6D the posture remains the same, with fingers extended. These experiments were run on three objects of different shape and size, which can be seen in Figure 7.2. For the first two objects M was set to 4 and for the last one to 3. The minimum diversity level was set to 40 for ROBOT and CL-MEBO. In addition, ROBOT was configured with $\delta(\cdot) = ||\cdot||_2$ and $BS = 2$.

Each experiment was run 10 times, with 250 and 500 function evaluations for 3D and 6D, respectively; and always with 25 initial points. The results are divided into three analysis: best grasp found (Section 7.4.1), final solution set (Section 7.4.2) and best grasp with occlusions (Section 7.4.3).

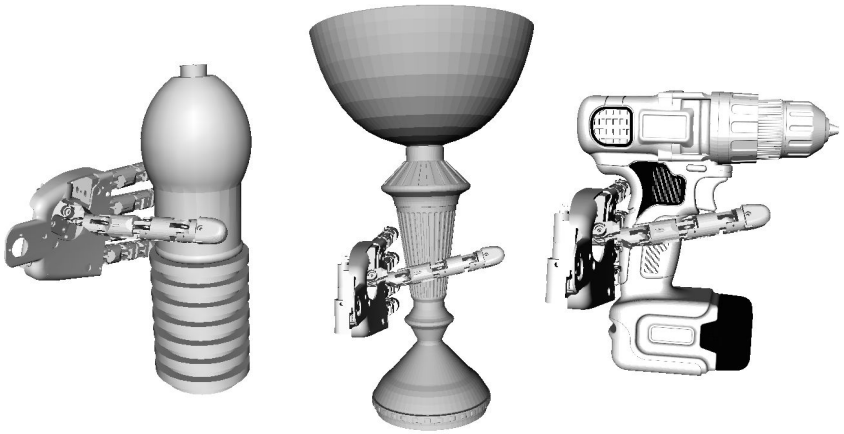


Figure 7.2: Objects used in simulation for grasp evaluation. From left to right: bottle, vase, drill.

7.4.1 Best grasp found

Here the mean best grasp found by each optimizer is compared, these results can be found in Table 7.8. While Figures [7.3, 7.4, 7.5] show the evolution of the best grasp found during the optimization process for bottle, vase and drill objects, respectively. The first thing that can be observed is the difference in quality between the different objects, this is because the epsilon metric varies according to the shape and size of the object and, therefore, it does not mean, for example, that the best grasp on the vase object is worse than best one on the bottle.

Then we can see how the best grasp achieved by all optimizers is similar in 3D. Although the standard deviation of the results shows a significant difference in stability between SO-MS and the others, this can also be seen in the figures, where SO-MS has a high standard deviation during the whole process.

On the other hand, the results in 6D are quite different from the previous ones. In the bottle object all optimizers obtain a worse grasp than in 3D, the worst case being the MEBO-based models. This may be due to the fact that by adding orientation, more uniform objects (bottle) are more difficult to optimize, however in more irregular objects (vase) this benefits them as they can reach areas that position alone cannot. In addition, it is curious how SO-MS in 6D experiments significantly reduces its variation during the process compared to 3D.

	Bottle		Vase		Drill	
	TCP pos	TCP pose	TCP pos	TCP pose	TCP pos	TCP pose
MEBO	0.817 ±0.030	0.736 ±0.045	0.274 ±0.018	0.341 ±0.018	0.211 ±0.005	0.211 ±0.011
E-MEBO	**0.834 ±0.016**	0.690 ±0.067	0.276 ±0.020	0.347 ±0.021	0.216 ±0.004	0.213 ±0.015
CL-MEBO	0.809 ±0.024	0.723 ±0.037	0.269 ±0.020	0.350 ±0.019	0.220 ±0.005	0.217 ±0.010
SO-MS	**0.834 ±0.044**	0.752 ±0.049	**0.288 ±0.027**	0.349 ±0.023	0.216 ±0.022	0.234 ±0.012
ROBOT	0.829 ±0.011	**0.790 ±0.029**	0.287 ±0.022	**0.382 ±0.031**	**0.222 ±0.006**	**0.240 ±0.009**

Table 7.8: Mean and standard deviation of the best grasp quality (epsilon metric) found in each experiment.

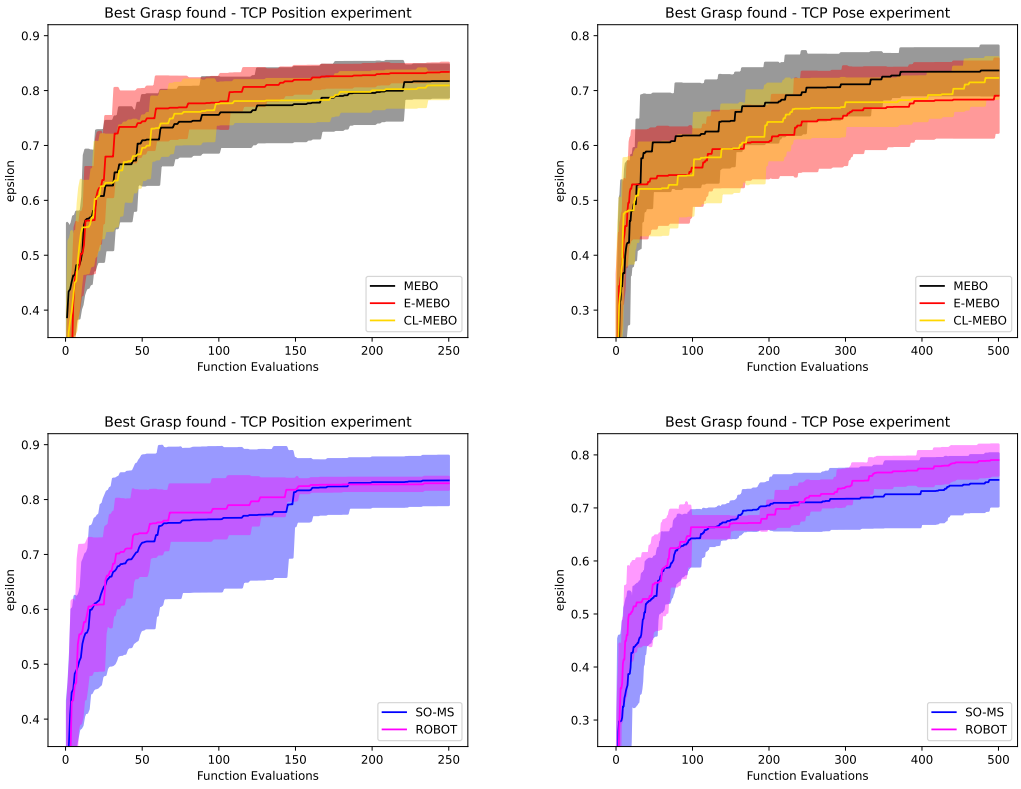


Figure 7.3: Evolution of the best grasp quality found in the bottle object in TCP position (Left) and TCP pose (right) optimization. Top row: MEBO, E-MEBO, CL-MEBO. Bottom row: SO-MS, ROBOT.

