Bayesian Optimization for MPPI Control of Robot Arm Planar Pushing

Yulun Zhuang
Robotics
University of Michigan
Ann Arbor, United States
yulunz@umich.edu

Ziqi Han
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, United States
ziqihan@umich.edu

***Abstract*—In this project, we focus on optimizing the hyperparameters of Model Predictive Path Integral (MPPI) control for a pushing task with non-trivial obstacles using Bayesian Optimization Algorithm (BOA). We implement the BOA with GPytorch library, a Gaussian Process (GP) library, to balance exploration and exploitation effectively in the parameter space. Our approach enhances the performance of MPPI in planar pushing task by iteratively fitting a GP to known samples and utilizing acquisition functions like Upper Confidence Bound (UCB) and Expected Improvement (EI). We compare our optimization results against three baselines: manually-tuned parameters and parameters tuned by Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method and another Bayesian Optimization library implemented in *scikit-learn*. Our results demonstrate the effectiveness of Bayesian Optimization in improving MPPI performance, contributing to the research on practical applications of GPs in robotics and control systems.**

***Index Terms*—Bayesian Optimization, Gaussian Process, GPyTorch, Hyperparameter Optimization**

I. Introduction

In this project, we delve into the investigation of Gaussian Processes (GPs) for Bayesian Optimization [1] within the context of optimizing the parameters of Model Predictive Path Integral (MPPI) control, specifically for a pushing task in a bullet pushing environment. The intricate pushing task involves navigating through a non-trivial set of obstacles, building upon the foundations established in previous research. By leveraging a GP in Bayesian Optimization, our objective is to uncover an optimal set of parameters for MPPI that can efficiently execute a variety of pushing tasks.

Our methodology assumes that the initial state is known, and the environment can be reset after each run during the optimization of the parameters. We meticulously account for the distinct scales and constraints on the parameters, such as ensuring a positive noise variance. We implement Bayesian Optimization using the GPytorch library [2] for our GP, and assess the performance of MPPI with our optimized parameters against two baseline comparisons. The first baseline employs manually-defined parameters, while the second baseline adopts a distinct optimization method, such as a black-box method like Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [3].

We exhibit the efficacy of GPs for Bayesian Optimization in augmenting the performance of MPPI for complex pushing tasks through a systematic, iterative process that strikes a balance between exploration and exploitation. By fitting a GP to known samples and harnessing acquisition functions like Upper Confidence Bound (UCB) [4] or Expected Improvement (EI) [5], our approach progressively refines the regions in parameter space worth exploring. This enables us to minimize the number of steps required to pinpoint a combination of parameters in close proximity to the optimal combination, rendering Bayesian Optimization suitable for scenarios where sampling the function to be optimized is computationally demanding.

Through the execution of this research, we aspire to contribute to the burgeoning body of knowledge on the practical applications of GPs in the realm of robotics and control systems [6]. Our results underscore the potential of Bayesian Optimization in enhancing MPPI performance within complex environments, providing valuable insights for future investigations and real-world applications in robotics and control systems.

II. Related Work

The traditional way of performing hyperparameter optimization has been grid search, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm [7]. However, both grid search and random search share the following common limitations: computational inefficiency and suboptimal performance.

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [3] is a state-of-the-art optimizer for continuous black-box functions. It can effectively explore high-dimensional solution spaces. By maintaining and updating a multivariate normal distribution, the algorithm iteratively

This is the final project for course ROB 498: Robot Learning for Planning and Control. The open-source code of the project can be found in https://github.com/silvery107/bayesian-opt-gpytorch

generates and evaluates solution vectors until an optimal solution is identified. Notable for its adaptive step length adjustment, efficient high-dimensional exploration, and robust global search capabilities, CMA-ES has been extensively applied in areas such as parameter optimization, machine learning, and neural networks.

III. Implementation

A. Bayesian Optimization

Bayesian optimization provides an elegant method for addressing the black-box optimization problem by utilizing information from previous search points to determine the next search point [8]. This approach has been demonstrated to outperform other global optimization algorithms in terms of efficiency and effectiveness [9].

Bayesian optimization operates by constructing a posterior distribution of functions, typically a Gaussian process, that best approximates the function to be optimized. As the number of observations increases, the posterior distribution improves, allowing the algorithm to identify regions in the parameter space that are more promising for exploration and those that are less relevant, as illustrated in Figure 1.

