
UNIVERSIDADE D
COIMBRA

Francisco Carvalho Tavares

# AUTOMATED PARAMETER TUNING FOR EVALUATION OF SLAM METHODS

## A RUST-BASED APPROACH

**Master's Dissertation in Electrical and Computer Engineering, supervised by Dr. David B. S. Portugal and Eng. Mário Cristóvão and presented to the Faculty of Science and Technology of the University of Coimbra**

February 2026


# Automated Parameter Tuning for Evaluation of SLAM Methods

## A Rust-based approach

**Supervisor:**

Prof. Dr. David Portugal

**Co-Supervisor:**

Eng. Mário Cristóvão

**Jury:**

Prof. Dr Rui Araújo

Prof. Dr. João Ruivo Paulo

Prof. Dr. David Portugal

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

February 2026

# Acknowledgments

To my supervisor, Dr. David Portugal, whose guidance and support were essential to my success. I also wish to thank Eng. Mário Cristóvão who, alongside Dr.Portugal, provided valuable feedback and insights that proved fundamental to the successful completion of this dissertation.

To my parents, thank you for your support and belief in me. Your presence and encouragement have meant the world to me.

# Resumo

A parametrização de soluções de Localização e Mapeamento Simultâneo (SLAM) tem potenciais benefícios para aplicações de robótica móvel em diversos ambientes. Algoritmos de optimização de parâmetros de sistemas SLAM podem ser ferramentas valiosas no estudo da influência dos parâmetros de um sistema SLAM, bem como na optimização da trajetória e redução de erro em aplicações de robótica móvel.

Este trabalho consiste na implementação e teste de três algoritmos de optimização de parâmetros para sistemas SLAM: Pesquisa em grelha (Grid Search), Pesquisa aleatória (Random Search) e *Simulated Annealing*. Os algoritmos foram escolhidos tendo em conta a diversidade de algoritmos estudados e o tempo disponível para a sua implementação. Os algoritmos podem ser usados para optimizar qualquer solução de Simultaneous Localization and Mapping (SLAM) disponível, embora os testes sejam feitos com Faster-LIO e Fast-LIVO2, duas soluções de SLAM muito populares na comunidade.

Neste documento é apresentada uma revisão do estado da arte, identificando as principais lacunas de pesquisa e fornecendo uma contribuição científica com o desenvolvimento deste trabalho. Os algoritmos desenvolvidos foram testados em duas soluções de SLAM. Os resultados mostram o potencial do *Simulated Annealing* em relação aos restantes algoritmos implementados, conseguindo em certos casos reduzir o erro da trajetória em mais de metade. Por último, são apresentadas sugestões para melhorias futuras.

**Palavras-Chave**: SLAM, Optimização de Hiperparâmetros, Faster-LIO, Fast-LIVO2, *Simulated Annealing*

# Abstract

The parameterization of Simultaneous Localization and Mapping (SLAM) solutions offers potential benefits for mobile robotics applications in a wide range of environments. Parameter optimisation algorithms for SLAM systems can be valuable tools for studying the influence of a SLAM system's parameters, as well as for trajectory optimization and error reduction in mobile robotics applications.

This work consists of the implementation and evaluation of three parameter optimisation algorithms for SLAM systems: Grid Search, Random Search, and Simulated Annealing. The algorithms were selected considering the diversity of optimization methods studied and the time available for their implementation. The proposed algorithms can be used to optimize any available SLAM solution; however, the experimental evaluation is conducted using Faster-LIO and Fast-LIVO2, two SLAM solutions that are widely used within the research community.

This document presents a review of the state of the art, identifying the main research gaps and providing a scientific contribution through the development of this work. The implemented algorithms were evaluated on two SLAM solutions. The results demonstrate the potential of *Simulated Annealing* when compared to the other implemented algorithms, achieving, in certain cases, a reduction of the trajectory error by more than half. Finally, suggestions for future improvements are presented.

**Keywords**: SLAM, Hyperparameter optimization, Faster-LIO, Fast-LIVO2, Simulated Annealing

*"He who only wishes and hopes does not interfere actively with the course of events and with the shaping of his own destiny."*

*Ludwig Von Mises*

# Contents

# List of Acronyms

| | |
|---|---|
| **APE** | Absolute Pose Error |
| **ATE** | Absolute Trajectory Error |
| **BO** | Bayesian Optimization |
| **DoG** | Derivative of Gaussian |
| **EA** | Evolutionary Algorithm |
| **EI** | Expected Improvement |
| **HB** | Hyperband |
| **HPO** | Hyperparameter Optimization |
| **HPT** | Hyperparameter Tuning |
| **ICP** | Iterative Closest Point |
| **IMU** | Inertial Measurement Unit |
| **LiDAR** | Light Detection and Ranging |
| **LoG** | Laplacian of Gaussian |
| **NDT** | Normal Distributions Transform |
| **PI** | Probability of Improvement |
| **PSO** | Particle Swarm Optimization |
| **RANSAC** | Random Sample Consensus |
| **RMSE** | Root Mean Square Error |
| **RPE** | Relative Pose Error |

| | |
|---|---|
| **RUSTLE** | Reliable User-friendly and Straightforward Tool for Localization Experiments |
| **SA** | Simulated Annealing |
| **SH** | Successive Halving |
| **SMBO** | Sequential Model-Based Optimization |
| **SLAM** | Simultaneous Localization and Mapping |
| **ULCB** | Upper/Lower Confidence Bound |
| **VO** | Visual Odometry |

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Context and Motivation

SLAM is one of the most studied topics in robotics [12, 13]. Its purpose is to simultaneously estimate the robot's position and orientation(pose) and map the robot's environment. SLAM methods often integrate data from devices like Light Detection and Ranging (LiDAR), cameras, or ultrasonic sensors with algorithms like Kalman filters and particle filters, or more advanced and recent approaches, such as Visual [14] and LiDAR SLAM [15]. These methods are crucial for applications in Robotics, such as autonomous navigation.

The accuracy of a SLAM method is usually evaluated using the Absolute Pose Error (APE) and the Relative Pose Error (RPE). The APE refers to the absolute difference between the estimated pose and the ground truth, while the RPE refers to the difference between consecutive estimated poses and the respective ground truth of consecutive poses. SLAM's efficacy is dependent on various factors, such as sensor quality, available computational resources, loop closure and algorithm type. The algorithm type is particularly important because different scenarios demand different types of SLAM methods. On the other hand, different SLAM methods have different number of hyperparameters, impacting the size of the parameter space and the difficulty of arriving at the optimal hyperparameter configuration for a given scenario.

One key challenge in SLAM research is determining the best hyperparameter configuration for a given dataset. Our research group has been developing named RUSTLE that streamlines a process, allowing different SLAM methods to be run asynchronously, while giving performance reports, allowing for the manual tuning of SLAM's hyperparameters.

There is, however, a missing and crucial component from frameworks like RUSTLE: in the literature, SLAM methods are usually not optimally tuned, or the comparisons are performed between methods that are close to optimal tuning and methods that are tuned merely to achieve

sufficiently satisfactory results, and therefore any comparisons cannot be considered fair and conclusive. Applying hyperparameter optimization algorithms and automating the tuning process would not only relieve the user of the laborious process of manually searching the parameter space for an optimal configuration, but also allow for a more fair comparison between SLAM methods.

## 1.2 Main Goals

The main goals of this dissertation are:

- Development of an automatic HPT framework within RUSTLE to better optimize and compare the performance of multiple SLAM solutions.
- Assessment of the impact of different hyperparameters have on several SLAM methods.
- Evaluation of the efficiency of the developed optimization algorithms and compare them to each other.
- Consolidated overview of the developed work and identify lessons learned and potential future improvements.

## 1.3 Document Overview

The remainder of this dissertation is organized as follows. In chapter 2, essential background concepts and related works are outlined. In chapter 3, a description of the design and implementation of the tuning module developed for RUSTLE is provided. In chapter 4, presents the tests results, as well as a brief discussion of those results. In chapter 5, the work is briefly summarized, highlighting its successes and weaknesses, while suggesting avenues for future improvements.

# 2 Background and Related Work

This chapter presents the topic of SLAM, its subtypes and metrics, as well as the topic of hyperparameter optimization and its specific relevance in the context of SLAM optimization. Finally, related relevant work is discussed, so as to provide a context and a starting point for the work developed in this dissertation.

## 2.1 SLAM

SLAM is a fundamental problem in robotics and computer vision, wherein a robotic system moves around in an unknown environment, and builds a map of said environment while simultaneously determining its own pose within said map [16].

SLAM systems use sensors such as LiDAR's, cameras, IMU's and GPS to collect data about the environment, which then is processed by the backend [16], which is implementation dependent.

Generally speaking, a SLAM method involves the following steps (not necessarily in this order):

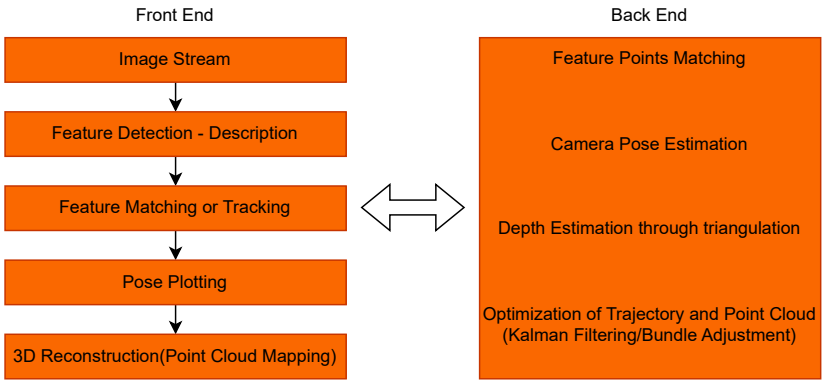


Figure 2.1: Common steps of a generic Visual SLAM system [1].

