Optimizing Software Parameter Tuning with Software-in-the-Loop

A Comparative Study of Black-Box Optimization Methods for the Calibration of Transmission Software

Master's thesis in Computer science and engineering

Steffanie Kristiansson
Viktoria Magnusson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Optimizing Software Parameter Tuning with Software-in-the-Loop

A Comparative Study of Black-Box Optimization Methods for the Calibration of Transmission Software

Steffanie Kristiansson
Viktoria Magnusson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Optimizing Software Parameter Tuning with Software-in-the-Loop
A Comparative Study of Black-Box Optimization Methods for the Calibration of Transmission Software
STEFFANIE KRISTIANSSON
VIKTORIA MAGNUSSON

Supervisor: Alasdair Paren, Department of Computer Science and Engineering
Advisor: Simon Lillskog, Volvo AB (Volvo Trucks Technology & Industrial Division, TTI)
Examiner: Krasimir Angelov, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Volvo Battery Electric Truck.

Typeset in LaTeX
Gothenburg, Sweden 2026

Optimizing Software Parameter Tuning with Software-in-the-Loop
A Comparative Study of Black-Box Optimization Methods for the Calibration of Transmission Software
STEFFANIE KRISTIANSSON
VIKTORIA MAGNUSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modern vehicle software systems are complex and contain numerous tunable input parameters that influence the behaviour of the vehicle. Configuring and testing these by hand is time-consuming and expensive. To streamline and automate this process, Software-in-the-Loop (SiL) is used to iteratively run a simulation of the vehicle and its surrounding environment whilst updating the vehicle configuration. This thesis investigates various optimization methods for automatically tuning software parameters in the software system in a Battery Electric Vehicle (BEV) using SiL. The objective is to minimize the time between a requested signal and the actual signal. Due to its simulation-based nature, the objective function is gradient-free and expensive and is therefore treated as a Black-Box Optimization (BBO) problem. Investigated methods include Bayesian Optimization (BO), Particle Swarm Optimization (PSO), Covariance Matrix Adaption Evolution Strategy (CMA-ES), and Particle Swarm Optimization Bayesian Optimization (PSOBO), as well as baseline methods for comparison. The results indicate that advanced optimization algorithms consistently achieve lower objective values than the baseline methods. CMA-ES demonstrates the best performance across all test cases. Future work includes investigating the transferability of the optimized parameters from the simulation to a physical vehicle.

Keywords: Software-in-the-Loop, Black-Box Optimization, Bayesian Optimization, Particle Swarm Optimization, CMA Evolution Strategy, Hybrid Algorithm, Simulation-based Optimization

Acknowledgements

We extend our sincerest gratitude to our academic supervisor, Alasdair Paren, and our company advisor, Simon Lillskog for their support, steadfast guidance, and invaluable insights. Additionally, we would like to thank Siddarth Srinivas and Julien Ferri. Finally, we would also like to thank the Clutch and Shift team at Volvo TTI for their continuous support on this project as well as Volvo Group for this opportunity.

Steffanie Kristiansson, Viktoria Magnusson
Gothenburg, 2026-06-02

Contents

List of Acronyms

SiL	Software-in-the-Loop
BO	Bayesian Optimization
PSO	Particle Swarm Optimization
PSOBO	Particle Swarm Optimization Bayesian Optimization
GA	Genetic Algorithm
CMA-ES	Covariance Matrix Adaption Evolution Strategy
OF	Objective Function
BBO	Black-Box Optimization
PID	Proportional-Integral-Derivative
GS*	Grid Search*
GS	Grid Search
RS	Random Search
ECU	Electronic Control Unit
BEV	Battery Electric Vehicle
GP	Gaussian Process
EI	Expected Improvement
GSP	Global Simulation Platform
IAE	Integral Absolute Error

Glossary

1. ***Action request*** *is a signal generated by the driver or simulation to the vehicle or vehicle plant model to request a change in the vehicle behaviour or a desired action, such as increasing motor torque or accelerating.*

2. ***Algorithm setting*** *is a configurable parameter of an optimization algorithm that controls its search behaviour, stopping conditions, or computational execution. It is not part of the simulation model being optimized.*

3. ***Optimizable variable*** *is a scalar element of a software parameter that is adjusted by an optimization algorithm. It corresponds to an element of a software parameter that is represented by as a scalar, array, matrix, or a nested collection.*

4. ***Optimization algorithm*** *in this context is an algorithm that iteratively adjust variables to minimizes an objective function.*

5. ***Simulation*** *is a virtual representation of a physical system and its environment, used to evaluate the system behaviour under specified conditions.*

6. ***Software parameter*** *is a configurable entity within the software system that influences system behaviour and can be adjusted during the tuning process. It may consist of a singular scalar value, array, matrix or a collection of such elements. It serves as an input to vehicle specific software.*

7. ***Vehicle plant model*** *is the simulation model that emulates a vehicle, ensuring realistic input to the control algorithms.*

8. ***Vehicle specific software*** *is software that defines the behaviour and configuration of a particular vehicle type or system, and can be deployed in a physical vehicle or a vehicle plant model.*

List of Figures

List of Tables

List of Listings

1

Introduction

The software in trucks consists of many parameters that control communication and behaviour between Electronic Control Units (ECUs). These parameters can, for example, influence the automatic gear selection and speed limiting. They can be tuned toward optimal values to achieve the best performance in many aspects, such as comfort of the driver, the surrounding environment, as well as efficiency and noise pollution.

A software parameter can be considered a *degree of freedom* in the software, which affects functionality both running in parallel and in sequence. In practice, developers tune only a subset of parameters and aim for performance that is "good enough" rather than optimal, since there is no obvious way in the simulation tool of determining where to find promising regions in the search space of parameter values. This process is often guided by engineering judgement in simulation models and validated through iterative physical testing, which makes the outcome subjective and time-consuming.

At Volvo Group, simulations are used to test and tune parameter values. This is a difficult task since many control functions interact simultaneously, the work space grows quickly with the number of parameters, and running the simulation is computationally expensive.

1.1 Background

At Volvo Group, simulations are executed with Simulink through their Global Simulation Platform (GSP) for efficient testing and tuning of vehicle models. Currently, the software parameter tuning is performed manually by a person who selects a vehicle model, inputs desired parameters, runs the simulation, and receives outputs, as illustrated in Figure 1.1. This process is time-consuming and evaluation of the results are dependent on the person's judgement. This thesis will build upon that environment with modifications enabling Software-in-the-Loop (SiL) to streamline and automate the tuning of software parameters related to the gearbox control system, without need of supervision.

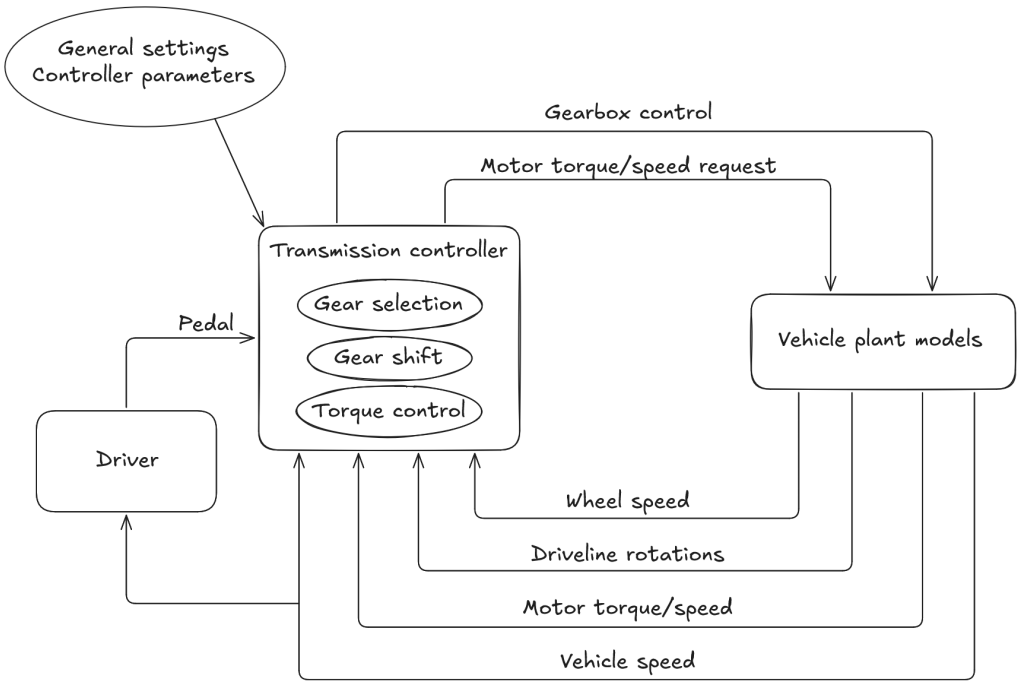


Figure 1.1: A simplified illustration of the transmission control system model.

SiL is a simulation technique used to validate software in a virtual environment in a closed loop, supporting the development stages of finding bugs and possible optimizations in the software before deploying code in hardware. In a SiL setup, the control software is executed as compiled code in a simulation tool, in this case Simulink, of various models. This allows repeatable and deterministic tests of truck model dynamics without hardware and physical environment dependencies, which makes it suitable for iterative optimization of software controller parameters. In the context of this project, the simulation is a virtual environment that emulates a BEV, including both the vehicle and its surroundings.

Software control parameters are the considered input parameters that control various vehicle functions, including motor torque and gearbox operation in the truck. There are hundreds of such parameters; therefore, this thesis limits the set of parameters to a select few that are related to gearbox control.

1.2 Aim

This thesis aims to explore whether automatic optimization techniques can be used to identify suitable software parameters for the vehicle-specific software using SiL.

Furthermore, we aim to assess which optimization technique is most suitable for this type of simulation-based optimization.

1.3 Limitations

The thesis limits the scope to a specific type of truck model, BEV, which is a vehicle that is powered by electricity that is stored in a battery system and differs in many ways to usual diesel trucks.

The simulation model will be treated as a black box, which means the optimization can be computationally expensive, as the algorithm relies on repeated sampling of candidate solutions. This implies that the Objective Function (OF) is derivative- and gradient-free, which restricts which algorithms can be used in the framework.

Further constraints for this thesis are limited computational resources and a limited search space, which restricts the number of parameter combinations that can be evaluated and the complexity of the optimization process.

The algorithm settings of the optimization algorithms will be manually configured, and no automatic search algorithms will be implemented to find the best settings.

1.4 Report Structure

The following chapter presents current literature on the subject of this thesis, which serves as foundation for the chosen approaches conducted. It then presents the necessary theoretical background, followed by the methodological approach. The results of the experiments conducted in this thesis are presented and compared. Finally, the findings are discussed and concluded.

2

Related Work

This chapter reviews related work in simulation-based parameter optimization.

Previous work by S. Borg and V. Hui [1] addressed a similar problem, namely the optimization of input parameters for engine calibration of fuel-driven engines using BBO algorithms. They investigated Genetic Algorithm (GA), BO, and a hybrid method of these approaches. These generated superior results compared to earlier work at Volvo. They implemented a framework to evaluate the engine models to perform this optimization. The underlying structure of the framework can be appropriately modified for use in this thesis. Building upon this framework, the main modification required is the adaptation of a whole vehicle system simulation rather than a subsystem level. Their work is tailored to the combustion engine, which means an application of motor control to wheels remains unexplored.

Regarding parameter optimization in SiL environments, there is another study by Scheuerle et al. [2] that presents a successful approach. They developed a framework in the company's internal environment that uses PSO, which gave promising results but needs more research and tests.

Huang et al. [3] compares their test parameter tuning using a Gaussian process to two baselines: tuning by hand and grid search. This shows a clear indication on whether their algorithm converges toward a better optimum than simple sampling across the search space. Z. Fei et al. [4] present results of the validation process using SiL-testing, which shows the effectiveness and precision of SiL as a testing technique. This is achieved by using two evaluation techniques, Pearson correlation and Relative Root Mean Squared Error. The study proposes data synchronisation to tune only one parameter. This is achieved by synchronised repetitions, requiring multiple physical tests using the hardware. These evaluation metrics are relevant in this context, but our problem involves tuning multiple parameters simultaneously within a larger search space.

In summary, existing work has investigated the use of different methods to optimize parameter tuning with promising results. However, these studies primarily focus on fuel-driven vehicles and rely on models and parameter structure specific to those systems. Although BEVs share several characteristics with fuel-driven vehicles, the main difference is that BEV has en electric motor that controls the torque on the wheels, instead of an engine. Furthermore, current literature highlights the need for further research, tests, and exploration among different optimization methods. This

thesis builds upon current literature to explore and evaluate different methods for software parameter tuning in BEVs.

3

Theory

This chapter presents the theoretical background and key concepts required to understand the work in this thesis. It begins by introducing the domain and simulation environment, followed by an overview of single- and multi-objective optimization. It then introduces the characteristics of the objective function and the methods used in this study.

Consistently used notation:

d	Number of dimensions (number of optimizable variables)
$A \subset \mathbb{R}^d$	Parameter space
$x_i \in \mathbb{R}$	Scalar optimizable variable (i-th parameter)
$\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$	Parameter vector of the input
$x_i^{(0)}$	Nominal value of optimizable variable i

3.1 Domain

This work is situated in the area of vehicle simulation and control system evaluation. A simulation environment has previously been developed to enable testing of vehicle software in a controlled and repeatable setting that closely approximates real-world vehicle behaviour.

The environment is implemented in MATLAB and Simulink [5], [6], where software parameters are defined in patch files that are incorporated into the vehicle plant model's transmission software. These parameters are evaluated through simulation in scenarios that reflect real-world conditions. Each optimizable variable of the software parameters has a nominal value, $x_i^{(0)}$, which is the current best solution provided by Volvo Group.