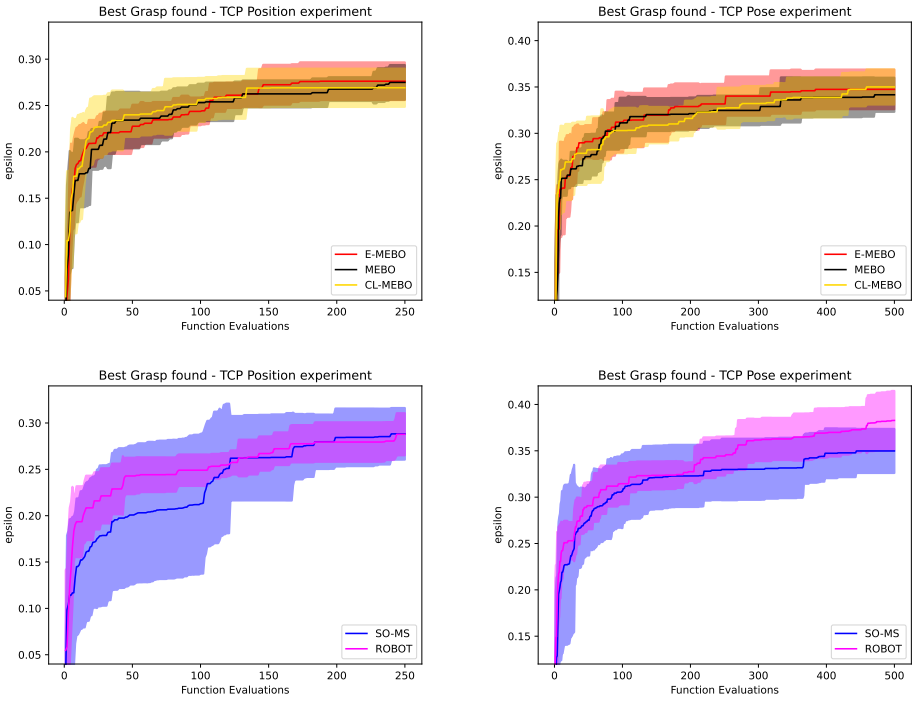


Figure 7.4: Evolution of the best grasp quality found in vase object in TCP position (Left) and TCP pose (right) optimization. Top row: MEBO, E-MEBO, CL-MEBO. Bottom row: SO-MS, ROBOT.

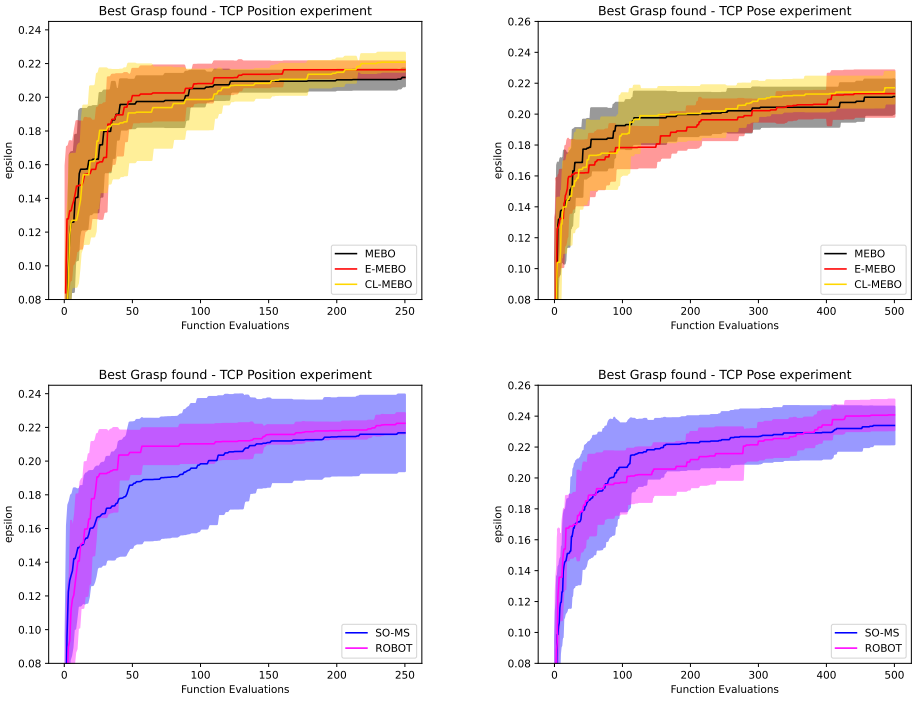


Figure 7.5: Evolution of the best grasp quality found in drill object in TCP position (Left) and TCP pose (right) optimization. Top row: MEBO, E-MEBO, CL-MEBO. Bottom row: SO-MS, ROBOT.

7.4.2 Grasp solution set

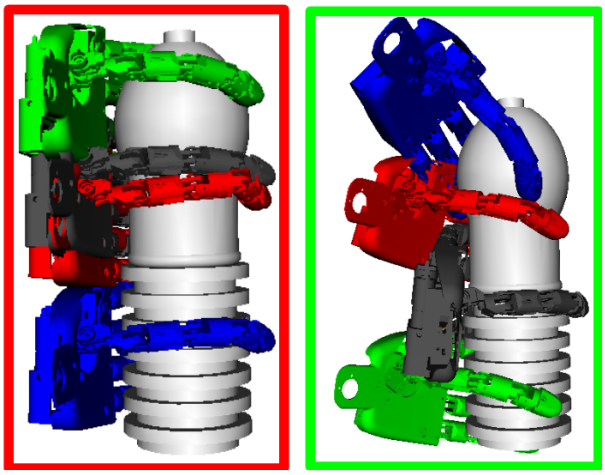
In this analysis we compare the set of M grasps returned as solution by the optimizers CL-MEBO, SO-MS and ROBOT. Table 7.9 shows the average number of diverse solutions per set (M_d) and the mean quality of the set (MGOs), in addition to the standard deviation.