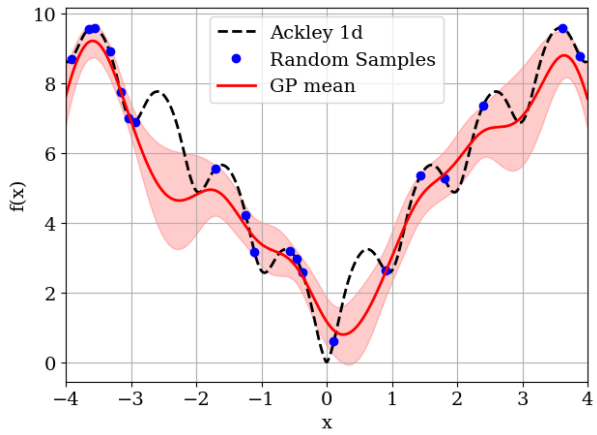


Fig. 1: Ackley's function at $y = 0$, GP fit with random samples

The algorithm iteratively balances exploration and exploitation based on its current understanding of the target function. At each step, a GP model [10] is fitted to the known samples (previously explored points), and the posterior distribution, combined with an exploration strategy such as Upper Confidence Bound (UCB) or Expected Improvement (EI), is employed to determine the next point for exploration. This process aims to minimize the number of steps required to find a combination of parameters that closely approximates the optimal combination. To achieve this, the method uses a proxy optimization problem (maximizing the acquisition function), which is computationally cheaper and can be solved using common optimization tools. Consequently, Bayesian Optimization is particularly well-suited for scenarios where sampling the function to be optimized is computationally expensive.

The pseudocode of the Bayesian Optimization Algorithm (BOA) is presented in Algorithm 1 [11]. In this algorithm, f represents the objective function of parameters $\mathbf{x}$ to be optimized, $\mathcal{X}$ denotes the parameter boundary, S is the acquisition function, and $\mathcal{M}$ signifies the Gaussian Process model. We implemented the BOA based on the GPyTorch library [2], a Gaussian process library built on PyTorch. GPyTorch enables straightforward implementation of popular scalable GP techniques and often significantly improves GPU utilization compared to solvers based on the Cholesky decomposition. Additionally, GPyTorch easily integrates with deep learning frameworks.

Algorithm 1 Bayesian Optimization Algorithm

Input: $f, \mathcal{X}, S, \mathcal{M}$
  $\mathcal{D} \leftarrow InitSamples(f, \mathcal{X})$
  for $i \in \mathcal{D}$ **do**
    $p(y|\mathbf{x}, \mathcal{D}) \leftarrow FitModel(\mathcal{M}, \mathcal{D})$
    $x_i \leftarrow \text{argmax}_{\mathbf{x}\in\mathcal{X}} \text{S}(\mathbf{x}, \text{p}(\text{y}|\mathbf{x}, \mathcal{D}))$
    $y_i \leftarrow f(\mathbf{x}_i)$ ▷ Expensive step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
  end for

B. Hyperparameters Optimization for MPPI Controller

To accomplish the planar pushing task, we leverage a flexible model predictive control (MPC) method. The MPPI control, a sampling-based algorithm capable of optimizing general cost criteria [12] [13]. This method has been demonstrated to be applicable to a broad range of stochastic systems, particularly those with dynamics represented by a neural network [14], enabling a purely data-driven approach to model learning within the MPPI framework.

The pseudocode of MPPI is shown in Algorithm 2 [14], where $\mathbf{F}$ represents the transition dynamics model, K is the number of samples, T is the number of timesteps, and $\mathbf{u}$ is the initial control sequence. The predefined running cost function and terminal cost function are represented by q and Φ, respectively. The control hyperparameters Σ and λ are optimized using the BOA. In the context of the MPPI algorithm, Σ is the covariance matrix of the Gaussian noise added to the control sequences during the sampling process, and λ is a temperature parameter that affects the exploration-exploitation trade-off in the optimization of control sequences. A suitable choice of these hyperparameters is crucial for the performance of the MPPI algorithm, and the Bayesian optimization serves as an efficient method to fine-tune them.

By optimizing the hyperparameters Σ and λ using Bayesian optimization, we can effectively enhance the performance of the MPPI algorithm in various tasks. The optimization process entails searching for an optimal set of hyperparameters that minimize the cost function while maintaining a balance between exploration and exploitation. This balance is critical, as overly aggressive exploration may lead to instability and inefficient control, while excessive exploitation could result in