Firstly, sensor data is collected from the different onboard sensors, such as cameras, LiDARs, Inertial Measurement Unit (IMU)'s, etc. Given that most systems have sensors that collect data at different frequencies, it is critical for the data to be time-aligned. All data is timestamped relative to a common clock (system clock). This synchronization step ensures that features, poses and inertial readings from multiple sensors correspond to the same physical moment [17].

Next, features are detected. In the case of Visual SLAM, this tipically mean detecting corners, which are often present in more than one frame, allowing for later matching of the same 3D points and for the estimation of the pose of the robotic system. Popular choices for corner detectors are the Harris detector [18] and the blob detector [19, 20].

Feature matching is simply the process of searching for pairs of feature descriptors across different image frames that likely correspond to the same 3D scene point [21].

Pose plotting is the process of estimating and visualizing the pose of a moving camera or robot system within a global coordinate frame over time. Each pose represents a snapshot of where the sensor was located and how it was oriented when an image or measurement was captured. By connecting these poses, one obtains a trajectory that shows the motion path through the environment. Accurate pose plots are essential for evaluating localization performance, identifying drift, and debugging SLAM algorithms. Pose plotting relies on solving geometric constraints between observed image features across frames, often using techniques such as Perspective-n-Point (PnP), bundle adjustment, or pose-graph optimization [22].

Lastly, 3D reconstruction focuses on building a geometric model of the environment using the estimated poses and visual data. Once camera poses are known, corresponding image features can be triangulated into 3D points to form a map, initially a sparse point cloud and, with additional processing, a dense point cloud [23]. This reconstructed structure represents the physical world captured by the SLAM system. High quality 3D reconstruction depends directly on reliable pose estimates, i.e., errors in trajectory estimation propagate into geometric distortions in the map [23, 24].

### 2.1.1 Types of SLAM solutions

If one were to categorize SLAM solutions using the sensor types as a criteria, three broad categories would emerge: Visual SLAM(VSLAM), LiDAR SLAM and multi sensor SLAM.

### Visual SLAM

Visual SLAM uses cameras to understand an environment by detecting and tracking features over time [25]. Common subtypes of Visual SLAM are (according to the type of camera they use): Monocular SLAM, which uses only one camera, Stereo SLAM, which uses 2 cameras, separated by a known baseline, and RGB-D SLAM, which uses cameras that in addition to visual and color information, also provide pixel depth data (distance from camera to an object).

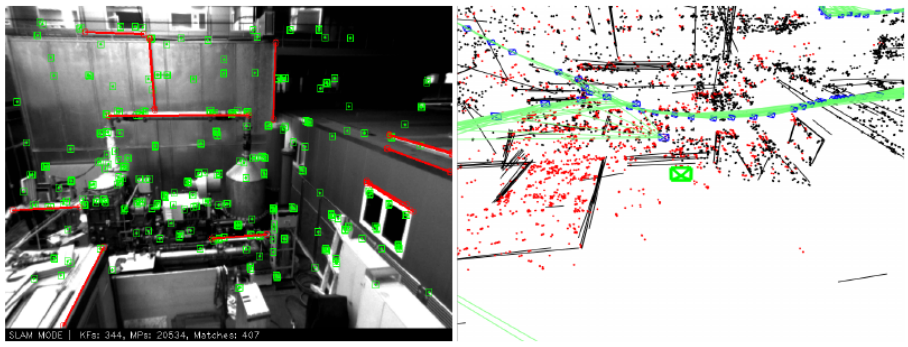


Figure 2.2: Visual SLAM feature detection and tracking (left) and 3D reconstruction (right) [2].

In terms of hyperparameters, let us consider a few of the most important ones and their effect on the performance of the visual SLAM system.

When performing feature detection and matching, a few important parameters are: **feature matching threshold**, **feature detection sensitivity** and the **scale factor** (in the case of pyramid-based detectors). The **feature matching threshold** determines how similar two feature descriptors must be to be considered a valid correspondence [26]. It directly affects how the system associates points between frames or keyframes. A high threshold ensures that only highly similar descriptors are matched [26], which increases precision but risks losing correspondences when illumination or viewpoint changes. A low threshold increases the number of matches but allows more outliers, which can corrupt motion estimation [26]. The **feature detection sensitivity** defines how easily the detector identifies keypoints in an image, usually based on a response strength or cornerness threshold. A higher sensitivity results in many detected features, improving robustness in textured environments, but it adds computational cost and potential noise. A lower sensitivity limits detections to the strongest points, making tracking faster but potentially unstable in low-texture areas. Lastly, the **scale factor** (only applicable for pyramid-based detectors) determine how much each image pyramid level is downsampled relative to the previous level [19]. A smaller scale factor adds more levels, improving the ability of the system to handle zooming and depth, but at the cost of aditional computational cost [19]. A higher

scale factor reduces the robustness to scale variations but speeds up feature extraction [19]. In short, this parameter controls a trade-off between real time performance and scale invariant tracking.

On the topic of pose estimation, Random Sample Consensus (RANSAC) is a particularly important algorithm. Specifically, the **maximum number of iterations** and the **minimum number of data points** to fit the model are important parameters. The **maximum number of iterations** specifies how many hypotheses RANSAC should run when estimating the camera pose. A higher value increases the probability of finding a correct model, even with many outliers, but adds computational cost [27], while a lower value is too permissive when accepting points into the model, and can lead to unreliable pose estimates [27]. This is a trade-off between robustness to outliers and real time performance. The **minimum number of data points** (or minimum inlier count) controls how many features correpondences are required to accept a pose estimation as valid. A low value makes the system acccept unstable or incorrect poses, resulting in drift [27], while a high value might make tracking unfeasible, when very few feature matches are available [27].

Finally, on the topic of loop closure, the **loop detection threshold** is one of the most important parameters, which determines how similar two keyframes must be (using a bag of words similarity score, for instance) [28]. A high threshold only triggers loop closure on highly confident matches. This prevents some false positives but might miss some oportunities to correct drift [28]. A low threshold increases loop closure sensitivity, but can incorrectly detect loops that might corrupt the map [28].

It should be noted that these parameters are only a fraction of the total number of parameters at play in visual SLAM solutions, and are only briefly mentioned here. Also, other, more *exotic*, less used methods might be used throughout the execution of the VSLAM algorithm, which requires a different set of hyperparameters that those presented previously.

### LiDAR SLAM

LiDAR SLAM uses, as the name suggests, LiDAR sensors, which unlike Visual SLAM, can measure distances directly using laser pulses. It is overall a more robust method for low light and featureless environments [29].

In terms of hyperparameters, a few of the most important hyperparameters often present in LiDAR based SLAM methods are: **voxel size**, **map resolution**, **maximum correspondence**

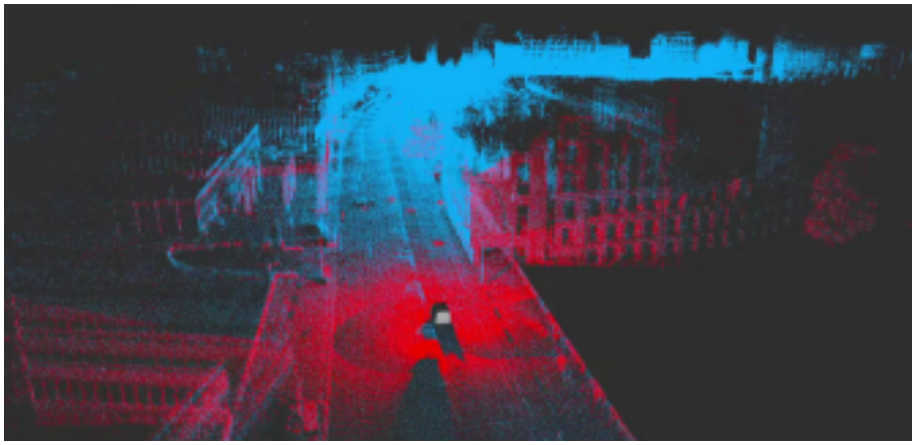

Figure 2.3: LiDAR SLAM point cloud example [3].

**distance** and **keyframe insertion threshold**. The **voxel size** determines how much the LiDAR point cloud is downsampled before registration. A smaller value helps to preserve detail and accuracy, but sharply increases computation time and noise sensitivity [30]. Larger voxels speed up the system and are more robust to noise, but might increase drift in the process [30]. The **map resolution** affects the granularity of the internal representation (voxel grid). A high resolution captures more fine-grained features but consumes more memory and processing time [31], while a coarse map runs faster but loses detail, affecting localization accuracy [31]. The **maximum correspondence distance** determines how far two points can be from each other to be considered a match during scan registration. If too small, the algorithm (Iterative Closest Point (ICP), Normal Distributions Transform (NDT)) may fail to produce enough correspondences, leading to unstable pose estimations [32]. On the other extreme, if the value set is too high, then incorrect matches will introduce more drift and distort the produced map [32]. The **keyframe insertion threshold** (based on distance, for example) controls how often new keyframes are created. Low thresholds yield many keyframes, increasing accuracy but slowing optimization [33], while high thresholds risk under sampling, introducing drift and lowering loop closure detection [33].