From the results it can be seen that the worst by far is SO-MS, which is unable to find diverse sets and in most cases collapses into one or two solutions for all objects. This explains its high MGOs, since it collapses to high metric grasps and there is no diversity in the set, then the grasps have practically the same quality. On the other hand, CL-MEBO and ROBOT obtain diverse sets in most cases, although at the cost of significantly reducing the grasp quality compared to the optimal ones seen in the previous section. Comparing ROBOT and CL-MEBO, there is a clear difference between MGOs, the former being higher in all cases.

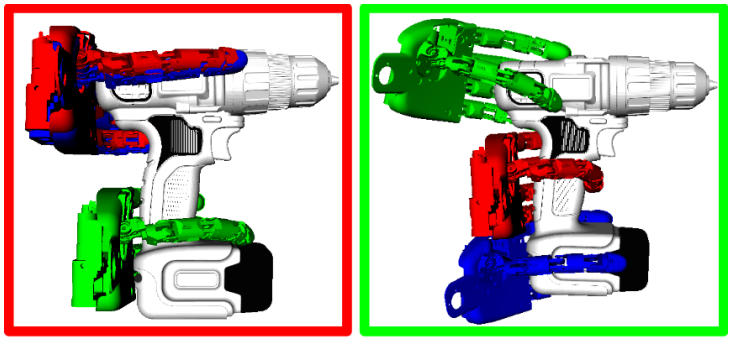
Finally, Figures [7.6, 7.7, 7.8] show an example of a bad and good diverse set across all objects for optimizers CL-MEBO, SO-MS and ROBOT, respectively. In these figures we can observe what has been discussed above. In the case of SO-MS, solution sets collapse into one or two grasps, even in "good" sets. On the other hand, in both ROBOT and CL-MEBO, the grasps are well distributed along the object. This difference is most noticeable in the bottle object.

		Bottle		Vase		Drill	
		M_d	MGOs	M_d	MGOs	M_d	MGOs
TCP position	CL-MEBO	**3.70 ±0.45**	0.646 ±0.038	**4.00 ±0.00**	0.198 ±0.013	2.60 ±0.48	0.199 ±0.012
	SO-MS	1.20 ±0.40	**0.808 ±0.052**	1.80 ±0.97	**0.258 ±0.031**	1.50 ±0.50	**0.204 ±0.025**
	ROBOT	2.00 ±0.00	0.729 ±0.038	**4.00 ±0.00**	0.218 ±0.014	**3.00 ±0.00**	0.200 ±0.006
TCP pose	CL-MEBO	**4.00 ±0.00**	0.525 ±0.033	3.90 ±0.30	0.285 ±0.021	**3.00 ±0.00**	0.177 ±0.018
	SO-MS	1.20 ±0.40	**0.695 ±0.050**	1.40 ±0.48	**0.326 ±0.026**	1.10 ±0.30	**0.225 ±0.012**
	ROBOT	**4.00 ±0.00**	0.652 ±0.026	**4.00 ±0.00**	0.305 ±0.014	**3.00 ±0.00**	0.212 ±0.014

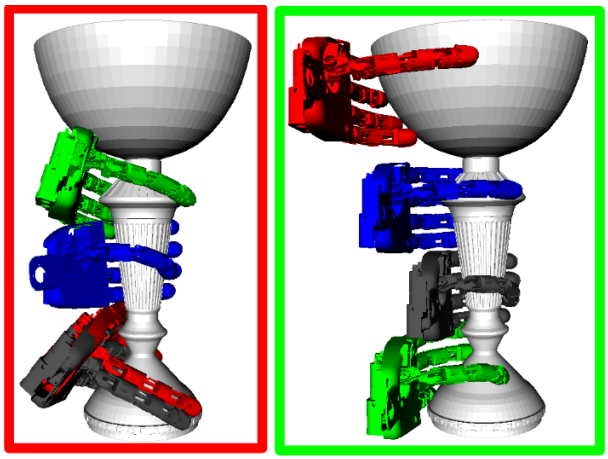
Table 7.9: Comparison of the grasp set returned as solution. M_d is the number of diverse solutions and MGOs is the average quality of the set.



(a) Bottle object. **(Left)** There are 3 diverse grasps: top, bottom and the two middle ones are the same. **(Right)** All grasps are distributed along the height of the object.

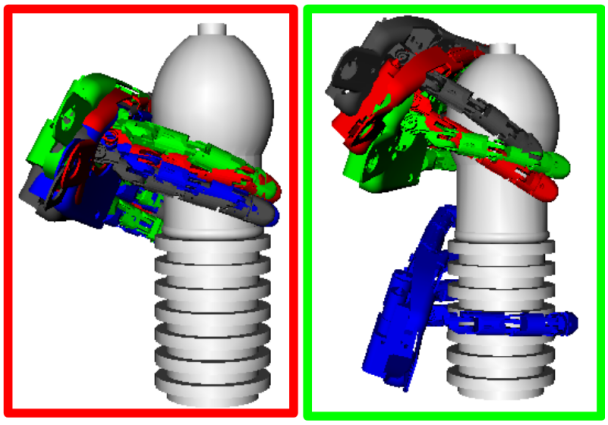


(b) Drill object. **(Left)** Red and blue grasps are the same. **(Right)** All grasps in the set are diverse.

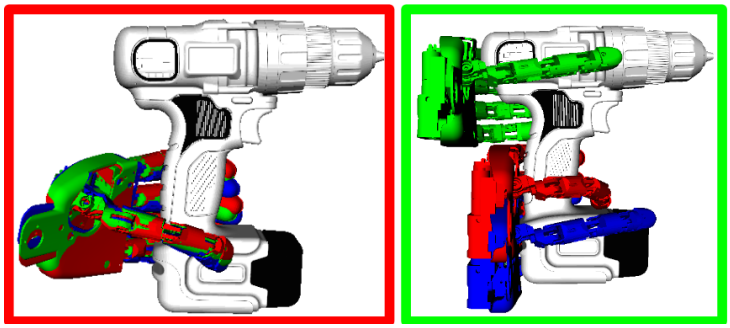


(c) Vase object. **(Left)** The two grasps at the bottom are the same. **(Right)** All grasps are located in a different grasp region.