Algorithm 2 Model Predictive Path Integral Control

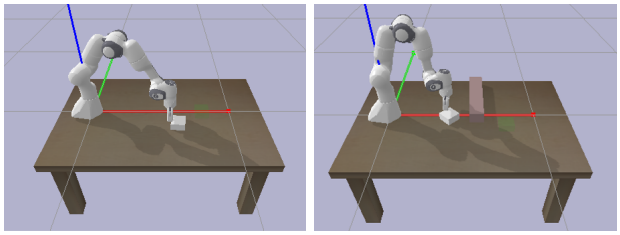
Input: $\mathbf{F}, K, T, (\mathbf{u}_0, \ldots, \mathbf{u}_{T-1}), \Sigma, \lambda$
Output: $\mathbf{u}_0$

```
D ← WarmupSamples(f, X)
while task not completed do
    x_0 ← GetStateEstimate()
    for k ← 0 to K − 1 do
        x ← x_0
        Sample E^k = {ε^k_0, ..., ε^k_{T−1}}
        for t ← 1 to T do
            x_i ← F(x_{t−1}, u_{t−1} + ε^k_{t−1})
            S(E^k) += q(x_i + λ u^T_{t−1} Σ^{−1} ε^k_{t−1})
        end for
        S(E^k) += Φ(x_T)
    end for
    β ← min_k S(E^k)
    η ← Σ_{k=0}^{K−1} exp(−(S(E^k) − β)/λ)
    for k ← 0 to K − 1 do
        w(E^k) ← exp(−(S(E^k) − β)/λ)/η
    end for
    for t ← 1 to T − 1 do
        u_t += Σ_{k=1}^{K} w(E^k) ε^k_t
    end for
    SendToActuators(u_0)
    for t ← 1 to T − 1 do
        u_{t−1} ← u_t
    end for
    u_{t−1} ← Initialize(u_{T−1})
end while
```

suboptimal performance due to inadequate exploration of the control space.

IV. Experiments and Results

In the experimental section of this project, we employed the panda pushing environment as our testbed, wherein a robotic arm is utilized to push an object to a designated location while navigating through obstacles, as shown in Figure 2. With respect to the controller, we use the pretrained multi-step residual dynamics model.



(a) Free pushing scene (b) Obstacle pushing scene

Fig. 2: Box pushing task under free and obstacle environment

A. Implementation Details

We test the correctness of our BOA implementation and success in finding the minimum objective of a 1D Ackley function (Figure 1) and then a Dd Ackley function.We decouple the optimization loop and implement it in a easy-to-used "ask-tell" interaction pattern, and give the control of evaluating objective function back to task, since for many tasks, the objective value can only be obtained at the end of expensive execution and our BOA should function as a plugin for them.

$$k\left(x_i, x_j\right) = \exp\left(-\frac{d\left(x_i, x_j\right)^2}{2l^2}\right) \tag{1}$$

We use Martern kernel for our GP model as it is a extension of the RBF kernel (Equation 1) and has a better generalization capability. It has an additional parameter *v* which controls the smoothness of the resulting function, and we choose *v* to be 2.5 indicates a twice differentiable function. In each optimization iteration, the GP model is retrained based on updated dataset for 50 epochs using $Adam$ optimizer with 0.1 learning rate and 1e-5 weight decay.

TABLE I: Comparasion of two types of acquisition functions

Aqusition	Ackley 1D		Ackley 2D	
	Min Objective	Time	Min Objective	Time
TS	**0.0003**	6.49	0.3068	6.56
EI	0.1236	**4.51**	**0.0366**	**4.51**

Two type of acquisition functions are implemented, the Thompson Sampling (TS) method and the Expected Improvement (EI) method. In our experiment with the Ackley function, the Thompson Sampling method usually takes twice the time for giving the next parameter suggestion due to a poor serial implementation of sampling from Normal distribution in PyTorch. Table I shows that even though the Expected Improvement method has a faster converge on lower dimensional problem, the diversity of sampling in the Thompson Sampling method can lead to a stable optimization result on higher dimensional problem.

The final optimizer for the planar pushing task is implemented in a "study-trial" design with auto logging and saving functions. We treat each experiment as a study with a set of hyper-hyperparameters (e.g. whether to include obstacle, random target state, scale in the cost, etc.) and the hyperparameter of MPPI controller is optimized through each trial (execute pushing task once) within a study. Since we are comparing our optimization results with other black-box optimization method and another Bayesian Optimization implementation in *scikit-learn*, we unified their interface in a "ask-tell" style and also keep input and output data types and structures consistent, so that they will have the same interface within our optimization study and lead to a clean and easy-to-use experiment script.

B. Experimental Setup

Throughout the experiments, we used manually adjusted parameters, parameters optimized by the CMA-ES algorithm, and parameters optimized by a widely used Bayesian algorithm(called Bayesian Reference) [15] as baselines to compare with our Bayesian optimization algorithm, conducting multiple comparative experiments to comprehensively validate the effectiveness of our algorithm.