Previous thesis work by S. Borg and V. Hui [1] extended this environment by introducing automation and iterative execution. This, in combination with optimization algorithms, allows software parameter configurations to be evaluated repeatedly through an optimization procedure, in which an algorithm modifies the parameter values with the objective of minimizing fuel consumption. Their optimization process is mainly implemented in Python.

3.2 Single- and Multi-Objective Optimization

In optimization problems, one or more objectives are defined to be either maximized or minimized. Depending on the goal and set constraints, a single objective problem or a multi-objective problem can be defined, in which there is a single solution or a set of solutions respectively [7].

In this thesis, the problem formulation is limited to single objective optimization.

3.3 Characteristics of the Objective Function

The OF is formulated as a cost function to be minimized:

$$\min_{\mathbf{x} \in A} f(\mathbf{x}), \tag{3.1}$$

where A is the parameter space and $\mathbf{x}$ is the input vector. The OF is defined as $f(\mathbf{x}) = g(S(\mathbf{x}))$, where:

- $S(\cdot)$: The simulation
- $g(\cdot)$: Post-processing function that returns scalar cost.

The internal structure of the simulation is unknown and is therefore treated as a black box. The OF can be expressed as:

$$f(\mathbf{x}) : A \to \mathbb{R},$$

where d denotes the dimensionality and $\mathbf{x}$ represents the parameter vector. The output is a scalar cost. The dimensionality corresponds to the number of tunable parameters in the simulation.

The OF has several characteristics that limit the choice of optimization algorithms and methods available. These characteristics are used to identify feasible methods.

- **Gradient-Free:** Due to the simulation-based nature of the problem and its unknown internal structure, gradient information is not available.
- **Non-convexity:** The OF is assumed to be non-convex and may therefore contain multiple local minima.
- **Expensive evaluation:** One evaluation requires a full simulation run, which is time-consuming and computationally expensive. This limits the number of evaluations that the algorithm can do, which requires it to be sample efficient.
- **Deterministic:** The output of the simulation is assumed to be consistent between iterations if the input remains unchanged. This simplifies the algorithms used, as stochastic noise will not have to be handled.
- **High dimensionality:** The number of optimizable variables varies between test cases, and high-dimensional cases can cause optimization algorithms to lose efficiency and accuracy.

These characteristics motivate the use of gradient-free, sample-efficient, and widely used algorithms such as Bayesian Optimization, Particle Swarm Optimization, and Covariance Matrix Adaption Evolution Strategy.

3.4 Constraints

The parameter space A is defined as a bounded subset of $\mathbb{R}^d$. Each optimizable variable is bounded due to physical constraints, such that for each degree of freedom $x_i \in [l_i, u_i]$ for $i \in \{1, \ldots, d\}$, where l_i and u_i are the lower and upper bounds, respectively.

An important aspect of developing adaptive systems is the trade-off between exploration and exploitation, where systems must balance the refinement of known promising regions with the acquisition of new information [8]. The size of A determines the search space and has direct implications on how an optimization algorithm balances exploration and exploitation. This means that A changes how the algorithm refines a promising region while exploring the search space to avoid premature convergence.

3.5 Black-Box Optimization Algorithms

BBO refers to optimization problems where the structure of the OF cannot be exploited or is unknown, which arise when execution involves a simulation or otherwise is complex and expensive to evaluate. The simulation receives a vector $\mathbf{x}$ and outputs a scalar value in $\mathbb{R}$ [9]. Since it has black-box access, the information will only be accessed at some query points. The following sections go into detail on algorithms that are commonly used in BBO problems.

3.5.1 Grid Search

Grid Search (GS) is a common algorithm in parameter tuning, especially hyperparameter tuning, that evaluates an OF by exhaustively searching a pre-defined grid of hyperparameter configurations. The entire search space is formed as the Cartesian product of smaller finite hyperparameter domain sets, denoted as G. The optimal configuration is chosen by

$$\arg\min_{\mathbf{x} \in G} f(\mathbf{x})$$

as shown in [10]. The number of unique values of a parameter is referred to as the resolution of the grid [11].

Since GS evaluates every possible combination, it suffers from the "curse of dimensionality", as the number of evaluations increases exponentially with the number of input parameters (increased dimensionality) [11]. For example, if the grid resolutions are r, and there are d dimensions, the total number of evaluations becomes r^d. Therefore, even a slight increase in the dimensionality or grid resolution can make this computationally expensive very fast.

3.5.2 Random Search

RS is another common algorithm for parameter tuning, which evaluates $f(\mathbf{x})$ by repeatedly sampling configurations from a predefined probability distribution p over the search space A. The selected configuration, after T trials, is

$$\arg \min_{\mathbf{t}=1,\ldots,T} f(\mathbf{x_t}),$$

where each point $\mathbf{x}_t \sim p$ is drawn independently [10].

Unlike GS, which evaluates all configurations on a fixed grid, RS explores the search space by random sampling. This generally makes it more efficient in higher-dimensional spaces as it avoids exponential growth in evaluations. However, the performance depends on the sampling distribution and the number of trials [12].

3.5.3 Bayesian Optimization

BO is a class of optimization methods that focuses on solving the problem

$$\min_{\mathbf{x}\in A} f(\mathbf{x}), \tag{3.2}$$

where f is expensive to evaluate, has an unknown and complex internal structure, and is most effective when the number of dimensions d of the input is $d \leq 20$ [13]. It has an efficient sample selection strategy, which limits the number of simulations needed for the algorithm to converge and works by collecting observations of f to gain information about solutions to Equation 3.2 [14].

A BO algorithm consists of two main components: a *Bayesian statistical model* for modelling the OF as a surrogate equipped with uncertainty estimates, and an *acquisition function* that decides where to sample next based on the surrogate model. Using the combination of these in each iteration, the process iteratively improves the parameters [13].

Gaussian Process and Surrogate Model: A Gaussian Process (GP) is the statistical model used to create the surrogate model in BO. This section only gives a short overview of GP, as it is a complex process implemented internally in the used libraries. A GP defines a distribution over possible functions that could describe the objective. Given a set of $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^d$, these function values form a vector $[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_k)]^T$. The GP defines a prior distribution over these functions

$$f(\mathbf{x}_{1:k}) \sim \mathcal{N}(\mu_0(\mathbf{x}_{1:k}), \Sigma_0(\mathbf{x}_{1:k})). \tag{3.3}$$

Suppose we observe $f(\mathbf{x}_{1:n})$ for some n, and we wish to observe a new point. Let $k = n + 1$ and the new point to be evaluated $\mathbf{x}_k$.

$$f(\mathbf{x}) \mid f(\mathbf{x}_{1:n}) \sim \mathcal{N}(\mu_n(\mathbf{x}), \sigma_n^2(\mathbf{x})). \tag{3.4}$$

From this, we can get the posterior mean $\mu_n(\mathbf{x})$ and posterior variance. $\sigma_n^2(\mathbf{x})$. This output, which is known as the surrogate model, is then used by the acquisition

function to select the next point to evaluate, balancing exploration and exploitation [13].

Acquisition Function: One of the most popular acquisition functions is Expected Improvement (EI) [15], due to its relatively good performance on most problems, simple implementation and usage [13]. The core idea is to predict the improvement to a new evaluation point from $\mathbf{x}_n$ with observed value $f(\mathbf{x}_n)$ at iteration n. Let the best value be $f_n^* = \min_{m \leq n} f(\mathbf{x}_m)$ of this point. Suppose we make a new evaluation at a new point $\mathbf{x}$. The new best value will either be $f(\mathbf{x})$ if $f(\mathbf{x}) \leq f_n^*$ or f_n^* if $f_n^* \leq f(\mathbf{x})$. We want to take the expected value and choose $\mathbf{x}$ to minimize it. The expected improvement is defined as

$$EI_n(\mathbf{x}) := E_n[[f(\mathbf{x}) - f_n^*]^+], \tag{3.5}$$

where $E_n[\cdot]$ is the expectation with respect to the posterior distribution of $f(\mathbf{x})$ at the n:th observation, as described in Equation 3.4. The expected improvement can be evaluated in closed form, where the EI algorithm then evaluates the part with the largest expected improvement:

$$\mathbf{x}_{n+1} = \arg\max \mathrm{EI}_n(\mathbf{x}), \tag{3.6}$$

which can then be used for the next iteration [13].

The process can be summarised in the following steps, as described by Wang et al. [16]:

1. **Initialization:** Sample an initial set of data points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset A$ and evaluate the OF on these points $\mathbf{Y} \leftarrow \{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)\}$.
2. **Surrogate model:** Fit surrogate model $\mathcal{M}$ on $\mathbf{X}$ and $\mathbf{Y}$.
3. **Acquisition step:** Pick the next evaluation point $\mathbf{x}_{n+1}$ by maximising the acquisition function.
4. **Evaluation:** Evaluate $y_{n+1} = f(\mathbf{x}_{n+1})$.
5. **Update dataset:** $\mathbf{X} \leftarrow [\mathbf{x}; \mathbf{x}_{n+1}], \quad \mathbf{Y} \leftarrow [\mathbf{Y}; \mathbf{y}_{n+1}]$.
6. **Iteration and termination:** Iterate the process starting from step 2 until the stop condition has been met.

3.5.4 Particle Swarm Optimization

PSO is an optimization technique that mimics a group of animals, such as a flock of birds or a school of fish. It is self-adaptive, stochastic, and based on a population used to determine the best solution of a complex search space. The population consists of a collection of particles with initial positions and velocities. To find the best solution, iterative evaluations of the OF are made for each particle and the neighbouring particles, and updates the particles, velocities, and neighbours based on previous evaluation, and stops when a stopping criterion is met [17]. A stopping criterion is a predefined rule used for terminating the iterative process.

Each particle is represented as a position $\mathbf{x}_i^g$ that corresponds to a candidate solution, and a velocity $\mathbf{v}_i^g$ that determines where and at what speed the particle is moving. The movement is also influenced by an inertia $\omega \mathbf{v}_i^g$, where ω is the inertia weight that modifies whether previous instances influence the movement of the particle.

Beyond the inertia, there is a cognitive and social component influencing the velocity of the particles. The cognitive component is based on the personal best position of each individual particle, $\mathbf{p}_i$, and is described by $\mathcal{U}(0, \phi_1) \odot (\mathbf{p}_i - \mathbf{x}_i^g)$ where $\mathcal{U}(0, \phi_1)$ is a uniform distribution of random numbers in $[0, \phi_1]$ generated in each iteration and $\odot$ is component-wise multiplication. The social component is based on the best candidate solution of the group $\mathbf{p}_g$, and is described by $\mathcal{U}(0, \phi_2) \odot (\mathbf{p}_g - \mathbf{x}_i^g)$. These terms are then used to update the velocity

$$\mathbf{v}_i^{g+1} = \omega \mathbf{v}_i^g + \mathcal{U}(0, \phi_1) \odot (\mathbf{p}_i - \mathbf{x}_i^g) + \mathcal{U}(0, \phi_2) \odot (\mathbf{p}_g - \mathbf{x}_i^g) \tag{3.7}$$

and position

$$\mathbf{x}_i^{g+1} = \mathbf{x}_i^g + \mathbf{v}_i^{g+1} \tag{3.8}$$

of each particle [18].

The following is a condensed overview of the algorithm (as described by Poli et al. [18]):

1. **Initialization**: Assign $\mathbf{x}_i^1$ and $\mathbf{v}_i^1$ at random to each particle within the search space.
2. **Fitness evaluation**: Evaluate the fitness of each particle and update the personal and global best fitness evaluation for each particle.
3. **Update velocity and position**: Update the position and velocity of each particle using Equation 3.7 and 3.8.
4. **Iteration and termination**: Iterate the process starting at step 2 and update until the stop criterion has been met.

Variations of the algorithm exist where there is no inertia, and others where other components are introduced.

3.5.5 Genetic Algorithms

GA is a population-based method that evaluates candidate solutions iteratively by applying operators such as selection, crossover, and mutation to find a better solution [19]. This is a strong and widely used optimizing algorithm for Proportional-Integral-Derivative (PID) controllers, as it improves the controller settings stability, resiliency and dynamic response [20].

3.5.6 Covariance Matrix Adaption Evolution Strategy

Evolutionary algorithms are a class of probabilistic optimization algorithms inspired by biological evolution [21]. CMA-ES is a derivative-free algorithm in this class, and works by iteratively sampling a population of candidate solutions from a distribution

and evaluating each sample. Then the distribution is updated with a bias towards more promising regions of the search space using step size, mean, and a covariance matrix. For each generation, the process is repeated to explore the search space.

The multivariate distribution is defined as:

$$\mathbf{x}_i^g \sim \mathcal{N}(\mathbf{m}^g, (\sigma^g)^2\mathbf{C}^g) \text{ for } i = 1, \ldots, \lambda, \tag{3.9}$$

where $\mathbf{m}^g$ is the mean, σ is the step size, and $\mathbf{C}^g$ is the covariance matrix. The covariance matrix is controlled by learning rates that balance adaption speed.

At each generation g, CMA-ES samples candidate solutions $\mathbf{x}_i^g$ using the probability distribution described in Equation 3.9. Then ranking the solutions based on the evaluation of the OF, $f(\mathbf{x}_i^g)$ for all $i \in 1, \ldots, \lambda$, and picks the best candidates μ among them. The mean is updated by taking the weighted mean of μ.

C is adapted according to another evolution path, p_C and the new candidates. p_C is the sum of consecutive steps, which can be referred to as the cumulation. This cumulation disregards the step size.

The step size σ is updated through an evolution path p_σ, where p_σ is the cumulation taking the step size into account. When initializing the algorithm, the evolution paths are empty. The process is iterated until a stopping criterion has been met [22].