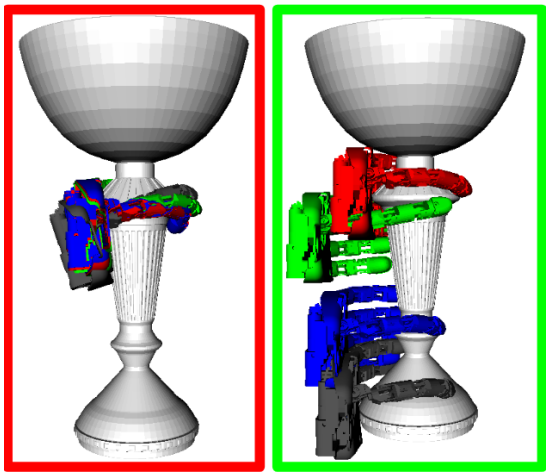
Figure 7.6: Worst (images with red border) and best (images with green border) grasp set in diversity for each object obtained with CL-MEBO.



(a) Bottle object. **(Left)** The grasp solution set collapse to one grasp. **(Right)** The three grasps at the top collapse into one, therefore there are two diverse grasps.

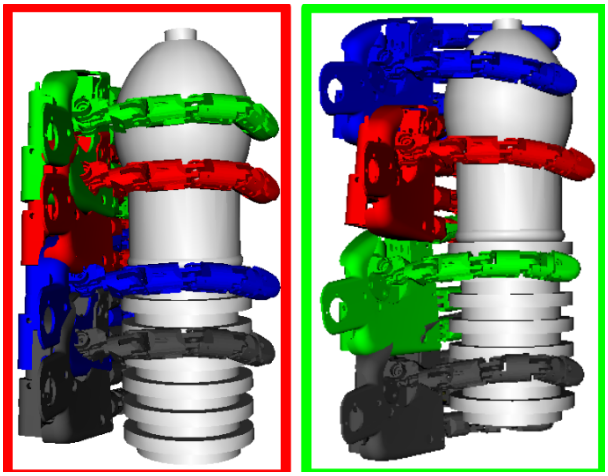


(b) Drill object. **(Left)** All grasps collapse to one. **(Right)** All the grasps in the set are diverse, although red and blue are close.

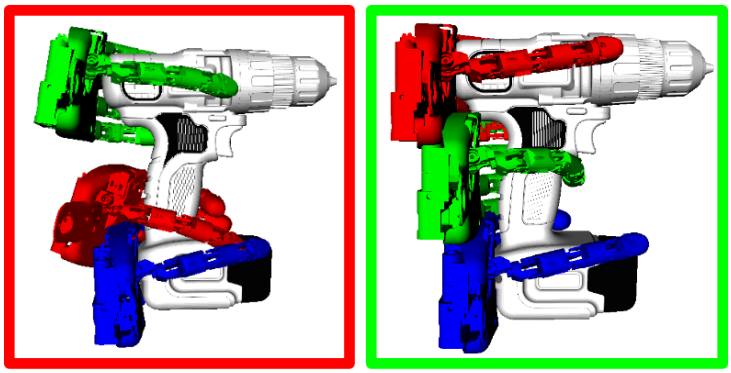


(c) Vase object. **(Left)** The grasp solution set collapse to one grasp. **(Right)** All the grasps are diverse but green and red are close.

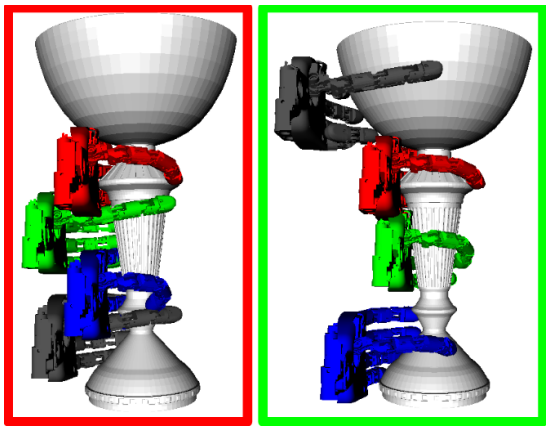
Figure 7.7: Worst (images with red border) and best (images with green border) grasp set in diversity for each object obtained with SO-MS.



(a) Bottle object. **(Left)** The red-green pair represents the same grasps and the same for the blue-black pair. **(Right)** All grasps are diverse and are well distributed.



(b) Drill object. **(Left)** The red grasp is close to the blue. **(Right)** Each grasp is located in a different region.



(c) Vase object. **(Left)** The green and red grasps are close, the same is true for the others. **(Right)** All grasps are well distributed along the object.

Figure 7.8: Worst (images with red border) and best (images with green border) grasp set in diversity for each object obtained with ROBOT.

7.4.3 Best grasp with occlusions

This section presents an analysis comparing the best grasp achieved by introducing occlusions. For example, if in the bottle we introduce an occlusion at the top and in the middle, with the results already available and taking into account those that are not occluded, what would now be the best grasp?. With this analysis we are measuring the performance of the optimizers against dynamic changes that may occur in the environment without the need to rerun the optimization. For each object a one small region will be occluded first and then a larger one, these occlusions can be seen in Figure 7.9. The occlusion regions have been chosen taking into account where the best grasps are located on each object. The analysis has been performed on the 3D and 6D experiments and can be found in Tables [7.10, 7.11], respectively.

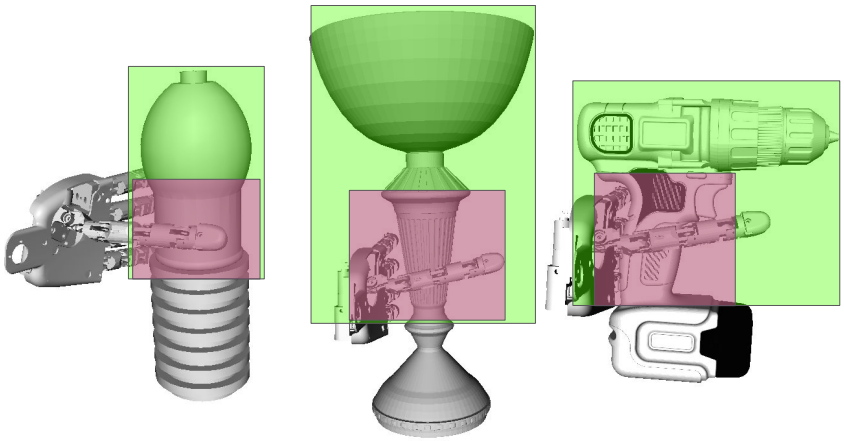


Figure 7.9: Representation of small (red) and large (green) occlusions used in each object.