### Multi-Sensor SLAM

Multi-Sensor SLAM encompasses SLAM solutions which make use of different types of sensors, taking advantage of the strengths of each one, making the final map and trajectories more robust than if each sensor was used on an independent SLAM system.

A few common multi sensor SLAM solutions include: visual inertial SLAM [34], LiDAR inertial SLAM [35] and visual-lidar-inertial SLAM [36]. **Visual inertial SLAM** combines the

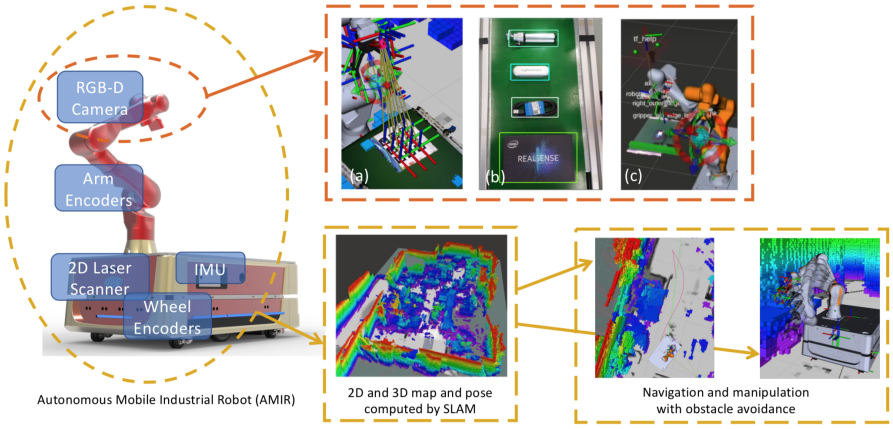


Figure 2.4: Multi-Sensor SLAM system for an industrial robot [4].

spatial information from a camera with the higher temporal resolution of an IMU to make up for the other sensor's weaknesses. When the IMU accumulates too much drift, the cameras help correct it, and the IMU helps under poor visual conditions. Its main disadvantage is the high sensitivity to visual degradation, such as fog and dark areas. **LiDAR-Inertial SLAM**'s main advantage is the high robustness in low light or outdoor environments and the resilience to visual noise, present in nightly, foggy and even dusty environments. Two main limitations are the more expensive/heavier setups and the limitation in texture representation (pure geometry only). In **Visual-Lidar-Inertial SLAM**, rich visual features, accurate range data and motion tracking are all present. It however requires a complex calibration and synchronization process, being more expensive and requiring more computational power than simpler multi sensor approaches to SLAM.

### 2.1.2 Evaluation of SLAM

Assessing SLAM performance requires quantitative and qualitative metrics that evaluate how accurate, robust and efficient the estimated map and trajectory are. A few evaluation criteria include the Absolute Trajectory Error (ATE), RPE, resource usage (memory and CPU time consumption), etc.

The **ATE** measures the global deviation between estimated and ground truth trajectories. At each point in the trajectory, the difference between the estimated pose and the ground truth pose is computed. For a more general measure of the ATE, one might also calculate the Root Mean Square Error (RMSE), as follows:

$$ATE_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|S(p_i^{est}) - p_i^{gt}\|^2}. \tag{2.1}$$

The **RPE** measures the local consistency of the trajectory by evaluating the difference in relative motion between estimated and ground truth poses over a fixed time interval. As with the ATE, the formula for the RMSE of the RPE is as follows:

$$RPE_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|(T_i^{est})^{-1}T_{i+\Delta t}^{est} - (T_i^{gt})^{-1}T_{i+\Delta t}^{gt}\|^2}. \tag{2.2}$$

It is important to take into account the system resource utilization, both the memory and CPU time consumption, when benchmarking and comparing different SLAM solutions, due to the trade-offs between accuracy/noise robustness and memory utilization/time consumption.

In addition to the metrics mentioned previously, one might want to optimize for other metrics not included on this list. In that case, it might be useful to use a fitness function where normalized weights are attributed to each metric, according to the importance of each metric.

Finally, when evaluating and comparing different SLAM solutions, it is important to establish a fair playing field for all the algorithms to be compared. One of the ways to do that is to use publicly available standardized datasets, such as the KITTI Dataset [37]. As for evaluation frameworks, one of the most widely used is EVO [38], which provides error metrics and visualization tools, so as to better compare the performance of different SLAM solutions.

## 2.2 Parameter optimization techniques

Parameter optimization techniques can be broadly categorized into search-based, model-based and population-based approaches. Each approach uses different strategies to search the parameter space and obtain an approximation of the optimal solution, such as randomly sampling configurations (random search), mimicking physical processes to get a faster convergence (Simulated Annealing) or even by pre-selecting a parameter space and dropping half of the worst performing configurations at each pass (successive halving). Some of these approaches require a budget to be defined, meaning a time limit, or a maximum number of configurations to be tested.

### 2.2.1 Search based approaches

One popular type of approach to the problem of Hyperparameter Optimization (HPO), which will be used as a baseline on this dissertation's work, is a search based approach, such as Grid Search and Random Search. These approaches are popular due to their implementation simplicity and paralelization possibilities.

#### Grid Search

Grid Search is a basic solution for HPO. It simply consists of an exhaustive search and evaluation of all possible hyperparameter combinations within a predefined parameter space [39]. In spite of the development of more sophisticated and specific algorithms in recent decades, Grid Search remains popular due to its simple implementation and trivial paralelization [39]. The main drawback is its computational cost, due to the curse of dimensionality being a serious problem in models with large numbers of hyperparameters [39]. This might be summarized by the following equation:

$$N_c = \prod_{n=1}^{k} N_{P_i}, \tag{2.3}$$

where $N_c$ is the total number of configurations and $N_{P_i}$ is the number of possible values for hyperparameter i of the model.

Inspection of Equation 2.3 reveals that this algorithm does not scale efficiently, due to the rapid increase in the number of configurations, which makes this the main hurdle in search based approaches. One way to overcome this is by paralelizing the execution of various configurations across several CPU cores and threads. Another way is to pre-select the parameters to optimize, and/or discarding hyperparameters which have little effect on the model's performance. However, this strategy has its downsides. If the initial analysis of the relevant parameters is inaccurate, parameters that are significant in certain regions of the parameter space may be prematurely discarded [40]. Additionally, parameters that individually have small effects on the output metric may have greater effects when combined [41].

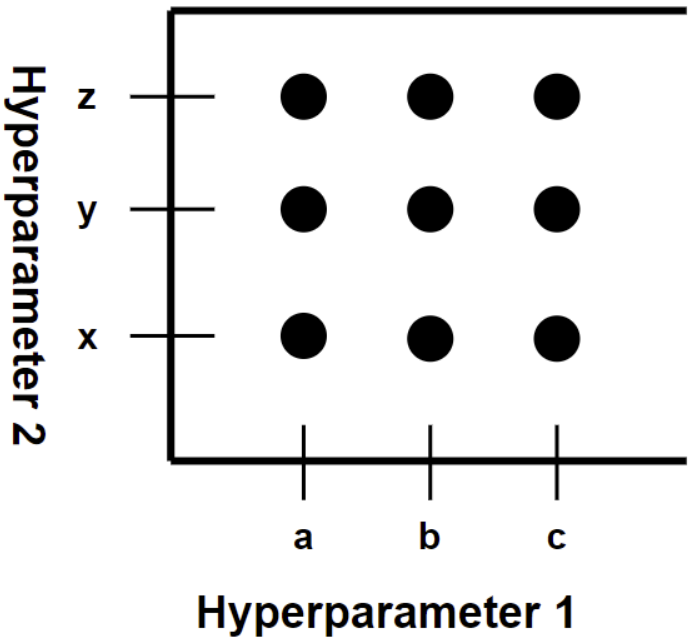


Figure 2.5: Graphical illustration of parameter space exploration in grid search [5].

### Random Search

Random Search is a variation of Grid Search. It randomly samples configurations in the aforementioned parameter space [42]. Both Grid Search and Random Search are very similar implementation wise, with one major difference: Random Search requires a budget to be specified, whether it is time, number of configurations, etc [40]. Compared to Grid Search, Random Search offers faster convergence toward local or global optima [40]; however, this advantage becomes less pronounced as the parameter space grows.

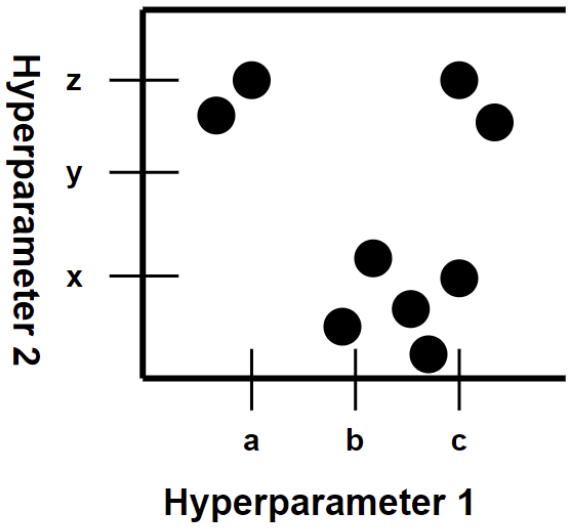


Figure 2.6: Graphical illustration of parameter space exploration in random search [5].