1. **Initialization:** Set evolution paths $p_\sigma = 0$, $p_C = 0$, $C = \mathbf{I}$, $g = 0$. Choose the mean and step size, $\mathbf{m} \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}$ depending on the test case.
2. **Sample:** Choose a population of candidate solutions $\{\mathbf{x}_1, \ldots \mathbf{x}_\lambda\} \subset A$ based on the distribution found in Equation 3.9 and evaluate them.
3. **Rank:** Rank the OF evaluations and pick a subset of the best candidates.
4. **Update Distribution:** Update $\mathbf{C}$, σ, and $\mathbf{m}$ using learning rates to control adaption and stability.
5. **Iteration and termination:** Iterate the process starting from step 2 until a stopping criterion has been met.

3.5.7 Particle Swarm Optimization Bayesian Optimization

Li et al. [23] describe a hybrid version of PSO and BO, using PSO as an internal optimizer within BO. PSO is embedded into BOs process to optimize the acquisition function by searching for input that maximize the acquisition function in each iteration, returning the next evaluation point. The surrogate model is sequentially updated on this new evaluation point.

This differs from the process of BO described in Section 3.5.3, where the next evaluation point is obtained directly from the acquisition function during each iteration, in contrast to the hybrid version, using a population-based optimizer.

4

Methods

This chapter describes the methodology of the system and the implementation of the simulation framework. It details the simulation environment structure, the optimization loop, and how different algorithms are implemented and configured. In addition, it explains the preprocessing of input software parameters, formulation of the objective function, and the evaluation design.

4.1 Optimization Framework

The optimization framework was implemented in MATLAB and Simulink [5], [6], customized for BEV models, which differ from previous work [1]. The framework evaluate and compare the performance of different optimization algorithms. An automated SiL testing technique was integrated with realistic vehicle behaviour and scenarios, to apply vehicle software parameter tuning.

The workflow begins with initializing the optimization process and software parameters. For each iteration, an optimization algorithm selects a new input vector that is written to an input file. This input file is used to patch the vehicle information to the vehicle plant model, after which the simulation is executed. The results are saved in an output file and passed on to the post processing function.

The simulation together with the post processing function define the OF, which maps input parameters to the scalar result. An algorithm's stopping condition is checked based on the evaluated result. If it is not, the optimization algorithm generates a new input vector, and the process is repeated. When the stopping criteria are met (such as maximum iterations or threshold been reached), the algorithm is terminated and returns the final solution. The workflow is illustrated in Figure 4.1.

The decision to implement the framework and algorithms in MATLAB was mainly due to compatibility with the existing interface of the simulation. Other strong choices included Python for its efficiency and use in optimization problems, but the process of opening and closing a MATLAB engine each iteration slows down the workflow. Furthermore, MATLAB provides built-in functions and libraries that support the implementation of selected optimization algorithms.

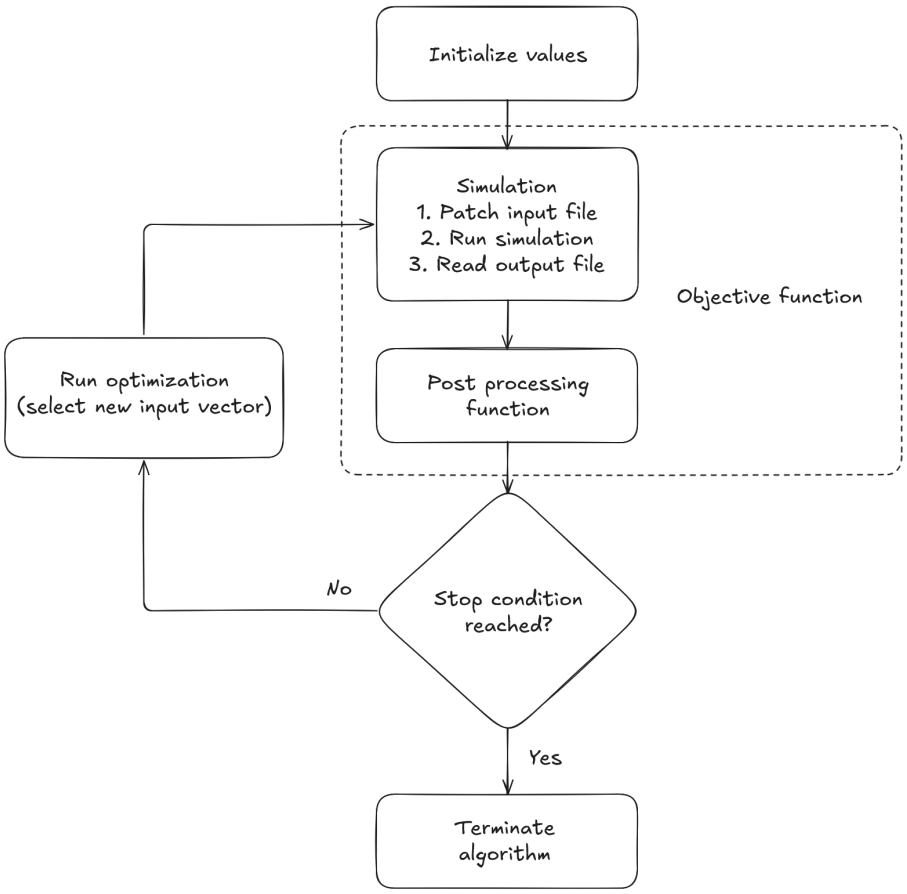


Figure 4.1: Flowchart of the optimization process.

4.2 Problem Formulation

Based on the optimization framework in the previous section, the problem addressed in this thesis is an optimization problem. The objective is to determine a set of parameters that minimizes the cost function, while maintaining realistic vehicle behaviour.

Since the optimization methods are performed and compared on the simulation model, the ground truth is unknown. Therefore, the methods are assessed and compared on their relative performance.

4.2.1 Input Software Parameters

The system of a vehicle is controlled by a set of software parameters, that affect performance and comfort, such as acceleration and braking. These parameters define the configuration of transmission control systems, which in turn affects the vehicle's behaviour and are the main subject of the optimization. The parameters of interest are structured as vectors, as can be seen in Listing 4.1. In this example, there is one input, x, and one output y, with different type of units and values.

In the instance of input seen in Listing 4.1, each individual value in the vectors is considered as an optimizable variable.

```
"parameterName": {
    "x": {
        "unit": "unit_x",
        "value": [x_1, x_2, x_3, x_4]
    },
    "y": {
        "unit": "unit_y",
        "value": [y_1, y_2, y_3, y_4]
    }
}
```

Listing 4.1: Anonymized example of an input parameter with eight optimizable variables.

The selection of parameters and their initial values is considered beyond the scope of this project and is therefore provided by experienced developers with domain knowledge.

4.2.2 Preprocessing of Input Software Parameters

The vehicle plant model requires the input software parameters to be of a specific structure. To streamline the optimization, these parameters are vectorized, while simultaneously preserving their original structural mapping.

As detailed in Section 4.4.3, each parameter has a lower and upper bound that defines the search space. All optimizable variables are normalized to a unit interval $[0, 1]$ before optimization. This transformation ensures a uniform representation of the software parameters, which may otherwise differ in orders of magnitude, that could lead to bias toward larger values. The normalization enables all optimization algorithms to work in a bounded and unbiased search space, while the vehicle plant model evaluations are performed using the corresponding physical values that are obtained through inverse mapping.

4.3 Post Processing Functions

The post processing function was introduced as $g(\cdot)$ in Section 3.3. The objective is to minimize the delay between the actual signal and the request signal following an action request, illustrated in Figure 4.2.

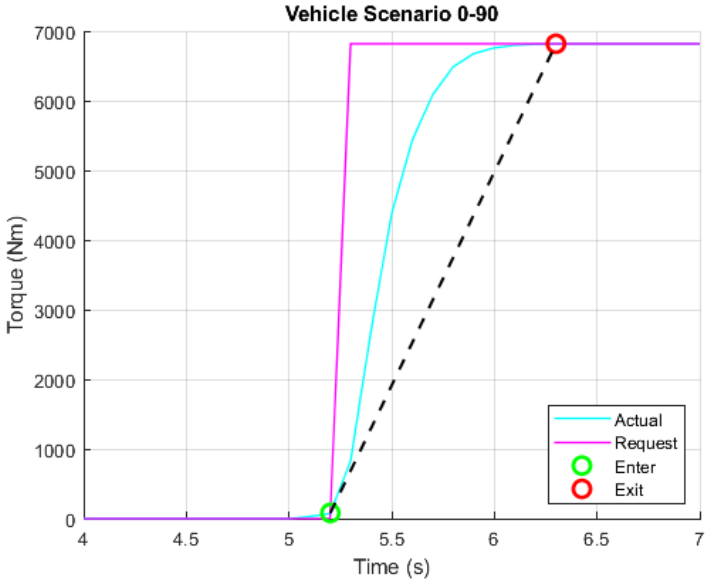


Figure 4.2: Example of difference between two output signals.

4.3.1 Time Difference

The time difference cost function measures the delay between the request signal and actual signal. In Figure 4.2, the scenario shows their differences for a vehicle accelerating from 0 km/h up to 90 km/h, where the request is faster than the actual signal.

Let $\{y_{1,k}\}_{k=1}^{N}, \{y_{2,k}\}_{k=1}^{N}$ be two scalar output signals sampled at discrete time instances $\{t_k\}_{k=1}^{N}$. Although the underlying vehicle dynamics are continuous, outputs from the simulation are only available at discrete sampling instances. Therefore, the cost function is formulated in discrete time using sampled signals.

Define their difference as $e_k = y_{1,k} - y_{2,k}$. To avoid any insignificant initial numerical noise in the beginning of the simulation, let

$$k_0 = \min\{k \in \{1, \ldots, N\} \mid |e_k| > \epsilon_0\}, \quad \epsilon_0 = 10^{-12},$$

if such k exists; otherwise, set the cost $J_{cost} = T_{\max} + 10$, where $T_{\max} = \max_{1 \le k \le N} t_k$, and return.

For $k \geq k_0$, define a threshold condition

$$e_k < -\delta, \quad \delta = 10^{-10}.$$

Let k_{enter} be the first index where this condition becomes true, and k_{exit} be the first index after k_{enter} where it becomes false. If either index does not exist, set $J_{cost} = \max_{k \geq k_0} t_k + 10$.

Otherwise, define the interpolation function:

$$t_{interp}(i) = t_i - e_i \frac{t_{i+1} - t_i}{e_{i+1} - e_i}, \quad i \in \{1, \ldots, N-1\},$$

and let $t_{enter} = t_{interp}(k_{enter})$ and $t_{exit} = t_{interp}(k_{exit})$. Finally, define the cost as the duration spent in the region as $J_{cost} = t_{exit} - t_{enter}$.

4.3.2 Integral Absolute Error

The IAE is defined as the absolute difference between the same two output signals mentioned in Section 4.3.1, i.e. the area between the request and actual signal starting from *entry* and ending in *exit* in Figure 4.2.

Using the same notation as previously, let the interval of interest be given by $I = [t_{enter}, t_{exit}]$. The IAE is then defined as

$$J_{\text{IAE}} = \int_{t_{enter}}^{t_{exit}} |y_1(t) - y_2(t)| \ dt.$$

Since the simulation outputs are sampled at discrete time instances, the integral is approximated using trapezoidal numerical integration.

Denote the sampled and interpolated time instances as $\{\tilde{t}_k\}_{k=1}^{M}$, within the interval I, and let the corresponding scalar output signals be $\{\tilde{y}_{1,k}\}_{k=1}^{M}$ and $\{\tilde{y}_{2,k}\}_{k=1}^{M}$. The discrete approximation then becomes

$$J_{\text{IAE}} \approx \sum_{k=1}^{M-1} \frac{\tilde{t}_{k+1} - \tilde{t}_k}{2} \left(|\tilde{y}_{1,k} - \tilde{y}_{2,k}| + |\tilde{y}_{1,k+1} - \tilde{y}_{2,k+1}| \right).$$

4.4 Design

To get a fair comparison, the algorithms should be implemented and run under circumstances that are as similar as possible.

4.4.1 Dimensionality

For each test case, all algorithms have been compared on the same input with the same number of dimensions.

Since BO tends to become less efficient with higher dimensionality, and PSO remains effective, it is crucial to benchmark and fix the budget on test cases with different numbers of dimensions, as done by N. Hansen et al. [24].

Therefore, we conducted the test in two different ways, one where all input parameters in Appendix A are used, and one where we reduce dimensionality by only keeping $parameter_3$ from Appendix A. This would reduce the number of optimizable variables from $d = 26$ to $d = 8$.

4.4.2 Stop Criteria and Budget

To ensure fair and robust comparison between algorithms when testing on different scenarios and parameter combinations, an objective budget was used for each test run. The budget can be based on two types of stop conditions: to stop after a certain

objective value has been reached, or to limit the time the algorithm gets to run for. These are referred to as fixed-target and fixed-budget, respectively [25].

The budget was set to a fixed number of function evaluations, 100, that apply to all algorithms.

4.4.3 Bounds

To ensure the search is of the same size for each algorithm, the same bounds apply across optimization algorithms. The starting point for defining the search space is the current values that have been hand-tuned through domain and expert knowledge by developers at Volvo. For each optimizable variable x_i, we consider bounds around a nominal value $x_i^{(0)}$:

$$x \in [0.85x_0, 1.15x_0] \text{ and } x \in [0.7x_0, 1.3x_0].$$

These bounds were selected using domain knowledge to enable exploration over the search space while remaining in feasible, optimizable variable value limits.

4.4.4 Algorithm Settings

The algorithm settings of the optimization algorithms impact their performance, e.g. swarm size, population size, and maximum number of evaluations. Respective algorithm used the same settings during experiments to get consistency across all the test cases. They were not adjusted to fit specific problem instances, since that could introduce an unfair advantage to any algorithm [25]. This approach ensures a fair comparison under fair conditions.