		MEBO	E-MEBO	CL-MEBO	SO-MS	ROBOT
Bottle	Small	0.592 ± 0.070	0.648 ± 0.060	0.654 ± 0.043	**0.673 ± 0.081**	0.611 ± 0.050
	Large	0.589 ± 0.074	0.648 ± 0.060	0.654 ± 0.043	**0.673 ± 0.081**	0.610 ± 0.052
Vase	Small	0.256 ± 0.026	0.262 ± 0.028	0.253 ± 0.026	0.277 ± 0.030	**0.285 ± 0.026**
	Large	0.201 ± 0.015	0.195 ± 0.024	0.201 ± 0.016	0.157 ± 0.062	**0.212 ± 0.023**
Drill	Small	0.210 ± 0.006	0.208 ± 0.012	0.214 ± 0.003	0.204 ± 0.030	**0.216 ± 0.004**
	Large	0.157 ± 0.016	0.156 ± 0.019	0.157 ± 0.017	0.142 ± 0.030	**0.159 ± 0.011**

Table 7.10: Mean and standard deviation of the best grasp with occlusions in 3D experiments.

In Table 7.10 we can see how in the vase and drill objects the best grasp achieved with the large occlusion is much lower than with the small occlusion. Also that the metrics achieved by the optimizers are very similar in these two objects for the two types of occlusions and there are hardly any differences between them. It is worth highlighting the case of SO-MS, which achieves a good grasp quality on almost all objects, taking into account that in the analysis of

		MEBO	E-MEBO	CL-MEBO	SO-MS	ROBOT
Bottle	Small	0.576 ± 0.080	0.587 ± 0.059	0.618 ± 0.067	0.577 ± 0.112	**0.624 ± 0.065**
	Large	0.466 ± 0.033	0.494 ± 0.044	0.523 ± 0.042	0.445 ± 0.187	**0.596 ± 0.079**
Vase	Small	0.315 ± 0.014	0.345 ± 0.023	0.347 ± 0.014	0.336 ± 0.028	**0.361 ± 0.025**
	Large	0.312 ± 0.014	0.345 ± 0.023	0.347 ± 0.014	0.244 ± 0.138	**0.358 ± 0.029**
Drill	Small	0.194 ± 0.021	0.192 ± 0.017	**0.212 ± 0.009**	0.182 ± 0.045	**0.212 ± 0.013**
	Large	**0.163 ± 0.007**	0.157 ± 0.008	0.158 ± 0.019	0.121 ± 0.048	0.159 ± 0.021

Table 7.11: Mean and standard deviation of the best grasp with occlusions in 6D experiments.

the previous section it did not show diversity. Although, it is still the optimizer with the highest variation as indicated by its standard deviation, on the other hand MEBO-based methods have the lowest variation.

This can also be seen in Table 7.11, where we can see that ROBOT is the best performing optimizer in all cases. In these experiments SO-MS achieves poor results with large occlusion, with the vase object being the worst case. The difference between MEBO-based methods, for both 3D and 6D experiments, is also minimal, with CL-MEBO being slightly better, possibly thanks to the local refinements it performs with clustering.

Chapter 8

Conclusion and future work

8.1 Conclusions

Once the work has been completed, we can affirm that all the proposed objectives have been fulfilled. To address the problem of grasping with robotic hands, a simulation environment has been developed using Simox. This environment is flexible, allowing new types of grasp planners to be added. It has been possible to work with complex robotic hands and to see the advantages over traditional grippers, although there is still a lot of research to be done.

The problem of grasping previously unknown objects has been approached from a Bayesian optimization perspective and, moreover, with the objective of achieving multiple diverse and valid grasps. Despite the limited research in multisolution Bayesian optimization, a first base model, MEBO, has been designed and implemented, which solves the multisolution problem without the need to specify the number of desired solutions or the minimum level of diversity among them. Later, a variant, E-MEBO, was defined, which performs a more exhaustive exploration of the search space to find more possible regions of interest. In addition, based on the latter, a latest version, CL-MEBO, was implemented, which, using clustering algorithms, among many other techniques, solves the multisolution problem when the user knows the number of solutions and the level of diversity desired. Moreover, these methods have been implemented in a professional Bayesian optimization library, BayesOpt.

On the other hand, a flexible and modular experiment execution system has been implemented. It allows running experiments of different types, such as grasping or synthetic functions, and making use of different optimizers, such as SigOpt or BayesOpt.

Finally, both the MEBO-based methods and other available MBO methods (SO-MS and ROBOT) have been tested by first using a multimodal function benchmark. Here we see how CL-MEBO and SO-MS obtain very good results, closely followed by E-MEBO. ROBOT being by far the worst optimizer. However, the results change when the methods are tested on a

real problem, robot grasping. In this problem we see that the best and most diverse grasps are obtained by ROBOT for both position optimization and pose optimization, followed closely by CL-MEBO. This small difference may be due to the fact that ROBOT uses a more complex system by maintaining regions of possible solutions throughout the optimization process. Whereas CL-MEBO uses a cluster approach but the consistency of the clusters is not maintained throughout the iterations. In addition, in this problem the MEBO-based methods obtain almost equal results. On the other hand, SO-MS obtains poor results when trying to achieve diverse grasps, often collapsing the solution set to one or two grasps. Finally, it has been observed that with multisolution methods the solution grasps lose a lot of quality with respect to the optimum.

8.2 Future work

There is still a lot of work to be done. For example, the optimization of the hand posture in the end could not be done due to the lack of public data, therefore a first work could go into collecting information about the eigengrasps and optimize the grasp by making use of them. Although more interesting would be to test the results obtained on a real robot, either the iCub or another, such as the Shadow hand.

It would also be interesting to test the methods on more complex robotics problems, such as taking into account the trajectory of the robotic arm to the grasping position or using two hands to grasp objects. Moreover, manipulation tasks, such as pick and place objects, could be performed to verify that the grasps are really good.

In addition, since this work uses multisolution Bayesian optimization, which has hardly been investigated, it would be interesting to check how the implemented methods perform in other problems, such as the tuning of hyperparameters of a neural network. Finally, improvements in MEBO could be implemented to obtain better results.

Chapter 9

Bibliography

[1] Oliver Kroemer, Renaud Detry, Justus Piater, and Jan Peters. Active learning using mean shift optimization for robot grasping. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2610–2615, 2009.