The experimental component consists of two main parts: hyperparameter training and performance validation.

In the training phase, we devised two distinct cost functions. The first one involves setting a larger horizon for the MPPI controller, defining the starting and ending points, and allowing the MPPI to plan multiple paths from the starting point to the endpoint. Subsequently, the cost of each trajectory is computed and summed to obtain the cost function. This optimization method does not require physically moving the object, resulting in faster computations. However, since the residual model we employ cannot guarantee that the object is always pushed to the ideal position, significant positional deviations may occur after several steps, rendering this cost unsatisfactory in reflecting the actual performance in real-world tasks. Consequently, we adopted the following cost function, training hyperparameters in real planar pushing experiments.

We first established an overall cost function to evaluate the efficacy of the robotic arm in pushing the object to the target location. We set the maximum number of pushes in each epoch to 20, with more than 20 pushes deemed as task failure. If the deviation between the object and the target is less than 0.1 within 20 pushes, the task is considered successful. We designed a heuristic cost function f_{cost} based on several factors, including the distance from the object to the target d_{goal} , the number of pushes by the robotic arm N, and task failure P:

$$f_{\text{cost}} = d_{\text{goal}} + \alpha \cdot N + (1 - \beta) \cdot P \tag{2}$$

Here, α is a hyperparameter used to balance the cost magnitude of each component, β is set to 0 when the task is successful, otherwise it is set to 0. We adjusted the parameters based on the overall cost function to minimize the cost as much as possible.

On this foundation, we established two training tasks. The first task involves no obstacles, with the robotic arm pushing the object to reach different locations. Within the accessible range of the experimental environment, we randomly set target points, changing them for each epoch. The second task entails fixed obstacles, with the robotic arm pushing the object around the obstacles to reach a fixed endpoint. These two scenarios comprehensively cover all possible situations the robotic arm might encounter while pushing objects.

C. Results

In the training process, we set epochs (times the planar pushing task will be run) to 10, 50, and 500, respectively, resulting in multiple sets of hyperparameters. Subsequently, we conducted 100 tests for each set of hyperparameters, recording the number of successful trials, the average number of steps taken, and the average cost. The results for the configuration with 10-epochs are presented in Table II. In the table, "Free Pushing" refers to an experimental environment with no obstacles and random endpoints, while "Obstacle Pushing" denotes a fixed-obstacle, fixed-endpoint experimental setting.

A 10-epochs training process is relatively short, and it is evident that both the CMA-ES and Bayesian algorithms fail to achieve satisfactory results, with their performance in the two experimental environments unable to match that of hand-tuned parameters. However, our proposed algorithm clearly outperforms both the CMA-ES and Bayesian Reference methods, enabling a faster discovery of more favorable hyperparameters. For tasks with long durations, achieving better optimization results in a shorter time frame is of particular importance, and in this regard, our algorithm stands at the forefront.

TABLE II: Optimization results of 10 epochs

Algo	Free Pushing			Obstacle Pushing		
	Cost	Step	Goal	Cost	Step	Goal
BOA	0.79	5.18	98%	4.75	9.53	66%
BOA-REF	1.86	5.05	88%	4.76	9.56	66%
CMA-ES	0.89	5.04	97%	5.11	9.61	63%
Hand Tuned	**0.78**	**4.99**	**98%**	**4.12**	**9.53**	**72%**

We increased the number of training epochs to 50, and the results are shown in Table III. It can be observed that, in the free pushing task, both CMA-ES and Bayesian Reference exhibit significant improvement compared to 10-epochs, whereas the performance of our algorithm declines. The efficacy of CMA-ES essentially surpasses that of hand-tuned parameters. In the obstacle pushing task, our algorithm demonstrates substantial enhancement compared to 10-epochs, and achieves a considerable leading advantage in terms of average loss, average steps, and success rate, outperforming other algorithms.

TABLE III: Optimization results of 50 epochs

Algo	Free Pushing			Obstacle Pushing		
	Cost	Step	Goal	Cost	Step	Goal
BOA	0.90	5.12	97%	**3.02**	**8.34**	**81%**
BOA-REF	0.77	**4.92**	98%	3.72	8.92	75%
CMA-ES	**0.68**	5.10	**99%**	4.76	9.56	66%
Hand Tuned	0.78	4.99	98%	4.12	9.53	72%