Performance variation of the algorithm is often caused by a subset of all its settings, rather than all of them. This defines a smaller set of possible configurations [26], which will scale down the amount of time needed to tune algorithm settings. Therefore, the algorithm settings were picked based on their impact on performance.

4.5 Optimization Methods

The following is a description of each optimization method's implementation and its algorithm settings.

4.5.1 Baseline

To set a baseline and prove that this problem is not solvable by a simple and exhaustive algorithm, we implemented GS and RS, which were later used to benchmark more complex algorithms.

GS goes over all combinations in the entire search space, which, in the case of big search spaces, quickly becomes infeasible. To stay within our budget, we conduct the experiments by only evaluating the first 100 combinations in the grid. We denote this optimization method Grid Search* (GS*). Random search tests random values in

the search space and can, by chance, find a good solution, but that is not guaranteed in each test case.

4.5.2 Bayesian Optimization

Bayesian Optimization was implemented using the function `bayesopt()` from MATLAB's Statistics and Machine Learning Toolbox [27]. It was mainly chosen due to its compatibility with the chosen language and framework. Furthermore, in contrast to a custom implementation of the algorithm, it reduces implementation bias and errors and limits the time needed for implementation. Through experimentation with settings, the algorithm settings in Table 4.1 were identified as the most impactful.

Algorithm setting	**Definition**	**Value**
AcquisitionFunctionName	Function that chooses the next value	EI
IsObjectiveDeterministic	True if objective function is deterministic	True
NumSeedPoints	Number of initial sample of objective evaluations	20
UseParallel	Run objective function in parallel	False
MaxObjectiveEvaluations	Objective function evaluation limit (fixed-budget)	100
MaxTime	Time limit (wall clock time)	Infinite

Table 4.1: Algorithm settings for BO.

We chose to use EI as it is one of the most widely used algorithms [15] and inherently balances exploration and exploitation, making it robust across most problems [28]. Future work could include exploration of different acquisition functions.

4.5.3 Particle Swarm Optimization

The `particleswarm()` function [29] from MATLAB R2021b's Global Optimization Toolbox was used for the implementation of PSO. By providing the lower and upper bounds that an optimizable variable is allowed to range between, an OF, and the number of dimensions, the algorithm tries to find a vector `x` such that the OF obtains a local minimum.

Optional arguments allowed us to override the default algorithm settings. In this work, the primary tunable algorithm settings were the swarm size and the maximum number of iterations, while the others were kept at default values.

Initial tests of the algorithm used a small set of scalar parameters to observe the behaviour and stability of the algorithm. Based on those results, higher-dimensional

algorithm setting sets were evaluated to assess scalability and convergence of the algorithm.

Since PSO evaluates the OF once per particle, in the initialization and each iteration, the total number of evaluations becomes

$$N_{eval} = swarmSize + swarmSize \times maxIter = swarmSize \times (maxIter + 1).$$

This relationship was used to determine the trade-off between computational cost and solution quality. The final configuration of the algorithm settings was selected based on computational efficiency and convergence performance, based on experiments. The chosen values are presented in Table 4.2.

Algorithm setting	**Definition**	**Value**
SwarmSize	Determines how many candidate solutions are explored during optimization	10
MaxIterations	Limit of how many iterations the function takes	9

Table 4.2: Algorithm settings for PSO.

4.5.4 Covariance Matrix Adaption Evolution Strategy

CMA-ES was implemented using an open source MATLAB script, `cmaes.m` [30]. The script, available under the GNU General Public License, follows the standard CMA-ES procedure of adapting the covariance matrix and step size iteratively.

By default, there are a number of termination criteria based on convergence measures, such as lack of improvement or diversity in the population [22]. This can cause the algorithm to end the process before the set budget is exhausted, which changes the budget to effectively be fixed-target. To counter this, these algorithm settings were disabled to allow CMA-ES to spend its allocated number of function evaluations in its entirety. The algorithm setting configurations are shown in Table 4.3.

Due to the structure of the implementation of CMA-ES used in this project, a small implementation dependent overhead in function evaluations occurs. In total, this results in 102 function evaluations, instead of the intended budget of 100 function evaluations. This deviation corresponds to a 2% increase compared to the other algorithms, which is considered negligible. Ensuring that CMA-ES performs 100 function evaluations would lead to a deviation in the behaviour which is to be avoided.

Algorithm setting	**Definition**	**Value**
MaxFunEvals	Determines the maximum number of OF evaluations (fixed budget)	100
popsize	Determines how many candidate solutions are explored in each generation	10
insigma	Determines the initial step-size (σ)	0.3
xstart	Initial search point	0
TolX	Stop process if difference in search distribution is smaller than TolX	0 (disabled)
TolFun	Stop process if variation in objective values is smaller than TolFun	0 (disabled)
TolHistFun	Stop process if variation in objective values is smaller than TolHistFun in a certain window	0 (disabled)
StopOnWarning	Stop process if warnings are received	'no'

Table 4.3: Algorithm settings for CMA-ES.

4.5.5 Particle Swarm Optimization Bayesian Optimization

The hybrid optimization method PSOBO is inspired by Li et al. [23], where PSO is embedded into BO to optimize the acquisition function. In contrast, the approach used in this work separates the two methods into separate phases, where PSO is used to globally explore regions in the search space, followed by BO for local refinement of the best candidate solutions.

The optimization was executed in two phases: the PSO-phase followed by the BO-phase. PSO is first executed to explore the search space and generate a set of candidate solutions. These candidate solutions are then used to initialize the BO-phase.

When PSO hits the stop criterion, BO builds the surrogate model based on the set of candidate solutions generated by PSO to refine the solutions that have been explored instead of sampling random seed points. This is done by passing the candidate solutions to the BO algorithm through the algorithm setting *initalX* in `bayesopt()`. If the number of these candidates is less than the number of seed points, BO generates the rest randomly.

The budget for the algorithms was distributed equally, resulting in 50 function evaluations each, presented in Table 4.4. This distribution was chosen to give each algorithm a fair chance to explore, and also to refine the best found solution.

Algorithm	Algorithm setting	Value
PSO	SwarmSize	10
	MaxIterations	4
BO	MaxObjectiveEvaluations	50
	NumSeedPoints	20

Table 4.4: Algroithm settings for the hybrid PSOBO method.

4.6 Test Cases

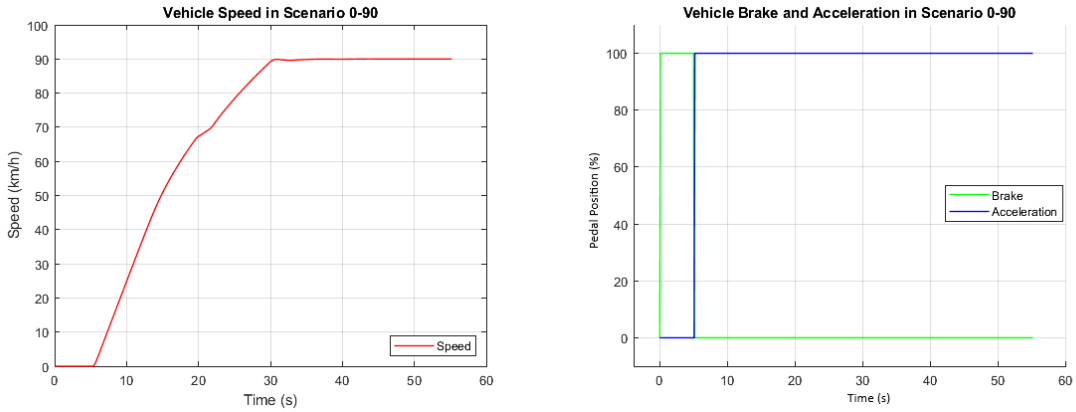
To assess the performance of the optimization algorithms, a set of test cases was defined. The purpose of the test cases is to objectively evaluate each algorithm on:

- Solution quality
- Sensitivity to both bounds and dimensionality
- The minimum cost
- The runtime

In this work, a test case refers to a predefined simulation scenario in which a vehicle plant model receives a sequence of action requests in a specific environment (such as accelerating on a road) in combination with a set of input parameters to be optimized upon, and the size of the search space.

The full input consists of three software parameters. These were represented by two vectors, with a total of 26 optimizable variables. The complete specification of these parameters is provided in Appendix A. To study the impact of dimensionality in the performance of our methods, a reduced problem with 8 variables was also conducted by selecting just one software parameter, $parameter_3$ (see Appendix A for details).

Due to the computational expense of each simulation, a simple but representative simulation scenario was chosen, which is referred to as 0 kmph $\rightarrow$ 90 kmph. In this scenario, the vehicle starts from standstill on a flat surface and accelerates up to 90 kmph, continues at this speed for 600 m, after which the simulation is terminated. This scenario captures the acceleration phase followed by a steady speed, making it suitable for evaluating the parameter changes impact on vehicle performance. A visualisation of the vehicle's behaviour and action requests can be seen in Figure 4.3.



(a) Vehicle speed.

(b) Brake and acceleration requests.

Figure 4.3: Visualisation of actions during simulation of scenario $0 \rightarrow 90$.

The search space for each parameter is defined as a bounded interval and is applied on each optimizable variable. Two levels of variability are considered, a narrow range of $\pm 15\%$ and a wider range of $\pm 30\%$, from the nominal values. This is done in order to investigate whether the chosen methods can refine pre-existing, hand tuned solutions, in addition to assess the methods' ability to explore other regions for better results.

Additionally, Case 4* was created to test the second post processing function mentioned in Section 4.3.2. The test was conducted by running each method once with a budget of 100 function evaluations on 26D and $\pm 30\%$ bounds around the nominal values.

The combination of these variations result in five test cases presented in Table 4.5.

Test Case	Dimensionality	Bounds	Runs	Budget
Case 1	8	$\pm 15\%$	3	100
Case 2	8	$\pm 30\%$	3	100
Case 3	26	$\pm 15\%$	3	100
Case 4	26	$\pm 30\%$	3	100
Case 4*	26	$\pm 30\%$	1	100

Table 4.5: Test cases of different dimensionality, search space bounds and runs.

4.7 Experiment For Result Verification

Since simulation output signals are sampled at discrete time instances, the resulting evaluations are also discrete. The cost function defined in Section 4.3.1 measures the duration spent in a region, but it doesn't measure the accumulated deviation between the signals.

To further evaluate the optimization methods' behaviour, IAE was considered as an additional experiment (defined in Section 4.3.2). The purpose of this experiment

was to investigate whether the relative performance of the optimization methods remained consistent with a different cost formulation that measures the accumulated signal deviation over time.

4.8 Performance Metrics

Although the OF is deterministic, the optimization methods used in this thesis are stochastic, meaning their performance can vary between runs. This means that one run per method is not enough to assess its performance.

To compare each method fairly, they were run three times and assessed on three metrics:

- Mean
- Standard deviation
- Mean runtime (wall clock time)

All experiments were carried out on computers with AMD Ryzen 5 PRO processor with Radeon 760M Graphics and 32 GB RAM, running on Windows 11 and MATLAB R2021b.

5

Results

This chapter presents results obtained from evaluating optimization algorithms in four test cases. Each method was applied to computationally expensive simulation-based OF under varying search space sizes and dimensions. The results are presented in terms of mean cost value, standard deviation, and mean runtime over repeated runs. Each test case is presented separately, followed by a cross-case comparison.

5.1 Nominal values

Running the objective function using the nominal values, $x_i^{(0)}$, yielded a cost of 1.099. These values are the current best that have been tuned by hand using domain and expert knowledge.

5.2 Case 1 (8D, $\pm 15\%$)

Method	Mean cost	Std. dev.	Mean runtime (s)
RS	0.999	0.000001	**8993**
GS*	0.999	0.0	10094
PSO	0.999	0.0	9719
BO	0.999	0.000001	9235
CMA-ES	**0.531**	0.057933	10183
PSOBO	0.898	0.0	9822

Table 5.1: Test case results of 8 dimensions with bounds $[0.85x_0, 1.15x_0]$.

CMA-ES achieved the lowest mean cost value of 0.531, outperforming the other methods. PSOBO yielded the second-lowest mean cost of 0.898. GS*, PSO, and BO converged to nearly identical cost, whilst RS performed the worst. Plots corresponding to all runs of each method are provided in Appendix B.1.1.

5.3 Case 2 (8D, ±30%)

Method	Mean cost	Std. dev.	Mean runtime (s)
RS	0.899	0.000003	10052
GS*	0.899	0.0	18553
PSO	0.899	0.0	15528
BO	0.899	0.000002	**9185**
CMA-ES	**0.431**	0.058327	10121
PSOBO	0.698	0.0	10264

Table 5.2: Test case results of 8 dimensions with bounds $[0.7x_0, 1.3x_0]$.

Expanding the search space bounds to ±30% reduced the achieved mean cost values for almost every algorithm. Again, CMA-ES produced the best result with a mean cost of 0.431, while RS performed worst again. Plots corresponding to all runs of each method are provided in Appendix B.1.2.

5.4 Case 3 (26D, ±15%)

Method	Mean cost	Std. dev.	Mean runtime (s)
RS	0.999	0.000005	10262
GS*	0.999	0.0	10830
PSO	0.999	0.0	17504
BO	0.832	0.057724	16734
CMA-ES	**0.632**	0.057779	10532
PSOBO	0.732	0.057816	**10022**

Table 5.3: Test case results of 26 dimensions with bounds $[0.85x_0, 1.15x_0]$.

Increasing the dimensionality to 26 optimizable variables, CMA-ES maintained the lowest mean cost, whereas RS, GS*, and PSO converged to similar but significantly worse mean costs. Plots corresponding to all runs of each method are provided in Appendix B.2.1.