[2] Luis Montesano and Manuel Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60(3):452–462, 2012. Autonomous Grasping.

[3] José Nogueira, Ruben Martinez-Cantin, Alexandre Bernardino, and Lorenzo Jamone. Unscented Bayesian optimization for safe robot grasping. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1967–1972. IEEE, 2016.

[4] João Castanheira, Pedro Vicente, Ruben Martinez-Cantin, Lorenzo Jamone, and Alexandre Bernardino. Finding safe 3D robot grasps through efficient haptic exploration with unscented Bayesian optimization and collision penalty. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1643–1648, 2018.

[5] Cristiana De Farias, Naresh Marturi, Rustam Stolkin, and Yasemin Bekiroglu. Simultaneous tactile exploration and grasp refinement for unknown objects. *IEEE Robotics and Automation Letters*, 6(2):3349–3356, 2021.

[6] Peter Englert and Marc Toussaint. Combined Optimization and Reinforcement Learning for Manipulation Skills. In *Robotics: Science and systems*, volume 2016, 2016.

[7] Ghazal Ghazaei, Iro Laina, Christian Rupprecht, Federico Tombari, Nassir Navab, and Kianoush Nazarpour. Dealing with Ambiguity in Robotic Grasping via Multiple Predictions. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part IV*, pages 38–55, 2018.

[8] Ignacio Herrera Seara, Juan García-Lechuz Sierra, Javier Garcia-Barcos, and Ruben Martinez-Cantin. Optimización bayesiana multisolución para la exploración eficiente de agarres robóticos. In *XLIII Jornadas de Automática*, pages 714–720. Universidade da Coruña. Servizo de Publicacións, 2022.

[9] Ibai Roman, Alexander Mendiburu, Roberto Santana, and Jose A. Lozano. Bayesian Optimization Approaches for Massively Multi-Modal Problems. In *Learning and Intelligent Optimization: 13th International Conference, LION 13, Chania, Crete, Greece, May 27–31, 2019, Revised Selected Papers*, page 383–397, Berlin, Heidelberg, 2019. Springer-Verlag.

[10] SigOpt. SigOpt: Multisolution Experiment. `https://docs.sigopt.com/advanced_experimentation/multi_solution`. [Accessed: 2022-11-24].

[11] Natalie Maus, Kaiwen Wu, David Eriksson, and Jacob Gardner. Discovering Many Diverse Solutions with Bayesian Optimization. *arXiv preprint arXiv:2210.10953*, 2022.

[12] Ruben Martinez-Cantin. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. *Journal of Machine Learning Research*, 15(115):3915–3919, 2014.

[13] Nikolaus Vahrenkamp, Manfred Kröhnert, Stefan Ulbrich, Tamim Asfour, Giorgio Metta, Rüdiger Dillmann, and Giulio Sandini. Simox: A Robotics Toolbox for Simulation, Motion and Grasp Planning. *Advances in Intelligent Systems and Computing*, 193, 01 2012.

[14] Philipp Schmidt, Nikolaus Vahrenkamp, Mirko Wächter, and Tamim Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6831–6838, 2018.

[15] Marco Santello, Martha Flanders, and John F Soechting. Postural hand synergies for tool use. *Journal of neuroscience*, 18(23):10105–10115, 1998.

[16] Maximo A. Roa and Raul Suarez. Grasp Quality Measures: Review and Performance. *Autonomous Robots*, 38:65–88, 07 2014.

[17] R.M. Murray, Z. Li, S.S. Sastry, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. Taylor & Francis, 1994.

[18] Byoung-Ho Kim, Sang-Rok Oh, Byung-Ju Yi, and Il Hong Suh. Optimal grasping based on non-dimensionalized performance indices. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 2, pages 949–956 vol.2, 2001.

[19] K.B. Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[20] Beatriz León, Joaquín L. Sancho-Bru, Néstor J. Jarque-Bou, Antonio Morales, and Máximo A. Roa. Evaluation of human prehension using grasp quality measures. *International Journal of Advanced Robotic Systems*, 9(4):112, 2012.

[21] E. Boivin, I. Sharf, and M. Doyon. Optimum grasp of planar and revolute objects with gripper geometry constraints. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 1, pages 326–332 Vol.1, 2004.

[22] Jacopo Aleotti and Stefano Caselli. Interactive teaching of task-oriented robot grasps. *Robotics and Autonomous Systems*, 58:539–550, 05 2010.

[23] A.T. Miller and P.K. Allen. Examples of 3D grasp quality computations. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1240–1246 vol.2, 1999.

[24] The iCub Project. The iCub Project: research-grade humanoid robot designed to help developing and testing embodied AI algorithms. `https://icub.iit.it/`. [Accessed: 2023-01-12].

[25] Lorenzo Natale, Chiara Bartolozzi, Francesco Nori, Giulio Sandini, and Giorgio Metta. iCub: history and evolution. *arXiv preprint arXiv:2105.02313*, 2021.

[26] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *ArXiv*, 2010.

[27] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[28] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[29] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[30] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian Optimization without the Lipschitz Constant. *J. Optim. Theory Appl.*, 79(1):157–181, oct 1993.

[31] Jonas Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4:347–365, 1994.

[32] Daniel James Lizotte. *Practical Bayesian Optimization*. PhD thesis, CAN, 2008. AAINR46365.

[33] Michael JD Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 26, 2009.

[34] H. J. Kushner. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86(1):97–106, 03 1964.

[35] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

[36] Peter Auer. Using Confidence Bounds for Exploitation-Exploration Trade-Offs. *J. Mach. Learn. Res.*, 3(null):397–422, mar 2003.

[37] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv*, 2009.

[38] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 12 1933.

[39] José Miguel Hernández-Lobato, Michael Gelbart, Matthew Hoffman, Ryan Adams, and Zoubin Ghahramani. Predictive entropy search for Bayesian optimization with unknown constraints. In *International conference on machine learning*, pages 1699–1707. PMLR, 2015.

[40] Salomon Bochner, Monotonic Functions, Stieltjes Integrals, Harmonic Analysis, Morris Tenenbaum, and Harry Pollard. *Lectures on Fourier Integrals. (AM-42)*. Princeton University Press, 1959.

[41] Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[42] Miguel Lázaro-Gredilla, Joaquin Quiñnero-Candela, Carl Edward Rasmussen, and Anibal R. Figueiras-Vidal. Sparse Spectrum Gaussian Process Regression. *Journal of Machine Learning Research*, 11(63):1865–1881, 2010.