### 2.2.2 Model based approaches

Model based approaches tackle the optimization problem in a different way, by building a surrogate model that describes the relationship between hyperparameter configurations and algorithm performance. The inner workings of the algorithm to be optimized are unspecified and it is therefore treated as a black box [43]. These kind of HPO techniques are better suited to more complex optimization problems and clarify the relationship between algorithm performance and hyperparameter settings, which might prove to be an appropriate option to pre-select the most important hyperparameters to optimize when there are dozens or hundreds of parameters.

#### Bayesian Optimization

Bayesian Optimization (BO) is a probabilistic model-based approach that optimizes black box functions that are expensive to evaluate [44]. It is particularly useful when the objetive function lacks an analytic expression and its evaluation is expensive, which is the case for SLAM methods.

**Algorithm 1** Bayesian Optimization

1: **for** $t = 1, 2, \ldots$ **do**

2: Find $x_t$ by optimizing the acquisition function over the GP:

$$x_t = \arg\max_x u(x \mid \mathcal{D}_{1:t-1})$$

3: Sample the objective function: $y_t = f(x_t) + \epsilon_t$

4: Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (x_t, y_t)\}$ and update the GP.

5: **end for**

Figure 2.7: Bayesian Optimization algorithm steps [6].

Bayesian Optimization has two main components:

- A Surrogate model, which is a probabilistic, fast proximate model that stands in for, and is much easier to evaluate than, an unknown(black box) or expensive function [45]. Instead of repeatedly evaluating the true function, which might take a long time, the surrogate model (usually a Gaussian Process [42]) learns from previous evaluations, and predicts both the expected mean value of the true function (in this case, the RMSE of the RPE of a SLAM solution) and the uncertainty at each point. The uncertainty is what informs

and guides the algorithm on what point to sample next, by balancing promising regions with unexplored ones (higher uncertainty).

- An Acquisition function, which is the rule BO uses to choose the next objective function point to evaluate [45]. It uses the surrogate model's predictions (mean and uncertainty) and combines them into a single score, which is then used to evaluate the best candidate point. Common Acquisition functions include Probability of Improvement (PI), which focuses on the **chance** of beating the current best result, and Upper/Lower Confidence Bound (ULCB), which explicitly trades-off mean and uncertainty using a tunable hyper-parameter [46].

BO uses the following iterative process to optimize the objetive function (see Figure 3.4):

1. Select the next point to be sampled with the aquisition function.
2. Evaluate the true function $f(x_1, x_2...x_n)$ at the selected point.
3. Update the surrogate model with the new information using gaussian process regression, which is the process of adding additional information of the sampled points to the **prior**.
4. Repeat steps 1, 2 and 3 until some stopping criteria is met (exhausted budget, achieved convergence, etc).

One major drawback of Bayesian Optimization is the impossibility of paralelization when compared to other baseline techniques, due to the fact the the surrogate model uses new points to update its parameters, meaning the learning process needs to finish before a new one can be launched [42].

### 2.2.3 Population based Approaches

These types of approaches are defined by a population of candidate solutions (sets of hyper-parameters) that are iteratively updated to optimize an objetive function [47], such as the APE or the RPE in the case of SLAM optimization.

#### Simulated Annealing

Simulated Annealing (SA) is a meta-heuristic that mimicks the physical process of heating a metal and then cooling it slowly [48]. Analogously, the algorithm freely explores solutions in the beginning, even ones that seem worse at face value, so as to maximize exploration, and then, as

the temperature decreases, according to a predefined cooling schedule, it focuses on refining a solution [49, 48].

The method starts by assigning an initial value to each hyperparameter, one that is high enough to allow compreensive search over the parameter space [49], due to the high variance of the parameter values in early stages of the algorithm. Then, it makes small changes to each parameter at a time and evaluates this new configuration, called a neighbor [48]. The acceptance of the neighbor as being a superior solution depends on a probabilistic distribution [48], which in itself depends on the temperature and the difference between the current solution's and the neighbor solution's evaluation, like so:

$$P = e^{-\frac{\Delta E}{T}}, \tag{2.4}$$

where T is the current temperature and $\Delta E$ is the difference between the cost of the new solution (the neighbor's) and the cost of the current solution, that is, $\Delta E = f'(x) - f(x)$.

Then, the algorithm updates the temperature, using the following expression: $T' = \alpha T$, where $\alpha$ is the cooling factor, which is usually given a value in the interval [0.8, 0.99].

The algorithm repeats the previous steps until the temperatures reaches a minimum value or a stopping condition is triggered, such as a maximum number of iterations [49]. Once execution stops, the best solution is returned as an approximation to the actual optimal solution. Figure 2.8 shows a graphic representation of SA's behavior.

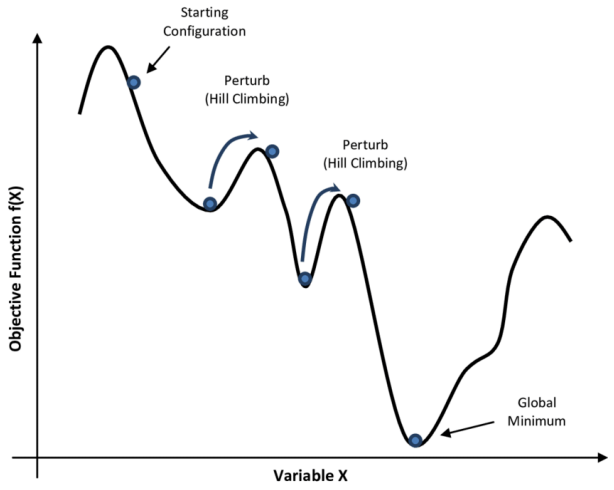


Figure 2.8: A graphic representation of the simulated annealing algorithm [7].

### Successive Halving

Successive Halving (SH) is an optimization method that efficiently allocates resources to the most promising configurations while reducing investment into less promising ones [50]. Similar to SA, this technique requires a budget to be specified (execution time, number of iterations, etc.).

At first, $n$ candidates are generated and evaluated, and the constrained resources are assigned equally to all configurations. Then, **the worst half of all configurations is discarded** and this process is repeated until only 1 configuration remains [51]. The hyperparameter configurations used in this method can be generated in a variety of different ways. A popular one is identical to the way parameter spaces are generated in Grid/Random Search algorithms, where the researcher manually selects the parameters to optimize, as well the their upper and lower bounds, as well as the step size.

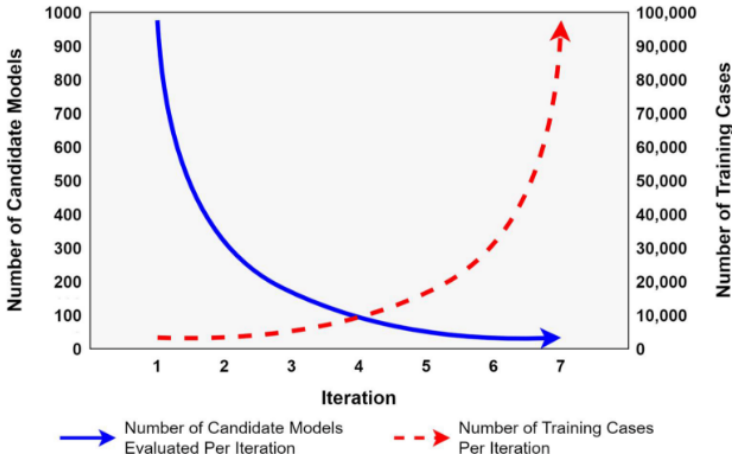


Figure 2.9: A graphic representation of the Successive Halving algorithm [8].

### Hyperband

Hyperband is a more efficient technique that builds on the Successive Halving algorithm [45]. By dynamically allocating computational resources, Hyperband quickly identifies promising configurations and discards ill-performing ones early on, saving both time and computational resources.

Hyperband works as follows:

1. Define a total Budget B and the proportion of configurations to evaluate at each iteration, $\eta$. Also, a maximum budget R for a single iteration is required [52].

2. Generate a large number of configurations [52].

3. Allocate a small initial budget to all configurations and perform the successive halving algorithm with one modification: halt its execution when only the top 1 / $\eta$ configurations remain, instead of halting when only the best configuration remains. Then, increase the budget for the remaining configurations [52].

4. Repeat step 3 until maximum budget R is reached or all configurations are exhausted, meaning only the best one remains [52].

By increasing the budget allocated to a decreasing number of configurations at each execution of the Successive Halving algorithm, Hyperband transitions from an exploration-focused method to an exploitation-focused one.

**Algorithm 1:** HYPERBAND algorithm for hyperparameter optimization.

**input** : $R$, $\eta$ (default $\eta = 3$)
**initialization** : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$
**for** $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with $(n, r)$ inner loop
  $T =$ `get_hyperparameter_configuration`$(n)$
  **for** $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  **end**
**end**
**return** *Configuration with the smallest intermediate loss seen so far.*

Figure 2.10: Hyperband described in a more systematic way [9].

### 2.2.4 HPT in SLAM

When optimizing the parameters of a SLAM method, several considerations must be taken into account.