5.5 Case 4 (26D, ±30%)

Method	Mean cost	Std. dev.	Mean runtime (s)
RS	0.798	0.000007	8880
GS*	0.798	0.0	9765
PSO	0.798	0.0	**8759**
BO	0.631	0.057722	9221
CMA-ES	**0.431**	0.057714	9484
PSOBO	0.531	0.057853	9993

Table 5.4: Test case results of 26 dimensions with bounds $[0.7x_0, 1.3x_0]$.

With increased dimensionality and search space bound, CMA-ES continued to outperform other algorithms. Regarding the previous case with the same dimensionality, all algorithms had improved mean cost. Plots corresponding to all runs of each method are provided in Appendix B.2.2, and an overview comparing each method in one run can be seen in Figure 5.1.

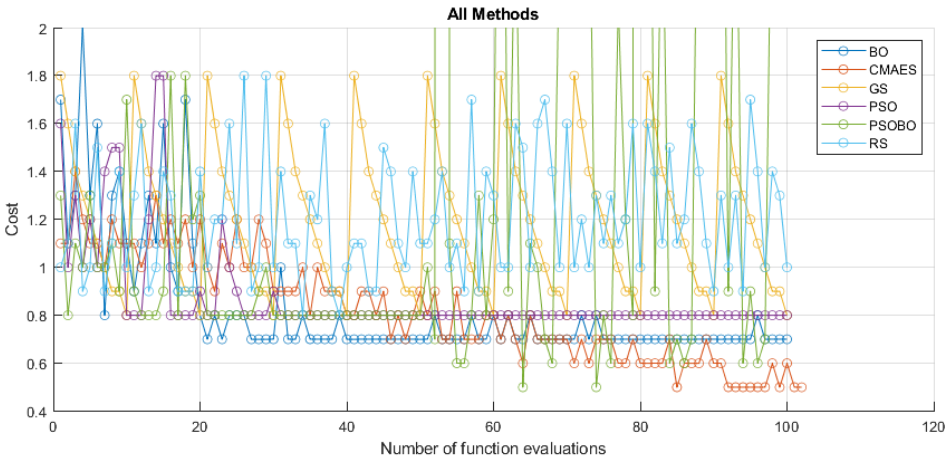


Figure 5.1: A comparative plot of one run of all methods.

5.6 Case 4* (26D, ±30%)

Method	Min area	Min cost	Runtime (s)
RS	976	0.798	**10660**
PSO	972	0.798	10748
BO	771	0.598	10782
CMA-ES	**506**	**0.497**	12072
PSOBO	749	0.498	11222

Table 5.5: IAE results of 26 dimensions with bounds $[0.7x_0, 1.3x_0]$.

According to the results of the IAE experiment, CMA-ES continues to outperform all other methods, and their ranking remains the same. However, BO and PSOBO improved in relation to the other methods, while PSO performed similarly to RS. GS* was left out of this experiment due to its predictable behaviour observed in previous test cases and time restraints.

Plots of respective runs can be seen in Appendix B.3, and the best resulting area based on the output signals can be seen in Figure 5.2.

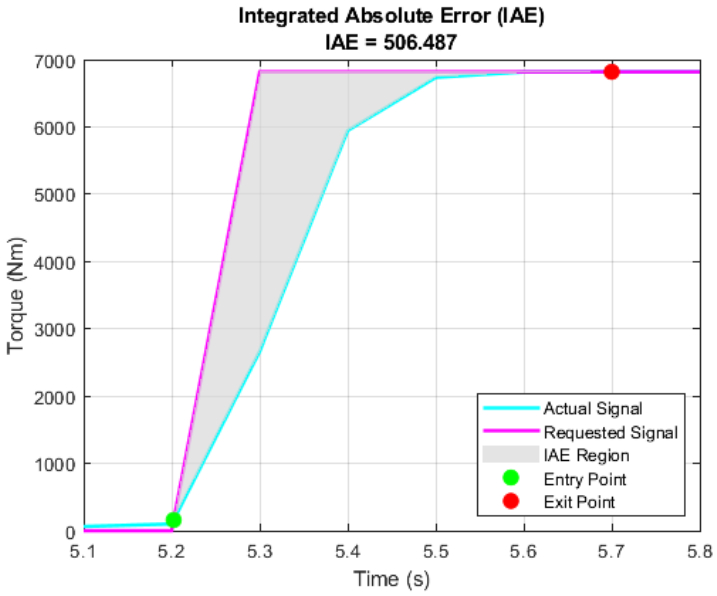


Figure 5.2: Best results of IAE (CMA-ES) in Case 4*.

5.7 Summary of Results

According to our results, CMA-ES consistently outperforms every other method across all test cases.

Case	Method	Mean cost	Std. dev.	Mean runtime (s)
1	CMA-ES	0.531	0.057933	10183
2	CMA-ES	**0.431**	0.058327	10121
3	CMA-ES	0.632	0.057779	10532
4	CMA-ES	**0.431**	0.057714	**9484**

Table 5.6: Best performance metrics across test cases.

The best performing method in case 4* is CMA-ES with a minimum area of 506, its corresponding minimum cost of 0.497, and a runtime of 12072 seconds.

6

Discussion

This chapter starts by discussing the results presented in the previous chapter and the performance of each method. Furthermore, it dives into the effect of increasing bounds and dimensionality, followed by a comparison and analysis of the post processing functions. Finally, it discusses the simulation-to-real-world gap and potential future work within the area.

6.1 Performance of Advanced Methods

According to our results, some of the more advanced optimization methods outperform random and exhaustive searches in SiL. The baseline methods, GS* and RS, were primarily included in the study to evaluate whether using relatively simple strategies was sufficient in order to identify competitive input parameters. The results, however, indicate that both methods converged to solutions with comparatively high cost across all test cases.

From the results observed in Section 5, CMA-ES consistently achieved the lowest mean cost across test cases regardless of dimensionality and search space size in comparison to both the advanced and baseline optimization methods. Across test cases 1-4, CMA-ES had a non-zero standard deviation, indicating varying behaviour. In contrast to PSO, which gave identical results across all runs.

Since the signals are obtained from simulation output, the first post processing function yields discrete values. The optimization landscape may therefore contain discontinuities, plateaus, and locally flat regions. This likely decreases the effect that input vector modifications have on the cost and further reduce the exploration in the optimization methods.

The flatness of the optimization landscape is a possible explanation for PSO's performance, since small differences in candidate solutions provide little information for particle movement, which can increase the risk for stagnation. The graphs in Appendix B presenting the progression of the method suggest this may be the case. The flat landscape may also make it challenging for BO to build the surrogate model and decrease the acquisition function's ability to obtain information about promising regions.

In cases 1 and 2, the results from Table 5.1 and 5.2 suggest that PSO and BO may be stuck in the same local minimum. PSO has been noted to have a tendency to get

stuck in local optima, which can cause the algorithm to converge to a bad solution prematurely [18]. Both PSOs and BOs behaviour may be due to a large flat region where the swarm and acquisition function have problems finding enough variation in OF evaluations to explore further, although this cannot be verified.

Interestingly, PSOBO outperforms both PSO and BO. One possible explanation is the increased exploration from combining PSO and BO. Another is that BO builds its surrogate model on a better sample than a random distribution of seeds, due to PSO's selection of candidate solutions.

Two of the explored methods, PSO and CMA-ES, are both population-based optimization algorithms; however, PSO achieved a higher mean cost in comparison. One reason could be that PSO is more likely to converge into a subspace of the search space, especially when the population size is small and the budget is restricted. However, whereas many population-based algorithms risk degenerating into local subspaces, CMA-ES avoids degeneration by its learning rates and step size, even for small population sizes [22].

6.2 Effect of Search Space Bounds

The results show that performance improves across all optimization methods in both 8 and 26 dimensions when increasing the bounds. Since the larger search space contains the previous smaller search space, increasing the bounds cannot exclude previously found solutions and can therefore preserve or improve the optimal achievable solution. However, a larger search space might lead to unintended behaviour of the vehicle.

One possible explanation is that increasing the search space allowed the methods to find better minima, even with the same budget. This suggests that the bounds can be expanded to explore regions that a developer never would have considered, nor has the time to test and verify. Experiments with a larger budget and expanded search bounds would allow long experimental runs that searches for promising regions that could improve performance.

This could potentially indicate that the search space has been unintentionally restricted to suboptimal regions.

6.3 Effect of Dimensionality

Interestingly, BO's performance improved in relation to the other algorithms when increasing the dimensionality. This contrasts with current literature which suggests that BO deteriorates when dimensionality increases due to difficulties building the surrogate model. One possible explanation is the reduction of flat regions, as more parameter combinations may produce more changes in the cost function. This gives the acquisition function more information about the landscape, which improves the surrogate model. Decreasing the dimensionality possibly flattens the optimization landscape, which could amplify issues with the stagnation of the algorithms.

CMA-ES, on the other hand, performed best in test case 4, which configured 26 optimizable variables.

6.4 IAE Cost Function

The second post processing function was included to verify that the results found by the first post processing function are not caused by its discrete behaviour. By integrating the area between the signals, a more continuous cost function can be achieved, which decreases flatness in the optimization landscape. Since the results were similar across test cases and the standard deviation was low, each method was only run once on test case 4*.

The similarity in minimum cost between case 4 and 4* suggests that the discrete output does not remove the optimization methods' ability to explore the landscape.

PSOBO and BO do, however, improve in relation to the other algorithms. This indicates that the acquisition function could extract more information when the cost function is more sensitive to change in input.

Therefore, the additional post processing function complements and validates the result of the first post processing function. Since the rankings of the cost functions are similar, based on both the original cost function and IAE, it indicates that the observed performance differences are less likely to be caused by the structure of the original cost function or the discretized data.

6.5 Simulation-To-Real-World Gap

The quality of the generated software input in a physical vehicle is dependent on the simulation's ability to accurately represent reality. Any mismatch between the simulation and reality decreases the transferability of the generated software input. This means that a parameter input configuration that generates a low cost does not necessarily produce desirable behaviour in a real world setting. Investigating the behavioural difference between the simulation and a physical vehicle was beyond the scope of this project and, therefore, not accounted for in the development of methods and conducted experiments.

The cost functions only considers the delay between the actual signal and the requested signal. This could make the actual signal overshoot, which potentially makes for an uncomfortable driving experience when the system oscillates as a consequence of this. This reduces predictability and robustness when performing actions when driving the vehicle.

Furthermore, the scenario used in the test cases was limited to a simple acceleration on a flat road. This simplifies the optimization process and limits the time needed to run the optimization process, which allowed us to test more methods and increase our budget. However, in practice, it limits the extent that the input parameters can be generalized to other scenarios.

Additionally, the optimization process was conducted under controlled conditions in the simulation. This does not take varying factors such as friction and environmental effects into account.

6.6 Future Work

One key limitation of the project was the expense of the OF, since each OF evaluation requires a simulation. As a result, running the optimization process was highly time-consuming, which limited the evaluation budget. This constrained the number of parameter combinations that could be explored. Therefore, improving runtime efficiency is an aspect that should be prioritised in future work. One approach is to increase the computational resources, such as distributing the process on multicore CPUs through parallelization. However, whether this approach is applicable on investigated methods, depends on their characteristics. For instance, PSO and RS are possible to parallelize since their evaluation in each iteration is independent, whereas BO builds upon previous evaluations. This would possibly reduce the wall clock time significantly, which would allow for a larger budget.

Increasing runtime efficiency would also streamline the development of finding better algorithm settings, since this process requires iterative evaluations. This is important, since convergence, performance, and solution quality are highly dependent on the algorithm settings. Further research into identifying better input parameter configurations would therefore benefit greatly from increasing efficiency.

Experimenting with algorithm settings and increasing the budget could facilitate exploring larger search spaces for more promising regions, finding better minima within smaller search spaces, and refining solutions to increase their quality. Introducing dynamic algorithm settings and tuning algorithm settings using other optimization algorithms would have a large impact on their performance. Future work could also include exploring alternative optimization algorithms, such as GAs, and compare their performance and behaviour against the results in this thesis.

Further testing on this area could also include larger swarm sizes for PSO. Such an experiment could explore whether the early convergence is due to algorithm settings or that it is not sample efficient enough. Another setting to explore is different step sizes for CMA-ES, and budget distribution with PSOBO. Another algorithm setting that could be experimented with is different acquisition functions for BO.

The current cost functions are, as previously mentioned, single-objective and do not take the oscillation that occurs when a signal overshoots its target into consideration. An improvement that could increase the realism is to expand our OF to a multi-objective function, which would also minimize the oscillation when a signal overshoots. This would potentially cause the optimization process to generate input parameters that lead to a more desirable behaviour in a physical vehicle and further close the simulation-to-real-world gap.

7

Conclusion

This thesis explores the viability of optimizing software parameters for transmission software with SiL. Several optimization methods were evaluated and compared on computationally expensive OFs on varying dimensionalities and search space bounds.

The findings suggest that advanced methods are more capable of navigating the optimization landscape to minimize the cost function than exhaustive and random search methods, which frequently converged to similar results regardless of dimensionality or bounds. Specifically, CMA-ES outperforms all other methods across all test cases. Following CMA-ES in performance is a hybrid method PSOBO that performs especially well with a continuous cost function. Separately, PSO performs similarly to the baseline methods in some test cases, and BO does improve with higher dimensionality and a continuous cost function.

The OF was of a simulation-based nature, which entailed certain limitations and obstacles. Working in a simulation environment implies a non-gradient and unknown structure of the OF, which consequently results in expensive and time-consuming evaluations. This highlights the importance of sample-efficient optimization strategies that balance exploration and exploitation.