[43] Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44:509–534, 2009.

[44] Philipp Hennig and Christian J. Schuler. Entropy Search for Information-Efficient Global Optimization. *J. Mach. Learn. Res.*, 13(null):1809–1837, jun 2012.

[45] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[46] Swagatam Das, Sayan Maity, Bo-Yang Qu, and P.N. Suganthan. Real-parameter evolutionary multimodal optimization — a survey of the state-of-the-art. *Swarm and Evolutionary Computation*, 1(2):71–88, 2011.

[47] Ka-Chun Wong. Evolutionary multimodal optimization: A short survey. In *Advances in Evolutionary Algorithms Research*, pages 1–15. Nova Science Publishers, Inc., 2015.

[48] Xiaodong Li, Andries Engelbrecht, and Michael G Epitropakis. Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization. *RMIT University, Evolutionary Computation and Machine Learning Group, Australia, Tech. Rep*, 2013.

[49] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A Multi-points Criterion for Deterministic Parallel Global Optimization based on Kriging. 03 2008.

[50] Javier González, Zhenwen Dai, Philipp Hennig, and Neil D. Lawrence. Batch Bayesian Optimization via Local Penalization, 2015.

[51] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005.

[52] SigOpt. SigOpt: A model development platform that makes it easy to track runs, visualize training, and scale hyperparameter optimization. `https://sigopt.com/`. [Accessed: 2022-11-24].

[53] SigOpt. SigOpt: All Constraints Experiment. `https://docs.sigopt.com/advanced_experimentation/all_constraints`. [Accessed: 2023-01-09].

[54] Gustavo Malkomes, Bolong Cheng, Eric H Lee, and Mike Mccourt. Beyond the Pareto Efficient Frontier: Constraint Active Search for Multiobjective Experimental Design. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7423–7434. PMLR, 18–24 Jul 2021.

[55] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

[56] Ignacio Herrera Seara. BayesOpt: Multisolution Bayesian optimization fork. `https://github.com/nachoh8/bayesopt_multisolution`, 2023. [Accessed: 2023-01-24].

[57] Javier Garcia-Barcos and Ruben Martinez-Cantin. Fully Distributed Bayesian Optimization with Stochastic Policies. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2357–2363. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[58] Javier Garcia-Barcos and Ruben Martinez-Cantin. Robust policy search for robot navigation. *IEEE Robotics and Automation Letters*, 6(2):2389–2396, 2021.

[59] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[60] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml*, volume 1, pages 727–734, 2000.

[61] The GPyOpt authors. GPyOpt: A Bayesian Optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016.

[62] Andrei Novikov. Pyclustering: Data mining library. *Journal of Open Source Software*, 4(36):1230, apr 2019.

[63] Ignacio Herrera Seara. Multisolution Active Grasping repository. `https://github.com/nachoh8/multisolution-active-grasping.git`, 2022. [Accessed: 2023-01-09].

[64] Natalie Maus. Official implementation of ROBOT method. `https://github.com/nataliemaus/robot.git`, 2022. [Accessed: 2023-01-09].

[65] H2T group Karlsruhe Institute of Technology. Official implementation of Simox. `https://git.h2t.iar.kit.edu/sw/simox/simox`, 2022. [Accessed: 2023-01-12].

[66] Matei T Ciocarlie and Peter K Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, 2009.

[67] Giuseppe Cotugno, Vishawanathan Mohan, Kaspar Althoefer, and Thrishantha Nanayakkara. Simplifying grasping complexity through generalization of kinaesthetically learned synergies. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5345–5351, 2014.

[68] Alexandre Bernardino, Marco Henriques, Norman Hendrich, and Jianwei Zhang. Precision grasp synergies for dexterous robotic hands. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 62–67. IEEE, 2013.

Appendices

Appendix A

Postural grasp synergies

Postural grasp synergies are correlated configurations of the hand joints that typically occur during grasping tasks. For instance the synchronous contraction of the hand muscles so that finger and palm contacts with the object occur simultaneously, during which the joints adopt highly correlated postures.

This was studied in [15], where they collected data about human grasps and concluded, using principal component analysis (PCA), that the first two principal components (PCs) explain more than 80% of the variance. The application of synergies in robotics has also been extensively studied: some by mapping human grasp data to robotic models [66]; and others by collecting grasp data directly from robotic hands [67, 68]. It is common in these papers to refer to PCs as *eigengrasps*. Figure A.1 shows an example of eigengrasps.

Figure A.1: Grasp poses corresponding to the origin of eight eigengrasps on the iCub hand.

Formally, in a robotic hand setting, the eigengrasps form a low-dimensionality basis for grasp postures, and can be linearly combined to closely approximate most common grasp postures. Therefore, we can choose a basis comprising b eigengrasps $e_i \in \mathbb{R}^d$, with $b \ll d$, such that the hand posture placed in the subspace defined by this basis can be expressed as a function

of the amplitudes α_i along each eigengrasp direction:

$$p = \sum_{i=1}^{b} \alpha_i e_i, \ \alpha_i \in \mathbb{R}$$

Therefore, a hand posture $p \in \mathbb{R}^d$ can be completely defined by the amplitude vector $\alpha = [\alpha_1 \ \alpha_2 \ ... \ \alpha_b]$, and consequently the optimization problem will be defined as:

$$w^*, \alpha^* = \arg \max_{w \in W, \ \alpha \in \mathbb{R}^b} \mathcal{Q}(w, \alpha, r, o).$$

Appendix B

Bayesian optimization as a partially observable Markov decision process

We can consider BO as a partially observable Markov decision process (POMDP). To understand what a POMDP is, it is first necessary to introduce some concepts:

- **Agent:** entity which will be trained to make correct decisions.
- **Environment:** is the surrounding with which the agent interacts, but the agent cannot manipulate it.
- **State:** defines the current situation of the agent.
- **Action:** the choice that the agent makes at the current state.
- **Policy:** is the thought process behind the choice of an action. In practice, it is a probability distribution assigned to the set of actions.