First, not all parameters are amenable to tuning. In practice, some parameters have fixed or default values and are treated as constants throughout the tuning process, such as camera intrinsic parameters in visual SLAM, or the noise and bias models of an IMU in Visual–Inertial SLAM. This distinction is particularly important when employing search-based approaches, as excluding non-tunable parameters can substantially reduce the size of the search space and, consequently, the number of configurations that must be evaluated.

With respect to the effectiveness of different tuning strategies, search-based methods are typically regarded as baseline approaches in the literature and tend to perform poorly in high-dimensional parameter spaces due to the inherent stochasticity of Grid and Random Search. In contrast, Simulated Annealing generally yields improved results by incorporating both local refinement and controlled exploration of the search space. However, achieving near-optimal solutions in a computationally efficient manner often requires more advanced techniques. Methods such as Successive Halving and Hyperband are well suited to constrained environments, where computational resources or time are limited. Bayesian Optimization further improves upon these approaches in terms of sample efficiency, as it typically requires fewer full evaluations of the objective function to identify high-performing configurations. Moreover, Bayesian Optimization is particularly effective when objective function evaluations are expensive or when the parameter space is low dimensional, which may be the case in some SLAM contexts, provided that the parameters to be optimized are carefully selected by the researcher.

## 2.3 Related Work

This subsection discusses published research in optimizing parameters in SLAM systems. It presents a brief description of each relevant article and the used SLAM method and optimizing strategy, as well as the results from the experiments that were made. It then goes over the current literature research gaps and presents the main contributions of this dissertation.

On search-based approaches, there has been some research on the topic. Particularly, Putra et al. have hand-selected and manually searched (brute force) the parameter space of a G-Mapping SLAM solution [53]. In this specific case, four parameters were tuned: **linear update step**, **angular update step**, **quantity of particle**, and **resampling threshold**. The particle number directly determines the number of possible robot position hypotheses the algorithm tracks. The linear update step is the minimum distance (in meters) the robot must move before updating the map. Similarly, the angular update step is the minimum angle the robot must rotate before triggering a map update. Finally, the resampling threshold controls the frequency at which particles are resampled. Through several executions of the algorithm, it was possible to reduce the navigation time of a robot from 32 seconds to 25 seconds.

Another study applied grid search to optimize 5 parameters in a feature based monocular visual odometry setting [54]. In all 10 sequences, measured ATE significantly decreased, showing the main advantage of this simple tuning approach. Naturally, the optimal configuration wasn't

the same in all sequences, showing that a parameter's optimal value depends on the dataset for which it is optimized, in order to find a balanced setup, the authors chose parameter values that consistently produced reduced ATE values across all sequences. It is also worth noting that while search based approaches are more suited for a parameter set, Grid Search is generally preferable to manual (Brute-force) search. Additionally, both previously discussed articles indicate that, given knowledge of the system, the parameter space may be reduced to a limited set of influential parameters, allowing Grid and Random Search to yield configurations that improve upon default settings.

As far as model based approaches go, BO is the most used algorithm in research. One paper used a variant of it, Sequential Model-Based Optimization (SMBO) to optimize the hyperparameters of a LiDAR-based Odometry algorithm, without knowledge of the inner workings of the system, e.g the system is treated as a black box [55]. Although data augmentation was used to prevent overfitting (by adding random noise to the point clouds, reversing the data order, and applying a random transformation to the point cloud), it was still possible to observe a noticeable decrease in the odometry drift error (in most tested sequences). Although beyond the scope of this dissertation (in which camera parameters are treated as constants), another work applied Bayesian Optimization to optimize the extrinsic parameters of a camera in a Visual-Inertial SLAM system [56].

With regards to population-based approaches, relatively little research has been conducted specifically in the context of SLAM. However, a study by Kostusiak and Skrzypczyński [47] used a Particle Swarm Optimization (PSO) algorithm and an Evolutionary Algorithm (EA) to tune the parameters of an RGB-D Visual Odometry system to improve motion estimation accuracy. Using the TUM RGB-D dataset for training and the PUT Kinect dataset for testing, the authors applied the two previously mentioned methods to adjust only five influential hyperparameters related to feature detection (AKAZE feature detector threshold) and outlier rejection (two RANSAC distance and inlier/outlier thresholds) [47]. The optimization, guided by the values of the ATE and RPE, showed that tuning these few parameters, particularly the AKAZE feature threshold and RANSAC distance limits, significantly reduced trajectory errors, with ATE dropping from about 1.6m to 0.29m [47]. While the Particle Swarm algorithm achieved the best accuracy out of the 2 tuning strategies, the Evolutionary Algorithm achieved similar results in one-fifth of the time [47]. Alternatively, the authors also manually tuned the parameters, with significantly inferior results than PSO [47].

| Article(s) | SLAM category | Optimization method(s) | Results |
|---|---|---|---|
| Z. Zheng (2020) [54] | Feature-based Visual Odometry (not full SLAM) | Grid Search | Across 8 optimized sequences, average ATE decreased, on average, 64.68% |
| A. Putra and P. Prajitno (2019) [53] | G-mapping SLAM (particle filter) | Brute Force | Navigation time of a predefined path was reduced from 32 to 25 seconds |
| K. Koide et al (2021) [55] | Lidar Odometry (not full SLAM) | based on Bayesian Optimization | **With data augmentation**, in both tested environments, drift error decreased by at least 9.8% |
| A. Kostusiak and P. Skrzypczyński (2019) [47] | RGB-D Visual Odometry | Particle Swarm Optimization (PSO) | RPE decreased by well over 50% |
| | | Evolutionary Algorithm (EA) | Similar performance to PSO, but tuning duration is 5x shorter |

Table 2.1: Summary of HPT methods used in the literature.

### 2.3.1 Literature Gaps

A few notable research gaps exist in the field of SLAM parameter tuning. Most notably:

- Lack of diversity in optimization strategies. As presented earlier, most of the research use either a search-based approach or a model-based approach (similar to BO), as well as only using one optimization method for their particular SLAM solution. When there is a comprehensive understanding of sensor behavior and the influence of parameters on SLAM performance, some parameters may be retained at their default values, allowing a straightforward search-based strategy to optimize the remaining parameters. As for the model-based approaches, they are used most when there is no extensive knowledge of the system at hand. Therefore, BO treats it as a black box and optimizes it. Apart from [47], research on population-based approaches for SLAM optimization remains scarce, whether variations of particle swarm algorithms, evolutionary algorithms, or other meta heuristics.

- Lack of frameworks to automatically optimize SLAM methods, allowing for an even playing field in SLAM research. Athough a framework like **SLAMBench2** [57] provides a platform with a controlled dataset and with standardized metrics, it lacks an automatic tuning feature. Similarly, **RUSTLE**[1], the tool which will be built and improved upon during the course of this dissertation, already allows for a somewhat organized approach to testing and optimizing complete SLAM solutions, but it lacks any optimization feature, meaning the only way to currently approach HPT in RUSTLE is by brute force optimizing an algorithm.

### 2.3.2 Statement of Contributions

- Implementation of several algorithms for tuning a SLAM system's parameters, namely Grid Search, Random Search, and Simulated Annealing

- Integration of the implemented tuning algorithms into the existing open source framework (RUSTLE) to allow for automated and streamlined tuning

- Validation and testing of the implemented tuning algorithms on two SLAM solutions

[1] `https://github.com/Forestry-Robotics-UC/rustle`

# 3 Methodology

This chapter presents the methodology behind the development and testing of the SLAM tuning module developed within RUSTLE. It begins by explaining what RUSTLE is and how it works and then explain how the tuning module fits in with RUSTLE and its codebase. Then, lower level details of each implemented tuning algorithm will be explained. Finally, a testing methodology will be introduced and properly explained.

## 3.1 RUSTLE

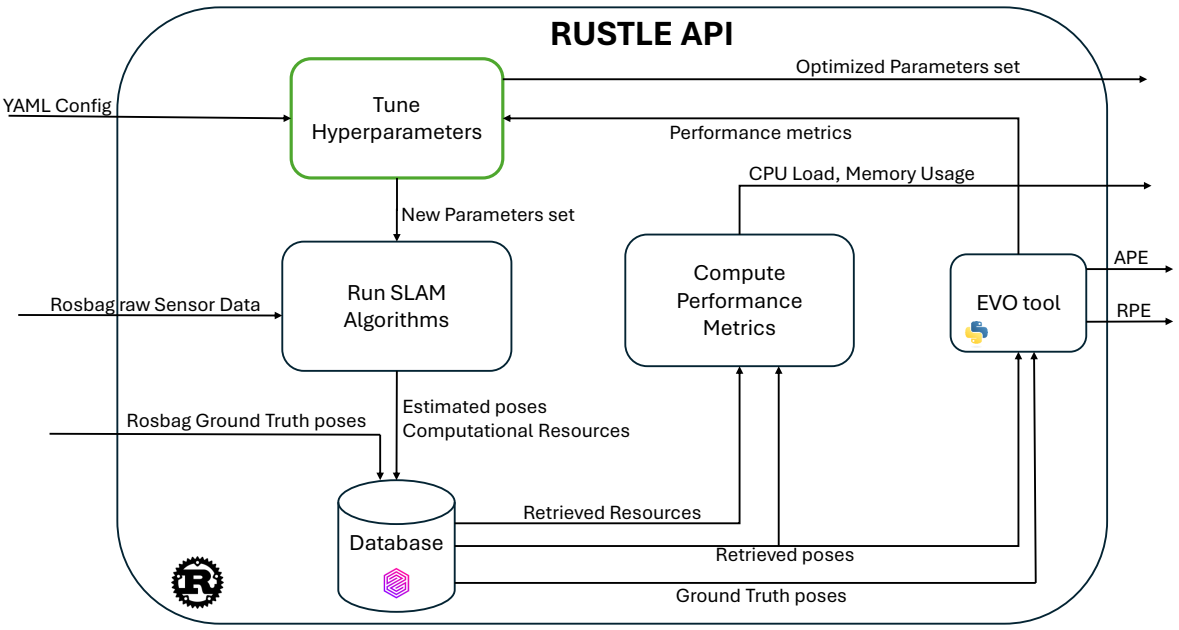


Figure 3.1: RUSTLE software architecture

RUSTLE is a command line application, written in Rust, which is designed to simplify the evaluation of SLAM algorithms in mobile robotics. Its main objective is to provide a simple, streamlined and reproducible way to run and compare SLAM algorithms. It tracks not only

trajectory accuracy (APE or RPE), but also runtime, CPU load and memory usage, offering a complete assessment of the algorithm's performance.