Suggested future work includes increasing the computational resources to improve runtime efficiency, especially through parallelisation. Additionally, further testing and tuning of algorithm settings and including multi-objective OFs to improve the input parameters, as well as closing the simulation-to-real-world gap.

The results are promising within our framework; however, further research and testing are needed to conclude the real-world application of the results.

Bibliography

[1] S. Borg and V. Hui, "Multi-Objective Optimization Under the Hood: Engine Calibration via Metaheuristics and Probabilistic Methods," eng, Master's Thesis, Chalmers University of Technology, 2025. Accessed: Feb. 18, 2026. [Online]. Available: http://hdl.handle.net/20.500.12380/310949.

[2] D. Scheuerle, J.-C. Goos, P. Klein, and E. Chrisofakis, "Parameter optimization with software-in-the-loop," in *Proceedings of the Symposium on Development Methodology, Wiesbaden*, Wiesbaden, Germany, Nov. 2023.

[3] Q. Huang et al., "Test parameter tuning with blackbox optimization: A simple yet effective way to improve coverage," 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:250647341.

[4] Z. Fei, M. Andersson, and A. Tingberg, "Correlation of software-in-the-loop simulation with physical testing for autonomous driving," in *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*, 2024, pp. 2050–2057. DOI: 10.1109/ITSC58415.2024.10919543.

[5] The MathWorks, Inc., *MATLAB r2021b*, https://www.mathworks.com, Natick, Massachusetts, United States, 2021.

[6] The MathWorks Inc., *Simulink*, Version R2021b Update 6, The MathWorks Inc., Natick, Massachusetts, USA, 2021.

[7] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: Fundamentals and evolutionary methods," en, *Natural Computing*, vol. 17, no. 3, pp. 585–609, Sep. 2018, ISSN: 1567-7818, 1572-9796. DOI: 10.1007/s11047-018-9685-y. Accessed: Apr. 15, 2026. [Online]. Available: http://link.springer.com/10.1007/s11047-018-9685-y.

[8] J. G. March, "Exploration and exploitation in organizational learning," *Organization Science*, vol. 2, no. 1, pp. 71–87, 1991, ISSN: 10477039, 15265455. Accessed: Apr. 15, 2026. [Online]. Available: http://www.jstor.org/stable/2634940.

[9] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, vol. 9, p. 100 011, 2021, ISSN: 2192-4406. DOI: https://doi.org/10.1016/j.ejco.2021.100011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2192440621001386.

[10] L. Franceschi et al., "Hyperparameter optimization in machine learning," *Foundations and Trends® in Machine Learning*, vol. 18, no. 6, pp. 1054–1201, 2025, ISSN: 1935-8245. DOI: 10.1561/2200000088. [Online]. Available: http://dx.doi.org/10.1561/2200000088.

[11] B. Bischl et al., *Hyperparameter optimization: Foundations, algorithms, best practices and open challenges*, 2021. arXiv: 2107.05847 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2107.05847.

[12] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, pp. 281–305, Feb. 2012, ISSN: 1532-4435. DOI: 10.5555/2188385.2188395.

[13] P. I. Frazier, *A tutorial on bayesian optimization*, 2018. arXiv: 1807.02811 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1807.02811.

[14] J. A. Paulson and C. Tsay, "Bayesian optimization as a flexible and efficient design framework for sustainable process systems," *Current Opinion in Green and Sustainable Chemistry*, vol. 51, p. 100 983, 2025, ISSN: 2452-2236. DOI: https://doi.org/10.1016/j.cogsc.2024.100983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452223624001044.

[15] D. Zhan and H. Xing, "Expected improvement for expensive optimization: A review," *Journal of Global Optimization*, vol. 78, no. 3, pp. 507–544, 2020, ISSN: 1573-2916. DOI: 10.1007/s10898-020-00923-x. [Online]. Available: https://doi.org/10.1007/s10898-020-00923-x.

[16] H. Wang and K. Yang, "Bayesian optimization," in *Many-Criteria Optimization and Decision Analysis: State-of-the-Art, Present Challenges, and Future Perspectives*, D. Brockhoff, M. Emmerich, B. Naujoks, and R. Purshouse, Eds. Cham: Springer International Publishing, 2023, pp. 271–297, ISBN: 978-3-031-25263-1. DOI: 10.1007/978-3-031-25263-1_10. [Online]. Available: https://doi.org/10.1007/978-3-031-25263-1_10.

[17] N. K. Jain, U. Nangia, and J. Jain, "A Review of Particle Swarm Optimization," en, *Journal of The Institution of Engineers (India): Series B*, vol. 99, no. 4, pp. 407–411, Aug. 2018, ISSN: 2250-2106, 2250-2114. DOI: 10.1007/s40031-018-0323-y. Accessed: Jan. 26, 2026. [Online]. Available: http://link.springer.com/10.1007/s40031-018-0323-y.

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

[19] D. Whitley, "A genetic algorithm tutorial," en, *Statistics and Computing*, vol. 4, no. 2, Jun. 1994, ISSN: 0960-3174, 1573-1375. DOI: 10.1007/BF00175354. Accessed: Jan. 29, 2026. [Online]. Available: http://link.springer.com/10.1007/BF00175354.

[20] Y. Shekhar, G. Verma, D. Singh, and S. Yadav, "Genetic algorithm-based optimization of pid controller for magnetic levitation systems: A comparative study," in *2024 International Conference on Control, Computing, Communication and Materials (ICCCCM)*, 2024, pp. 430–435. DOI: 10.1109/ICCCCM61016.2024.11039867.

[21] D. Corne and M. A. Lones, "Evolutionary algorithms," in *Handbook of Heuristics*. Springer International Publishing, 2018, pp. 1–22, ISBN: 9783319071534. DOI: 10.1007/978-3-319-07153-4_27-1. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07153-4_27-1.

[22] N. Hansen, *The cma evolution strategy: A tutorial*, 2023. arXiv: 1604.00772 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1604.00772.

[23] Y. Li and Y. Zhang, *Hyper-parameter estimation method with particle swarm optimization*, 2020. arXiv: 2011.11944 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2011.11944.

[24] N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff, "Coco: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, vol. 36, Mar. 2016. DOI: 10.1080/10556788.2020.1808977.

[25] F.-J. Willemsen, R. Schoonhoven, J. Filipovič, J. O. Tørring, R. van Nieuwpoort, and B. van Werkhoven, "A methodology for comparing optimization algorithms for auto-tuning," *Future Generation Computer Systems*, vol. 159, pp. 489–504, 2024, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2024.05.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24002498.

[26] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," *31st International Conference on Machine Learning, ICML 2014*, vol. 2, pp. 1130–1144, Jan. 2014.

[27] The MathWorks, Inc., *bayesopt function, global optimization toolbox, matlab r2021b*, https://www.mathworks.com/help/stats/bayesopt.html, Natick, Massachusetts, United States, 2021.

[28] H. Ros, N. Chan, M. T. Cook, and D. Shorthouse, "Artificial intelligence and machine learning guided optimization in drug delivery," *Advanced Drug Delivery Reviews*, vol. 232, p. 115 781, 2026, ISSN: 0169-409X. DOI: https://doi.org/10.1016/j.addr.2026.115781. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169409X26000153.

[29] The MathWorks, Inc., *particleswarm function, global optimization toolbox, matlab r2021b*, https://www.mathworks.com/help/gads/particleswarm.html, Natick, Massachusetts, United States, 2021.

[30] N. Hansen, *Cmaes.m: Matlab/octave implementation of cma evolution strategy*, [MATLAB/Octave source code, GNU GPL], 2001–2008. [Online]. Available: https://cma-es.github.io/cmaes_sourcecode_page.html#matlab.

A

Anonymized Input Parameters

This appendix presents the input software parameters used in the vehicle plant model described in Section 4.6. Due to confidentiality constraints, all parameter names, units and optimizable variables have been anonymized. The structure reflects the original format of the data used as input to the model.

Each parameter consists of two main components, x and y, representing the optimizable input and output variables respectively. The field *unit* specifies the measure unit. The *value* field corresponds to the data points.

```
{
    "parameter_1": {
        "x" : {
            "unit": "unit_x",
            "value": [x_1, x_2, x_3, x_4, x_5, x_6]
        },
        "y" : {
            "unit": "unit_y",
            "value": [y_1, y_2, y_3, y_4, y_5, y_6]
        }
    },
    "parameter_2": {
        "x" : {
            "unit": "unit_x",
            "value": [x_1, x_2, x_3]
        },
        "y" : {
            "unit": "unit_y",
            "value": [y_1, y_2, y_3]
        }
    },
    "parameter_3": {
        "x" : {
            "unit": "unit_x",
            "value": [x_1, x_2, x_3, x_4]
        },
        "y" : {
            "unit": "unit_y",
            "value": [y_1, y_2, y_3, y_4]
        }
    }
}
```

Listing A.1: Anonymized software parameter input used in the optimization.

B

Plot Results

In this appendix, the results are presented in plots of each algorithm and every run. It is divided by the number of dimensions and search space bounds.

B.1 Case Results with 8 Dimensions

B.1.1 Search Space Bounds ±15%

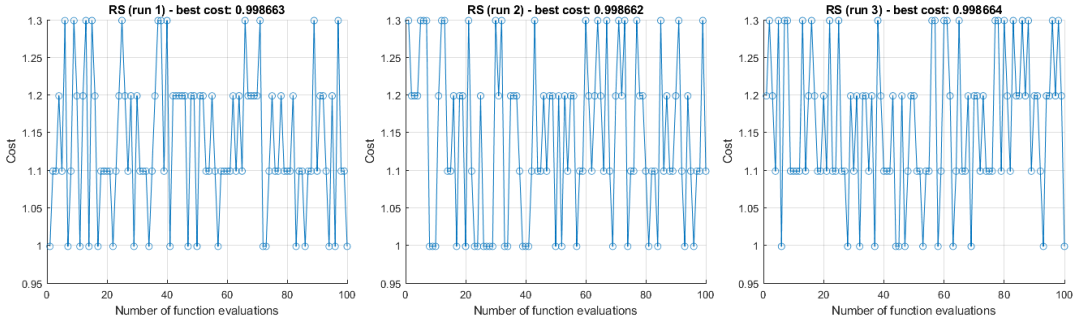


Figure B.1: Convergence of Random Search (8D, ±15% bounds).

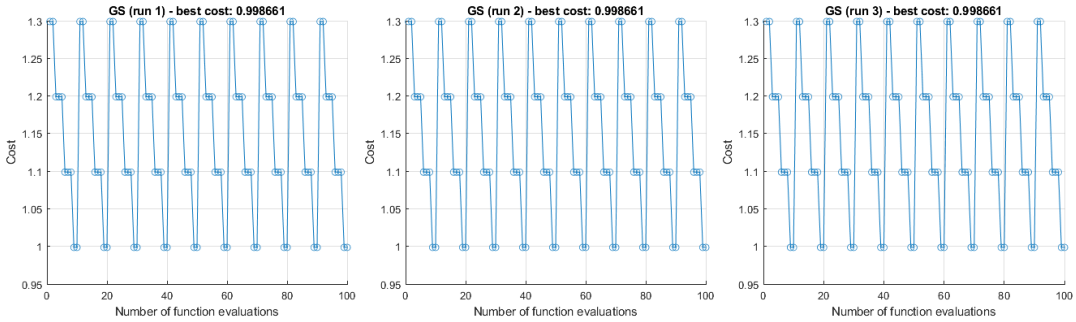


Figure B.2: Convergence of Grid Search (8D, ±15% bounds).

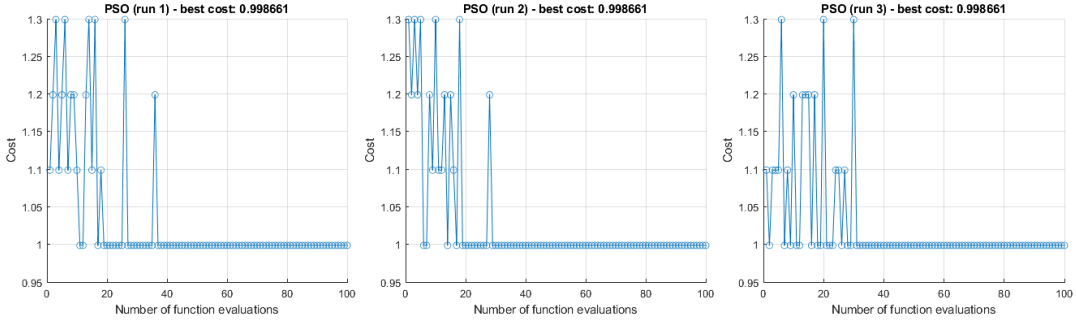


Figure B.3: Convergence of Particle Swarm Optimization (8D, ±15% bounds).

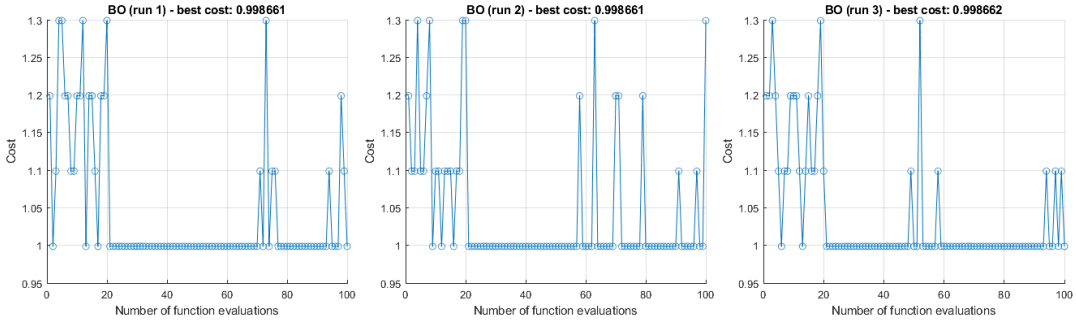


Figure B.4: Convergence of Bayesian Optimization (8D, ±15% bounds).