Then a Markov decision process (MDP) is a stochastic model where the agent and the environment are fully represented by state variables $s_t \in \mathcal{S}$ and the agent can change that state by performing actions $a_t \in \mathcal{A}$ in a Markovian way $p(s_{t+1}|a_t, s_t)$ i.e. the current state depends only on the immediate previous state. The MDP model also assumes that the agent is rational, acting to maximize the future expected reward $\mathbb{E}[\sum_{t=0}^{N} R(s_t, a_t)]$. The behavior of the agent is encoded in the policy which maps states to actions $a_t = \pi(s_t)$. In order to rank the possible actions at one state, we can compute the Q-*function* which represents the quality of taking a certain action considering the future reward. The optimal Q-*function*, Q^*, can be computed by doing full backups of future rewards and actions recursively. Then, we can obtain the optimal greedy function by:

$$\pi^*(s_t) = \arg\max_a Q^*(s_t, a_t)$$

In a POMDP the agent does not have full observability of the state and must rely on partial observations y_t following an observation model $p(y_t|s_t, a_t)$. So, the agent relies on beliefs, which are the distributions over possible states, given the known observations and actions $b_t = p(s_t|a_{0:t}, y_{0:t})$.

Finally, we can connect BO to POMDP, considering the optimization algorithm as an agent that is interacting with the space of functions (environment). The current state of the environment is the target function f. The agent does not have full observability of the state and can only perform partial observations by querying the function $y_t = f(x_t)$. Where the next point to evaluate x_t is obtained maximizing the acquisition function α (*Q-function*) on the surrogate model (belief). Table B.1 summarizes the relation between BO and POMDP.

POMDP / belief MDP	BO
State: s_t	Target function: f
Action: a_t	Next query: x_{t+1}
Observation: y_t	Response value: $y_t = f(x_t)$
Belief: $b_t = p(s_t)$	Surrogate model: $p(f)$
Q-function: $Q^*(b_t, a_t)$	Acquisition function: $\alpha(x, p(f))$
Reward: $R(s_t, a_t)$	Improvement (EI): $\max(0, y_{t+1} - \rho_t)$

Table B.1: Comparison of POMDP and belief MPD terms with respect to the corresponding elements in BO

Appendix C

Cluster MEBO algorithms

The following algorithm in pseudocode summarizes the CL-MEBO optimization process:

```
def CL_MEBO(I, N, B, M):
    # I: number of initial points,
    # N: total number of function evaluations,
    # B: batch size, M: number of solutions,
    D = latin_hypercube_sampling(I)
    init_surrogate_model(D)
    it = 0 # number of iterations
    while |D| < N:
        if |D| < N*0.4:
            batch = explorative_sampling(B)
        elif |D| < N*0.5:
            batch = boltzmann_selection(B)
        elif |D| < N*0.7:
            if it % 2 == 0:
                batch = boltzmann_selection(B)
            else:
                batch = cluster_batch_sampling(M, B)
        elif |D| < N*0.9:
            if it % 3 == 0:
                batch = boltzmann_selection(B)
            else:
                batch = cluster_batch_sampling(M, B)
        else:
            batch = cluster_batch_sampling(M, B)
        res = f(batch) # evaluate batch
        D = D + res # update dataset
        update_surrogate_model(D)
        it += 1

    solution_set = select_solution(D, M)
    return solution_set
```

The following algorithm in pseudocode summarizes the *Cluster correction* step of the *Cluster batch sampling*:

```
def CLUSTER_CORRECTION(clusters, divl):
    # clusters: set of <centroid, points> pairs,
    # divl: minimum level of diversity
    # 1. COMPUTE (NO) VALID CLUSTERS
    valid_clusters = {} # set of clusters
    no_valid_clusters = {} # set of set of clusters
    for cl in clusters:
        nvc = {cl} # cluster set with same diversity
        for cl2 in clusters - {cl}:
            if d(cl.centroid, cl2.centroid) < divl:
                nvc += cl2
        if |nvc| > 1:
            no_valid_clusters += nvc
        else:
            valid_clusters += cl
    # 2. MERGE NO VALID CLUSTERS
    for cl in clusters - valid_clusters:
        # get the no valid set with the max size
        merge_nvc_idx =
            argmax(|nvc| for nvc in no_valid_clusters
                    if cl in nvc)
        merge_nvc = no_valid_clusters[merge_nvc_idx]
        if merge_nvc not in valid_clusters:
            # merge set
            new_points = [] # new list of points
            for c in merge_nvc:
                new_points += c.points
            # compute the mean centroid
            new_centroid=mean([c.centroid for c in merge_nvc])
            valid_clusters += <new_centroid, new_points>
    return valid_clusters
```

Appendix D

Batch Bayesian optimization via Local penalization

Batch Bayesian optimization via Local penalization (BBO-LP) [50] is implemented in the GPyOpt library [61]. In this work we implemented this method in the BayesOpt library. However, this implementation does not work correctly and it has not been possible to find out where the errors are coming from. The error lies in the fact that the Lipschitz constant is not calculated correctly in BayesOpt, which causes the batch points to be almost equal. Table D.1 shows the big difference between the two implementations when obtaining the batches. Both implementations use the same parameters: UCB with $\beta = 2$ as acquisition function, a batch size of 3 and 150 function evaluations, with 15 initial points sampled with Latin Hypercube sampling. Each experiment was run a total of 10 times.

	Branin		SixHumpCamel		Beale	
	x	y	x	y	x	y
BayesOpt	0.0000	0.0000	0.0004	0.0002	0.0075	0.003
GPyOpt	24.8935	26.327	6.6896	2.9759	9.9616	9.7694

Table D.1: Mean of the batch variance per dimension in 2D functions with BBO-LP in BayesOpt and GPyOpt libraries.

In addition, the original implementation was considered to be used as a method to make comparisons, but due to the time cost of run an experiment (>2h with 250 function evaluations) its use was discarded. Some results achieved in the CEC13 benchmark [48] can be seen in Table D.4, the metrics and functions of this benchmark are explained in sections 6.2 and 7.1. Each experiment was run 4 times with the same parameters as in the previous experiments, but with 250 function evaluations and 25 initial points. It can be seen that the results are not bad.

Uneven Maxima				Five-Uneven-Peak Trap				Equal Maxima			
PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
0.7000	0.7000	0.8828	1.0000	1.0000	1.0000	0.1080	200.0000	1.0000	1.0000	0.0844	1.0000

Table D.2: 1D Functions.

Himmelblau				Modified Rastrigin			
PR ↑	SR ↑	CS ↓	MGOs ↑	PR ↑	SR ↑	CS ↓	MGOs ↑
0.9250	0.7000	0.8984	199.9882	0.4333	0.0000	1.0000	-2.0001

Table D.3: 2D Functions.

Table D.4: Results obtained with BBO-LP in the CEC13 benchmark.

List of Figures

List of Tables