At its core, RUSTLE takes three main inputs: a dataset with raw sensor data in rosbag[1] format, the SLAM algorithm's parameters in yaml format, and the algorithms themselves as Docker[2] images. The reason for using Docker is for reliably reproducing the algorithms across many different platforms and environments. During execution, the estimated pose data is streamed into the database, which, after conclusion, is used to calculate trajectory errors such as the APE and the RPE, using EVO [38] (Figure 3.1).

Algorithm executions are run as tests. Each test's configuration is described in an YAML file, which contains: its name, number of workers, number of iterations, list of algorithms, dataset on which to execute these algorithms and the test type. Additionally, some additional fields might be required, depending on the type of test to be executed.

Tests can be of a few different types, namely:

- Simple - The rosbag file simply plays from begining to end at normal speed
- Speed - Portions of the dataset are run at different speeds
- Cut - Only specified portions of the dataset are run
- Drop - Simulates sensor dropout, by reducing sensor message rate over a specified section of the dataset

Every tuning configuration on RUSTLE will be run as a simple test, with the caveat that the user can choose the portion of the dataset to run. This is to facilitate development and testing.

## 3.2 Feature Implementation

In the previous chapter, several tuning approaches were discussed, but only Grid Search, Random Search and Simulated Annealing were implemented. These three specific methods were chosen as a way to balance time efficiency and fine tuned exploration. Grid Search is an exhaustive baseline method. The user chooses how fine grained the parameter space is and

[1] `https://wiki.ros.org/rosbag`

[2] Docker is a containerization platform that packages an application and its dependencies into lightweight, portable containers, ensuring consistent behavior across different systems. It enables reproducible environments and simplifies deployment by isolating software from the host system configuration.

the algorithm explores it sequentially. Random Search is a natural improvement over Grid Search, allowing for a more time efficient **random** exploration of the parameter space. Finally, Simulated Annealing is a meta-heuristic that introduces a temperature-controlled acceptance mechanism.

Figure 3.2 displays a low-level diagram of the tuning module developed in Rust and which was integrated into the RUSTLE codebase. The user first creates a YAML configuration file with the SLAM algorithm to be optimized, the dataset on which to optimize the algorithm, the SLAM parameters to tune, any halting conditions, and other relevant data (algorithm dependent). Then, when running the optimization test, the code will keep track of the best configuration found so far, as well as some relevant data, such as parameter and RPE values at each iteration of the algorithm, and export that data. The best configuration is written to a YAML file and the **per iteration** data is written to a csv file, for later processing.

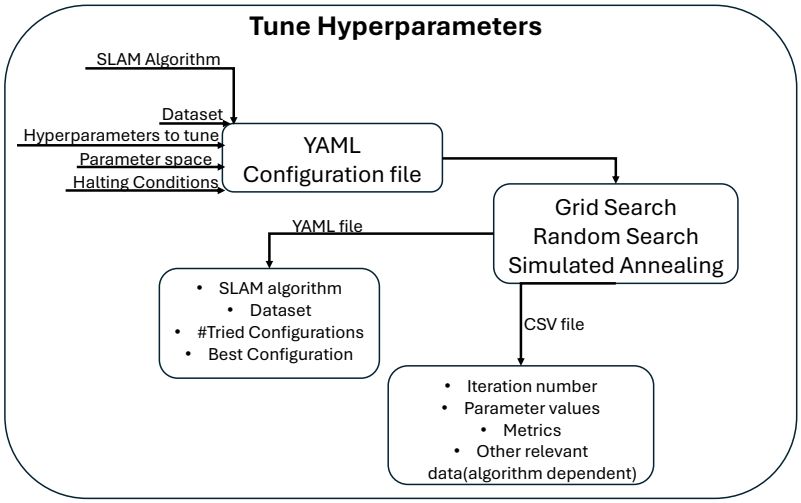


Figure 3.2: Low-level diagram of hyperparameter tuning module.

### 3.2.1 Grid Search

The Grid Search implementation considers the parameter space as a multi dimensional tensor, where each dimension corresponds to a different parameter, and has a differing number of possible values. In each iteration, to obtain the current values of the parameters

**Algorithm 2** Grid Search

1: **for** $t = 1, 2, ..., n$ **do**
2: Obtain the corresponding values of the parameters from the index of the configuration
3: Evaluate the configuration and extract metrics
4: If halting conditions are met(time limit or number of configurations), stop the algorithm's execution
5: **end for**

Figure 3.3: Grid Search procedure

The halting conditions can be customized via a YAML configuration file. Similarly, the parameter space can be designed in the following way:

$$parameter\_name : [first\_value, step\_size, number\_of\_values] \tag{3.1}$$

Additionally, the code looks for and catches most common errors, such as the user passing a String for a parameter that is supposed to be an integer.

### 3.2.2 Random Search

The Random Search Implementation is similar to Grid Search, except that the index of the configuration to be evaluated is randomly generated, and no configuration is evaluated twice, by way of checking a list of previously evaluated configurations.

As in Grid Search, in Random Search, both the parameter space and the halting conditions can be specified in a YAML configuration file.

### 3.2.3 Simulated Annealing

The Simulated Annealing algorithm implementation is identical to the one provided by Stefan Kroboth [58]. It is open source and written in Rust, and only small modifications had to be

**Algorithm 3** Random Search

1: **for** $t = 1, 2, ..., n$ **do**
2: Generate a random number between in the interval [0, n], and check if the number is on the previously generated numbers list. If it is, keep generating numbers until it is not.
3: Add the number to the previously generated numbers list.
4: Obtain the parameter's current values and update the SLAM configuration
5: Evaluate the configuration and extract metrics
6: **if** (elapsed_time > time_limit) or (evaluated_configs > max_iterations) **then** suspend algorithm execution
7: **end if**
8: **end for**

Figure 3.4: Random Search procedure

made for a seamless integration with SLAM tuning.

Perturbation functions are responsible for generating a new candidate solution. For simplicity's sake, only a Gaussian perturbation function was implemented for continuous parameters and a Uniform perturbation function for discrete parameters.

For continuous parameters, the used Gaussian perturbation function samples a number from a Gaussian distribution and adds the sampled value to the current value of the parameter (see Equations 3.2 and 3.3). Both the mean and standard deviation of this normal distribution are unique to each parameter and can be specified in a YAML configuration file.

For any continuous parameter $v$:

$$v_{i+1} = v_i + \Delta_v, \tag{3.2}$$

$$\Delta_v \sim \mathcal{N}(\mu, \sigma^2). \tag{3.3}$$

For non binary discrete parameters, a maximum step size is specified for each parameter, and a random number is generated within a range whose bounds are specified by the maximum step size and the temperature (see Equation 3.5), so it scales up and down with temperature variations.

For any non binary discrete parameter $k$:

$$k_{i+1} = k_i + \Delta_k, \tag{3.4}$$

$$\Delta_k \in [-T_i * max\_step, T_i * max\_step]. \tag{3.5}$$

Another important aspect of Simulated Annealing is the reannealing process. During the execution of the algorithm, it is sometimes necessary to reanneal, that is, to reset the temperature to a previous, higher level, to allow the algorithm to escape (or at least attempt to escape) from local minima. With that in mind, reannealing occurs under three circumstances. The algorithm automatically resets its temperature after a fixed number of iterations; it also does so when no neighboring solution is accepted for a specified stretch of consecutive iterations, and when, after a series of consecutive iterations, no candidate improves upon the current incumbent solution.

As for halting conditions, similarly to reannealing, the algorithm stops its execution if one of three conditions are satisfied. The algorithms stops its execution after a specified maximum number of iterations; it also does so if no neighboring solution is accepted for a specified stretch of consecutive iterations or when, after a series of consecutive iterations, no candidate improves upon the current incumbent solution.

## 3.3 Testing

### 3.3.1 Algorithms

This subsection discusses the SLAM methods used to test the implemented tuning algorithms and why they were chosen.

While almost any SLAM method could be used to demonstrate the results of the implementations of the tuning algorithms mentioned previously, both Faster-LIO and Fast-LIVO2 were chosen for specific reasons. Both Faster-LIO and Fast-LIVO2 are state of the art solutions and are used extensively in the recent literature [59, 60, 61, 10, 62].