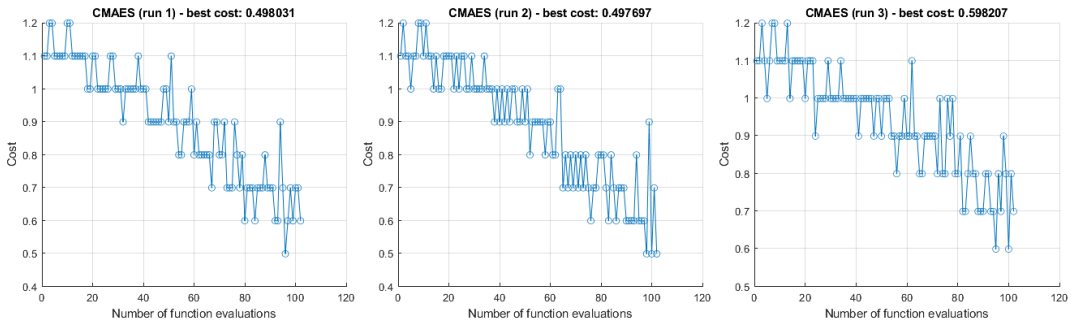


Figure B.5: Convergence of Covariance Matrix Adaption Evolution Strategy (8D, ±15% bounds).

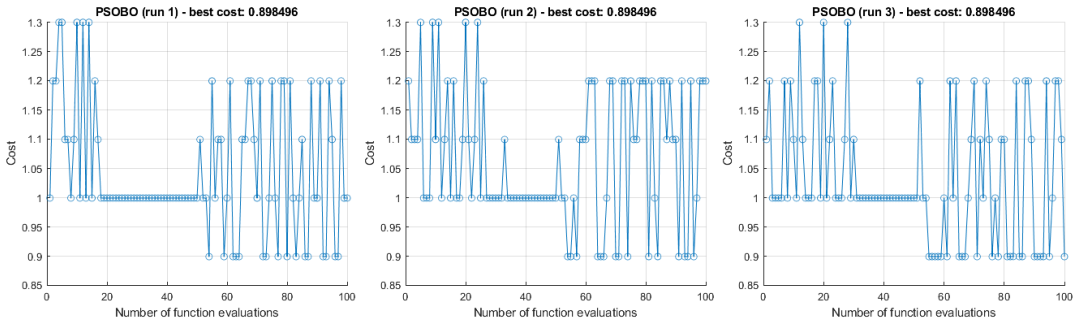


Figure B.6: Convergence of Particle Swarm Optimization Bayesian Optimization (8D, ±15% bounds).

B.1.2 Search Space Bounds ±30%

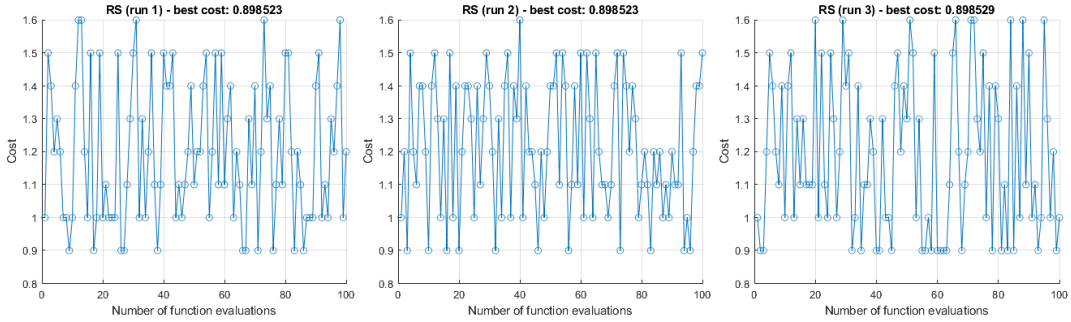


Figure B.7: Convergence of Random Search (8D, ±30% bounds).

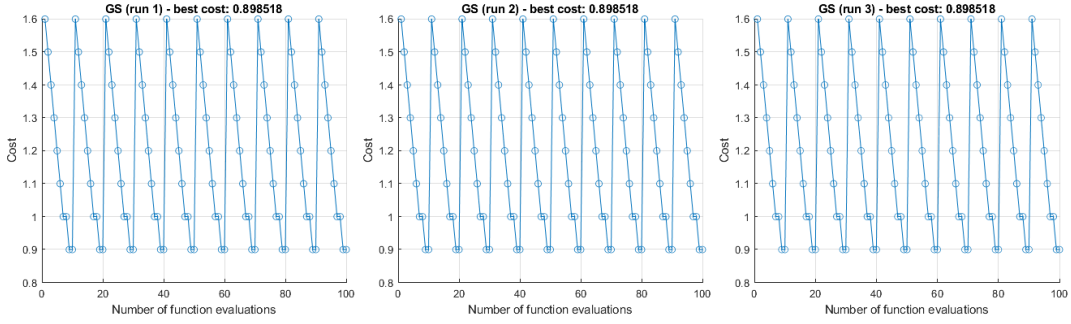


Figure B.8: Convergence of Grid Search (8D, ±30% bounds).

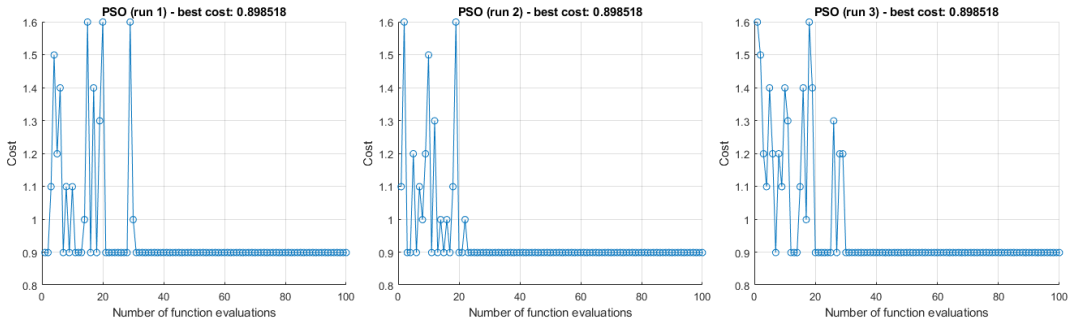


Figure B.9: Convergence of Particle Swarm Optimization (8D, ±30% bounds).

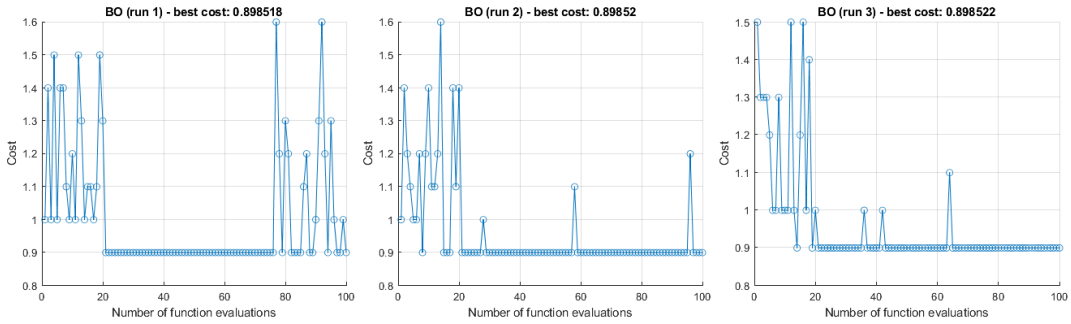


Figure B.10: Convergence of Bayesian Optimization (8D, ±30% bounds).

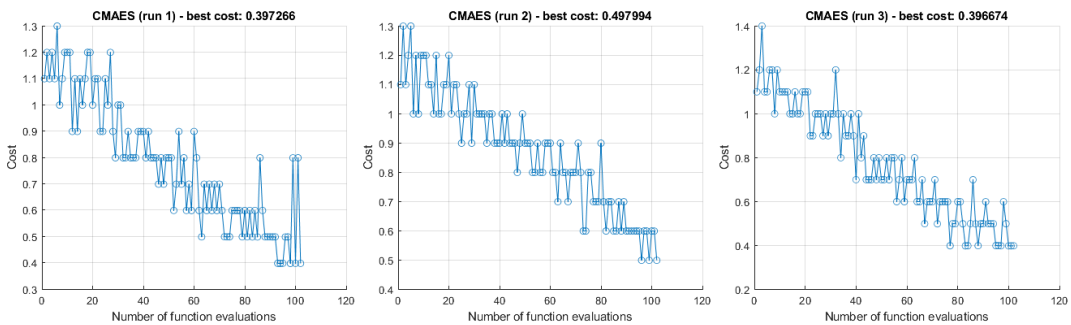


Figure B.11: Convergence of Covariance Matrix Adaption Evolution Strategy (8D, ±30% bounds).

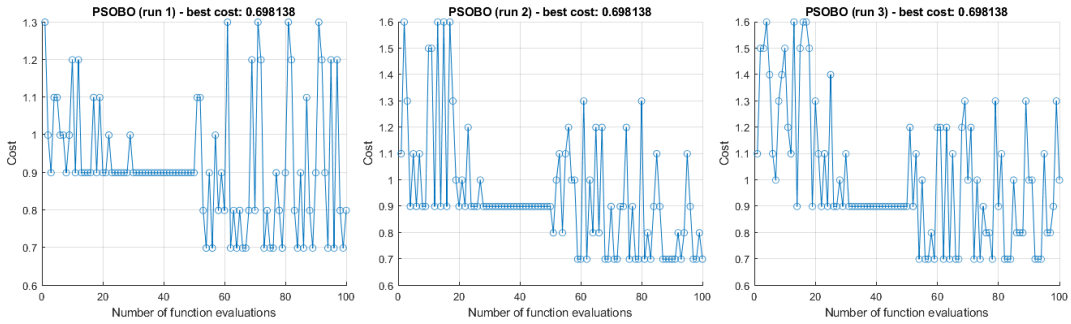


Figure B.12: Convergence of Particle Swarm Optimization Bayesian Optimization (8D, ±30% bounds).

B.2 Case Results with 26 Dimensions

.

B.2.1 Search Space Bounds ±15%

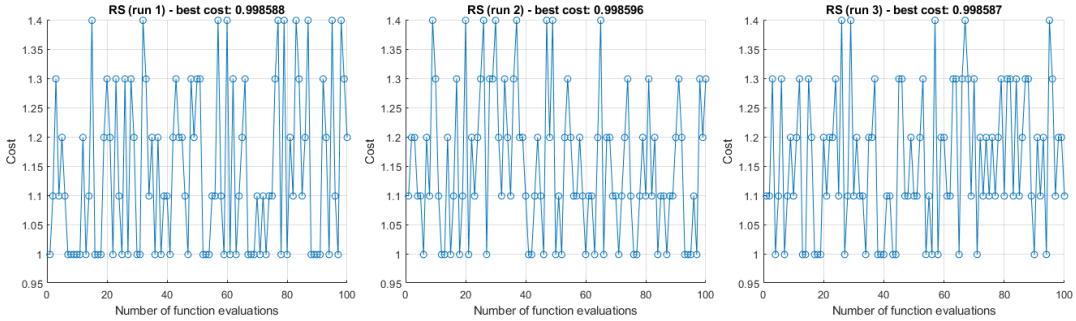


Figure B.13: Convergence of Random Search (26D, ±15% bounds).

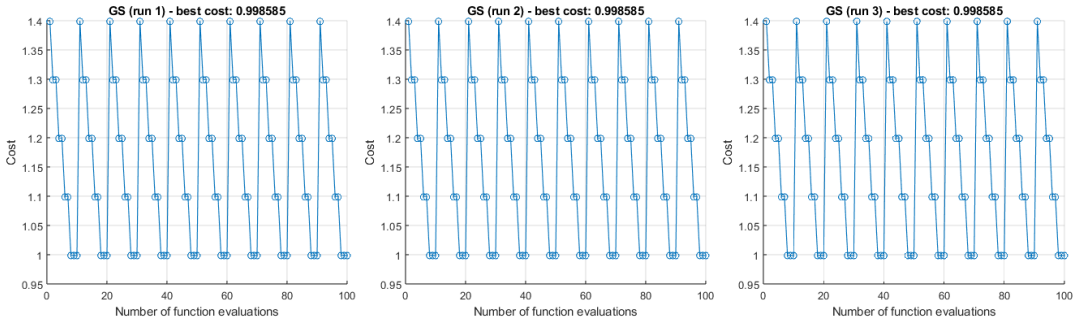


Figure B.14: Convergence of Grid Search (26D, ±15% bounds).

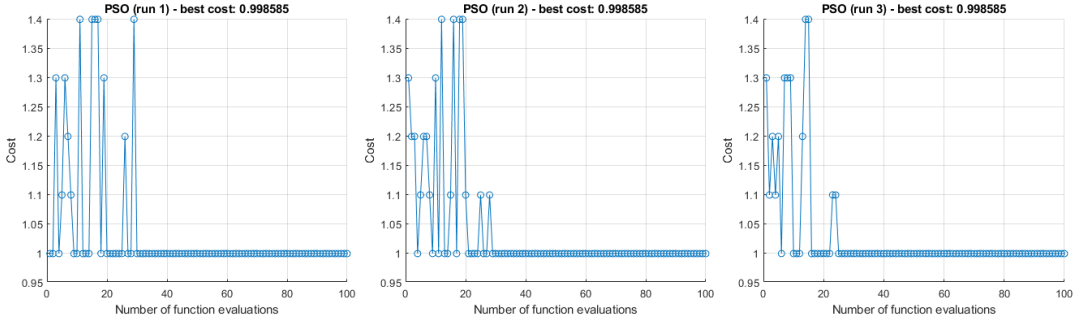


Figure B.15: Convergence of Particle Swarm Optimization (26D, ±15% bounds).