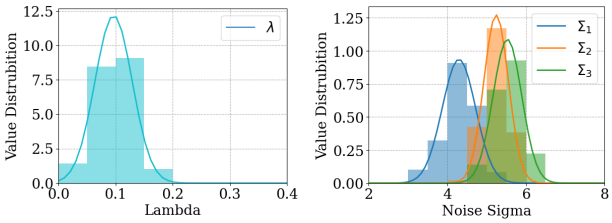
During the 50-epochs training process in the free pushing task, our algorithm exhibited an anomalous decline in performance. We postulate that this is due to the substantial randomness in the process of the robotic arm pushing the planar, and that the number of training steps was inadequate to achieve a stable optimal solution. Therefore, we designed an experiment with 500 epochs, as presented in Table IV. In this case, it is evident that our algorithm surpassed the hand-tuned parameters in both tasks, and in comparison with two other algorithms, demonstrated a considerable advantage in terms of the loss function and accuracy, even achieving an impressive 86% success rate in the task of pushing the planar while avoiding obstacles. It is particularly worth noting that the aforementioned anomalous decline in performance has been mitigated, resulting in the attainment of the anticipated outcomes.

Even through each optimization runs result in different set of hyperparameters, they still share a kind of convergence. The distribution of them are draw in Figure 3. The λ is optimized within $[0, 1]$ and converged to about 0.1 while the Σ, as is

TABLE IV: Optimization results of 500 epochs

Algo	Free Pushing			Obstacle Pushing		
	Cost	Step	Goal	Cost	Step	Goal
BOA	**0.65**	**4.74**	**99%**	**2.53**	9.15	**86%**
BOA-REF	0.76	4.88	99%	3.29	**9.00**	78%
CMA-ES	0.98	4.96	96%	4.11	9.73	72%
Hand Tuned	0.78	4.99	98%	4.12	9.53	72%

treated as a diagonal matrix, is optimized within $[0, 10]$ and its pivot elements are converged to 4.1, 5.2 and 5.8 respectively.



(a) Optimized $\lambda \in [0, 1]$ (b) Optimized $\Sigma \in [0, 10]$

Fig. 3: Distribution of optimized parameters

In addition, we measured the average runtime of the CMA, Bayesian Reference, and our algorithm, the result is shown in TableV. It can be observed that the average single-step runtime of the CMA-ES algorithm is significantly lower than that of the Bayesian algorithms, while the runtime of our algorithm is slightly behind Bayesian Reference, with the error still within an acceptable range.

TABLE V: Runtime

Algorithm	**Runtime**
CMA-ES	0.001
Bayesian-Opt	0.037
Ours	0.042

V. Conclusions

In conclusion, this project focuses on the application of Bayesian Optimization Algorithm (BOA) for optimizing the hyperparameters of Model Predictive Path Integral (MPPI) control in a planar pushing task with non-trivial obstacles and different targets. The GPytorch library is employed to implement BOA, effectively balancing exploration and exploitation in the parameter space. The optimized MPPI controller is compared against three baselines: hand-defined parameters, CMA-ES, and Bayesian Reference algorithm. The results demonstrated the effectiveness of BOA in enhancing MPPI performance, contributing to the practical applications of GP in robotics and control systems.

The experimental section of the project tested the BOA implementation using the panda pushing environment, a robotic arm pushing task with obstacles. The experiments were conducted with different numbers of training epochs (10, 50, and 500) to assess the performance of the BOA in comparison to the baselines. The results suggest that the proposed BOA generally performs well compared to the other algorithms, achieving better optimization results in shorter operating steps, and showing improvement in cost function and accuracy. However, it is important to note that the performance of the BOA in the 50-epochs training process for the free pushing task experienced an unexpected decline, which means that BOA requires more training epochs to converge.

Overall, this project highlights the potential of Bayesian Optimization in enhancing MPPI performance by selecting better hyperparameters, offering valuable insights for future research and real-world applications.

References

[1] J. Mockus, "Application of bayesian approach to numerical methods of global and stochastic optimization," *Journal of Global Optimization*, vol. 4, pp. 347–365, 1994.

[2] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, vol. 31, 2018.

[3] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[4] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[5] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, p. 455, 1998.

[6] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "An experimental comparison of bayesian optimization for bipedal locomotion," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 1951–1958.

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[8] J. Mockus, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, vol. 2, p. 117, 1998.

[9] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of global optimization*, vol. 21, pp. 345–383, 2001.

[10] C. E. Rasmussen, C. K. Williams *et al.*, *Gaussian processes for machine learning*. Springer, 2006, vol. 1.

[11] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[12] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[13] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[14] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[15] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014–. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[16] M. Pelikan, D. E. Goldberg, E. Cantú-Paz *et al.*, "Boa: The bayesian optimization algorithm," in *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1. Citeseer, 1999, pp. 525–532.

[17] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.