#### Faster-LIO

Faster-LIO was selected as the LiDAR–inertial SLAM system in this dissertation due to its demonstrated real-time performance and competitive accuracy in recent literature [59, 63]. The method employs a tightly coupled LiDAR–IMU formulation based on an iterated Kalman

filter [59], enabling efficient local state estimation without relying on loop closure or global optimization. This property makes Faster-LIO well suited for analyzing the influence of hyperparameter choices on odometry accuracy under controlled conditions. In addition, the algorithm exposes a compact yet meaningful set of tunable parameters, which is appropriate for systematic hyperparameter optimization.

### Fast-LIVO2

Fast-LIVO2 was chosen as a complementary SLAM system to extend the analysis to a multimodal LiDAR–visual–inertial system. By integrating visual information into the same tightly coupled estimation framework [10] (see Figure 3.5), Fast-LIVO2 allows for a controlled investigation of how additional sensing modalities affect both system performance and hyperparameter sensitivity.

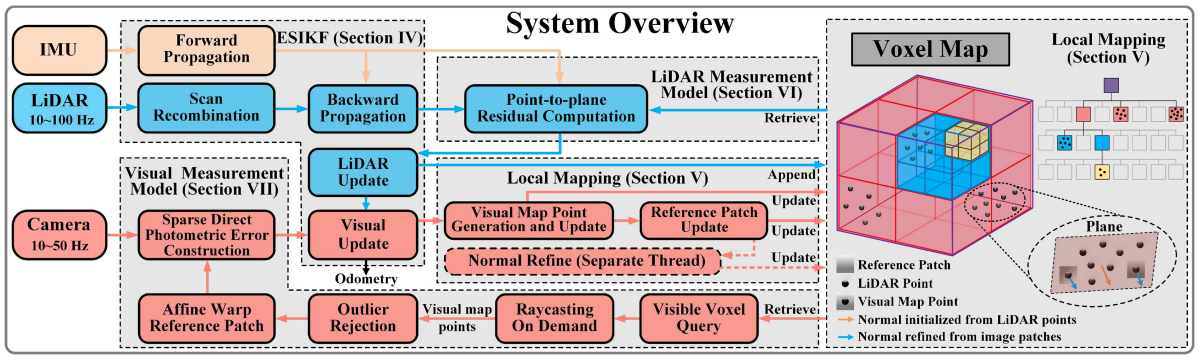


Figure 3.5: Fast-LIVO2 system overview [10]

### 3.3.2 Metrics

During the development, as well as during testing stages, only the RPE was considered for the calculation of the fitness function. The reason is fairly simple: both Faster-LIO and Fast-LIVO2 do not have loop closure mechanisms; as a result, the APE is inherently high. Consequently, optimizing directly for the APE would neither reduce nor eliminate this error, as trajectory drift would continue to accumulate, albeit at a slower rate. Despite this, the user can attribute weights to both the APE and RPE using a YAML configuration file. In addition, the implementation supports, with a few small modifications, to take into account other metrics (CPU load, memory usage, etc).

### 3.3.3 Dataset

During code development, and also during testing and result extraction, the BotanicGarden dataset [64], specifically the 1006_03 sequence was used. This dataset presents a challenging, semi-structured, environment (see Figure 3.6) with diverse sensor modalities, including a DALSA M1930 stereo camera, a DALSA C1930 stereo camera, a Velodyne VLP16 LiDAR, a Livox Avia LiDAR, and an Xsens Mti-680G IMU. All sensors are hardware-synchronized, ensuring time-aligned measurements suitable for sensor fusion. It also provides a robust ground truth, generated using a static terrestrial laser scanner to acquire high-density, high-resolution point clouds. These point clouds were subsequently aligned to produce ground truth trajectories with an accuracy of approximately 1 cm.

Figure 3.6: A bird view of the 3D survey map of BotanicGarden [11]

### 3.3.4 Tuning Strategy

For both Faster-LIO and Fast-LIVO2, a simple tuning strategy as devised., which aims to demonstrate the advantage of hyperparameter optimization in comparison to default parameters, as well as to compare the effectiveness of each implemented tuning algorithm. As for each tuned parameters in both Faster-LIO and Fast-LIVO2, the lower and upper bounds were chosen based on previous experience with these methods, mainly during the development and debugging of the tuning algorithms. Given the parameter bounds, appropriate parameter spaces were designed

for Grid/Random Search, aimed at a runtime of 5 hours, and then both tuning algorithms were executed. As for Simulated Annealing, it is marginally more complex. As explained before, perturbation functions have their own settings parameters, the delta_max for discrete parameters and the mean and standard deviation of the gaussian distribution for non discrete parameters. That being said, several runs of Simulated Annealing were needed to adjust these parameters before they allowed for a proper exploration of the previously defined parameter space (within the defined bounds).

### 3.3.5 Machine

All tests were conducted on a machine with the specifications listed in Table 3.1.

| | |
|---|---|
| CPU | AMD Ryzen 7 2700x 3.7GHz |
| GPU | Nvidia GeForce GTX 1060 6GB |
| Memory | 16GB |
| Disk | 1TB SSD + 1TB HDD |
| OS | Ubuntu 20.04.6 LTS |
| Kernel | 5.15.0-139-generic |

Table 3.1: Test machine specifications

# 4 Results

This chapter presents two case studies involving the tuning of two distinct SLAM algorithms (Faster-LIO and Fast-LIVO2) using the three implemented tuning algorithms. Given that the primary objective of this dissertation is the optimization of SLAM algorithms, a direct comparison of the performance of Faster-LIO and Fast-LIVO2 is not meaningful. Instead, the focus of this section is in showcasing how well the implemented tuning algorithms perform when compared with default parameters executions.

## 4.1 Faster-LIO

Prior to parameter tuning, Faster-LIO was evaluated using its default parameter configuration to establish a baseline for RPE. Table 4.1 presents the five-run average APE and RPE for Faster-LIO, obtained using the parameter values reported in Table 4.2.

| APE (RMSE) | RPE (RMSE) |
|---|---|
| 0.664834 | 0.015037 |

Table 4.1: Default parameter results (5 run average) for Faster-LIO

| | |
|---|---|
| point_filter_num | 3 |
| max_iteration | 3 |
| filter_size_map | 0.5 |
| filter_size_surf | 0.5 |
| cube_side_length | 1000 |
| ivox_grid_resolution | 0.5 |
| esti_plane_threshold | 0.1 |

Table 4.2: Default parameter values for Faster-LIO

Before running Simulated Annealing, it is necessary to define the hyperparameters of the algorithm.

Then, the following parameter space was devised. The two tables below present the parameter settings used for Simulated Annealing. The parameters are divided into two groups: discrete parameters (Table 4.3) and continuous parameters (Table 4.4).

| Parameter | Initial Value | Lower Bound | Upper Bound | delta max |
|---|---|---|---|---|
| point_filter_num | 20 | 2 | 50 | 8 |
| max_iteration | 10 | 3 | 50 | 5 |
| cube_side_length | 1100 | 700 | 2000 | 300 |

Table 4.3: Faster-LIO discrete parameter settings.

| Parameter | Initial Value | Lower Bound | Upper Bound | $\mu$ | $\sigma$ |
|---|---|---|---|---|---|
| filter_size_map | 0.5 | 0.1 | 0.75 | -0.01 | 0.003 |
| filter_size_surf | 1.0 | 0.25 | 1.25 | -0.01 | 0.003 |
| ivox_grid_resolution | 0.5 | 0.1 | 0.75 | -0.01 | 0.003 |
| esti_plane_threshold | 0.5 | 0.1 | 0.75 | -0.01 | 0.003 |

Table 4.4: Faster-LIO continuous parameter settings.

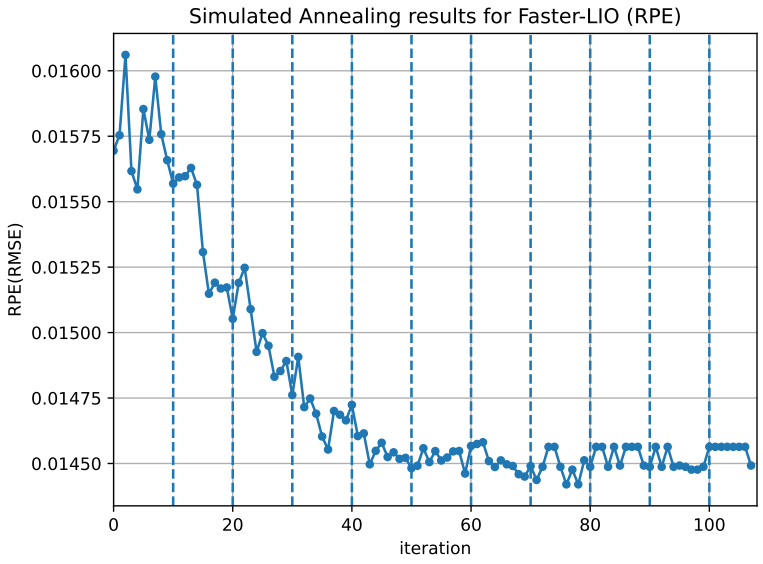


Figure 4.1: RPE per iteration of Simulated Annealing

From Figure 4.1, it can be seen that, roughly for the first 80 iterations, reannealing (vertical dotted lines) allows the algorithm to escape local minima and achieve a better solution than the incumbent. After this point, the algorithm stagnated, with no further improvements observed for nearly 30 iterations, which was therefore selected as the stopping criterion. While this parameter is configurable, preliminary experiments conducted on a 30-second segment of the dataset suggested limited benefit in extending the optimization beyond this threshold. Additionally, due to the considerable runtime of each simulated annealing run (approximately 2.5 minutes per iteration, amounting to 2–5 hours for 50–100 iterations), certain pragmatic compromises were required. The most important one was that most optimizations, in terms of parameter settings, such as delta_max for discrete parameters or gaussian curve $\mu$ and $\sigma$ values for continuous parameters were adjusted while working with 30 second sections of the dataset. While it does not guarantee that such adjustments will work on the entire dataset, it allows the researcher to understand how their values affect algorithm runtime and performance.