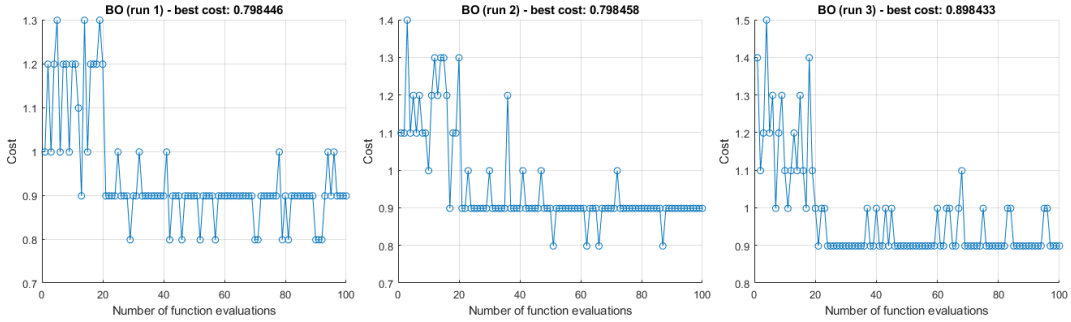


Figure B.16: Convergence of Bayesian Optimization (26D, ±15% bounds).

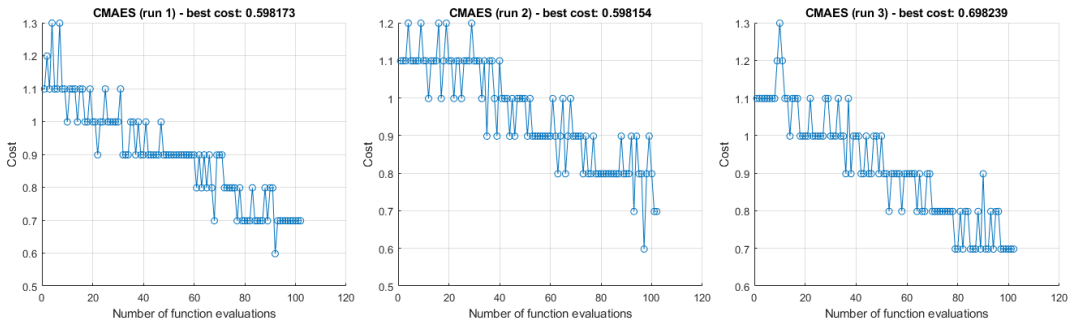


Figure B.17: Convergence of Covariance Matrix Adaption Evolution Strategy (26D, ±15% bounds).

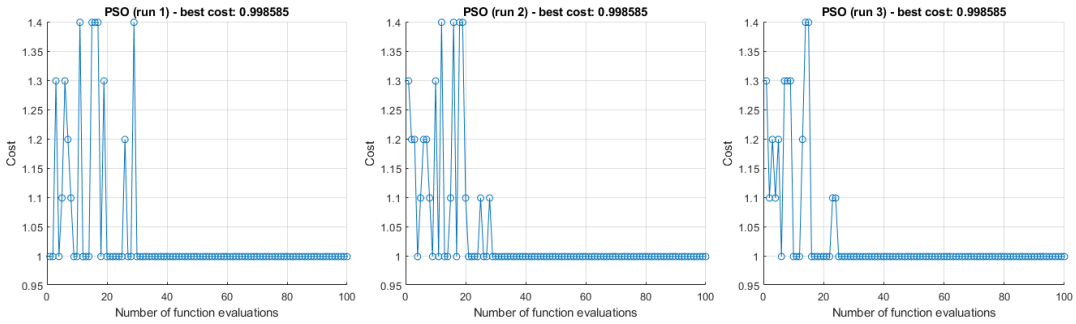


Figure B.18: Convergence of Particle Swarm Optimization Bayesian Optimization (26D, ±15% bounds).

B.2.2 Search Space Bounds ±30%

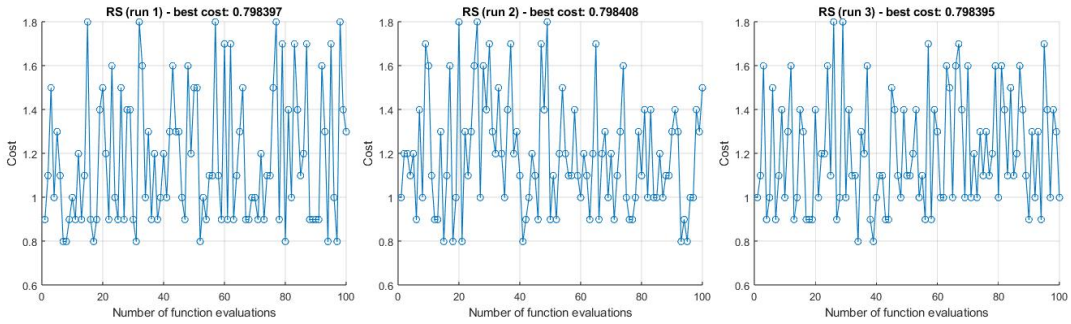


Figure B.19: Convergence of Random Search (26D, ±30% bounds).

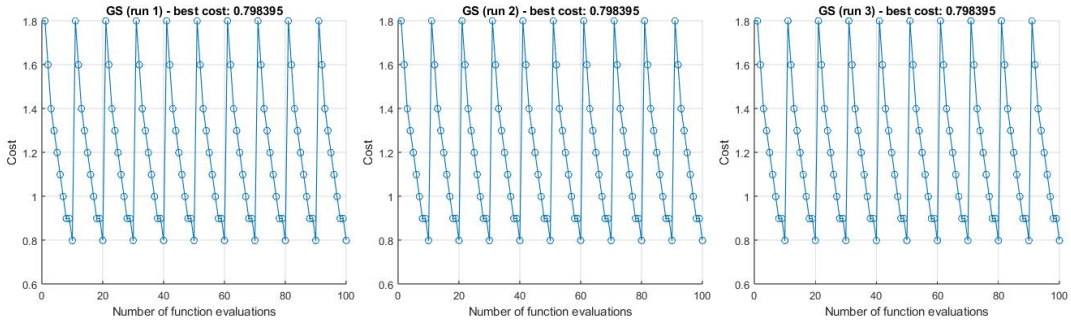


Figure B.20: Convergence of Grid Search (26D, ±30% bounds).

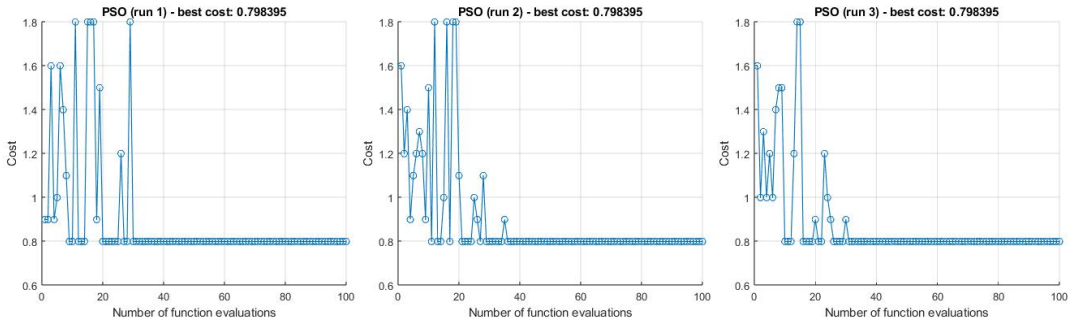


Figure B.21: Convergence of Particle Swarm Optimization (26D, ±30% bounds).

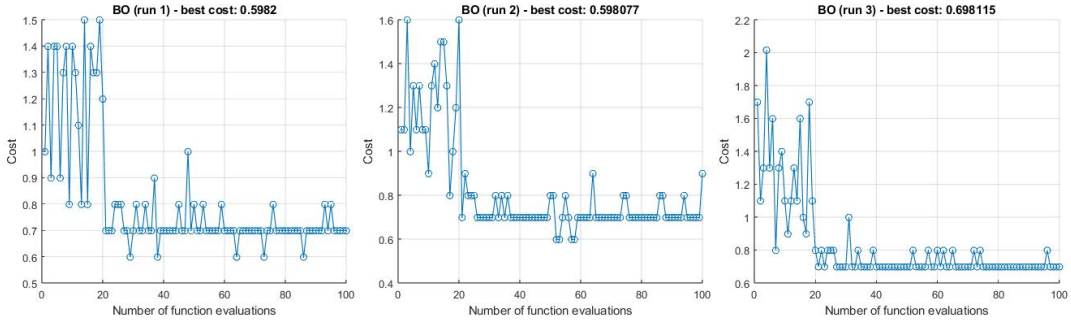


Figure B.22: Convergence of Bayesian Optimization (26D, ±30% bounds).

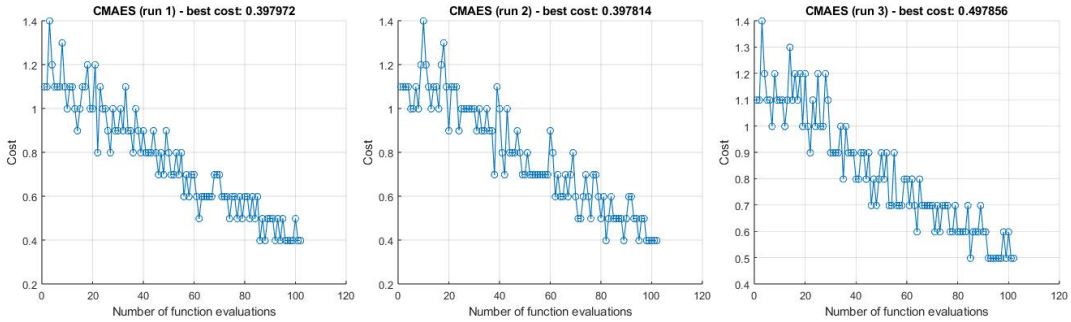


Figure B.23: Convergence of Covariance Matrix Adaption Evolution Strategy (26D, ±30% bounds).

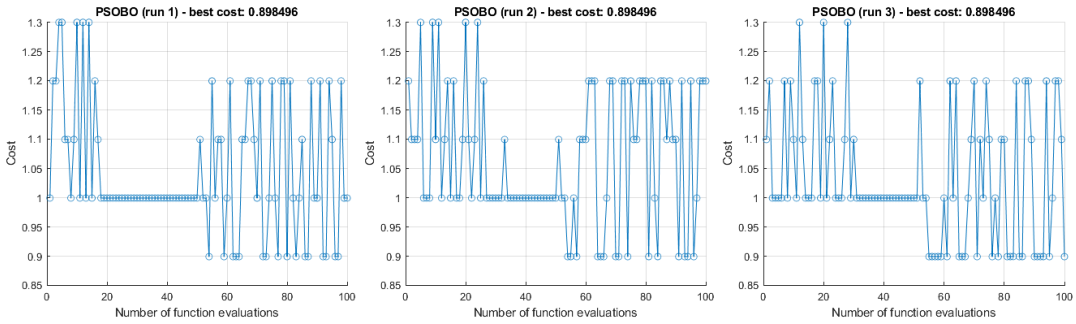


Figure B.24: Convergence of Particle Swarm Optimization Bayesian Optimization (26D, ±30% bounds).

B.3 Results of Case 4* (26D, ±30%)

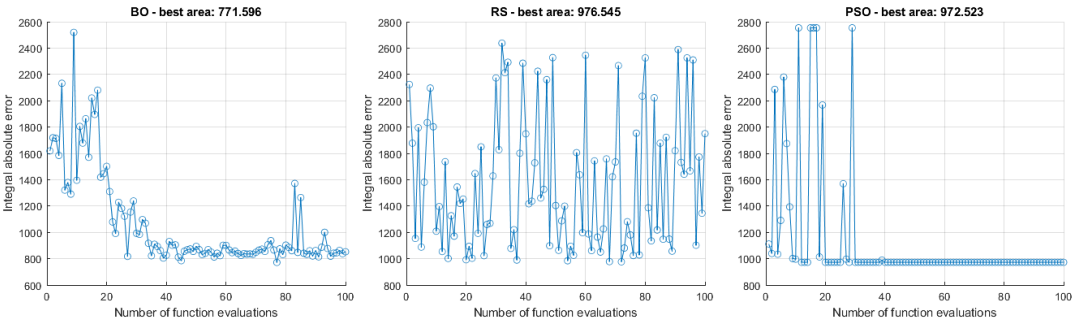


Figure B.25: Evaluations of BO, RS, and PSO.

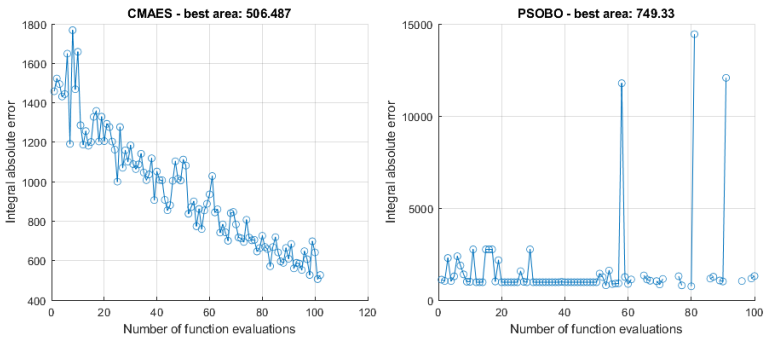


Figure B.26: Evaluations of CMA-ES and PSOBO.