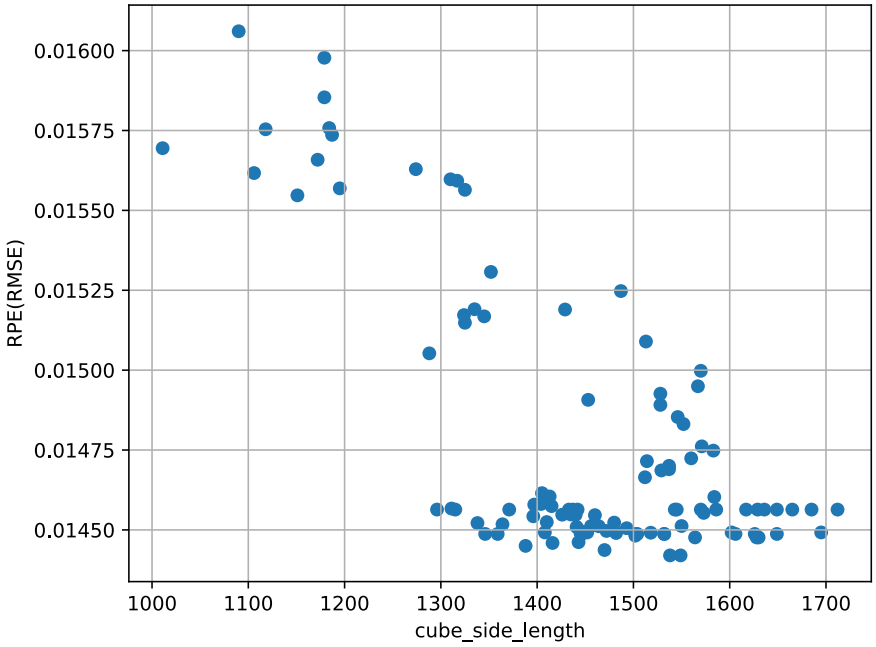


Figure 4.2: cube_side_length scatter plot

Figure 4.2 shows the scatter plot for the discrete parameter cube_side_length. It indicates the optimal range for this parameter is between 1300 and 1700. This is not conclusive, however. Correlation does not mean causation. Other parameters were tuned and their effects on the RPE may be greater, such as ivox_grid_resolution (Figure 4.3). In this case, the relationship between parameter value and the RPE is even clearer, for there seems to be an almost linear relationship between parameter value and RPE.

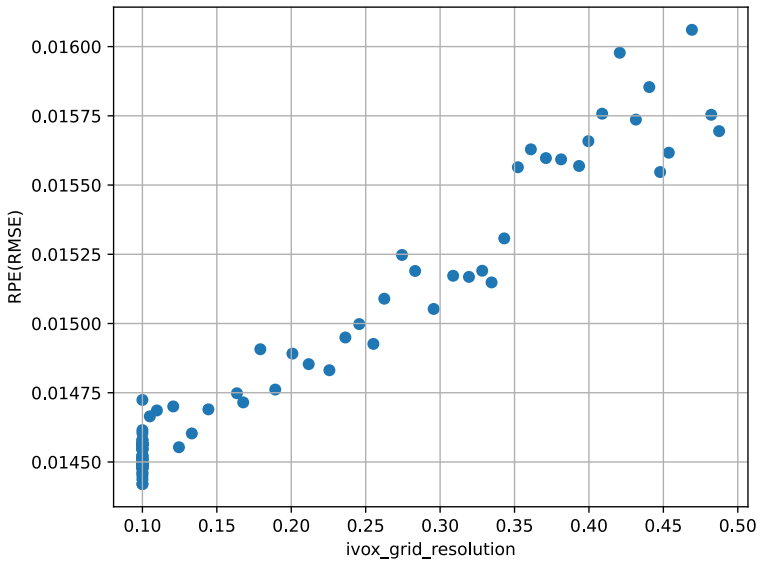


Figure 4.3: ivox_grid_resolution scatter plot

These scatter plots serve as a visual aid to the researcher to better analyze and understand the relationships between tuned parameter values and target metric values.

### 4.1.1 Grid Search

Grid Search was run for 5 hours. Its RPE per iterations is shown in Figure 4.4.

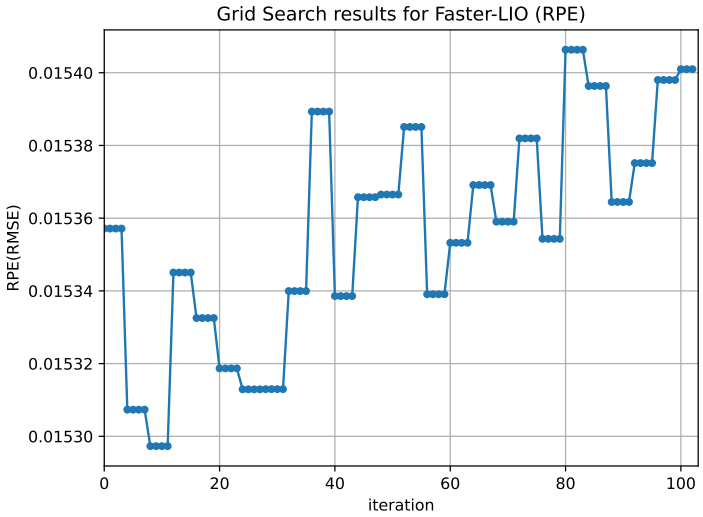


Figure 4.4: RPE per iterations of Grid Search

As with Simulated Annealing, several scatter plots are required to facilitate the interpretation of these results. Figures 4.6, 4.7 and 4.5 present such scatter plots.

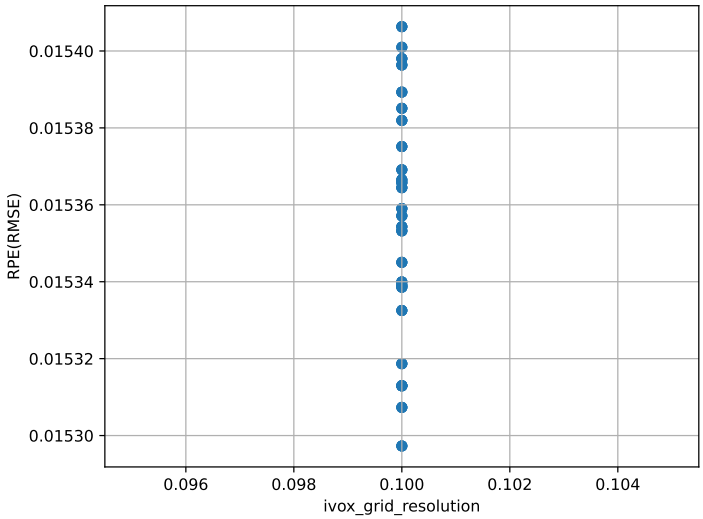


Figure 4.5: ivox_grid_resolution scatter plot

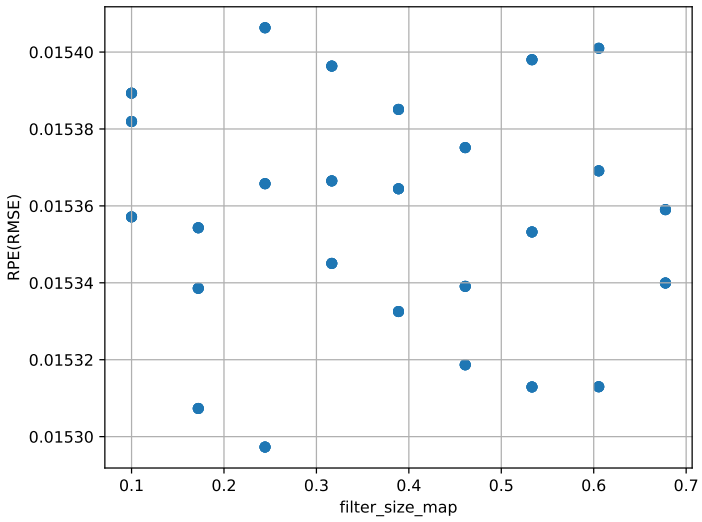


Figure 4.6: filter_size_map scatter plot

The first thing to note is that all tuned parameters have a pattern equal to that of ivox_grid_resolution, hence they are not relevant. The reason for this pattern is the sequential nature of Grid Search.

Secondly, the filter_size_map parameter does not seem to affect the RPE at all, given how sparsely distributed the scatter plot points are. However, filter_size_surf seems to be the only tuned parameter that significantly affects the RPE, having an almost linear relationship. However, given the Simulated Annealing results discussed previously, and specifically figures 4.3 and 4.2, it can be argued that the apparent lack of influence of parameters such as ivox_grid_resolution and cube_side_length on the overall RPE is a consequence of the limited coverage of the parameter space achieved by the Grid Search algorithm. At best, the results

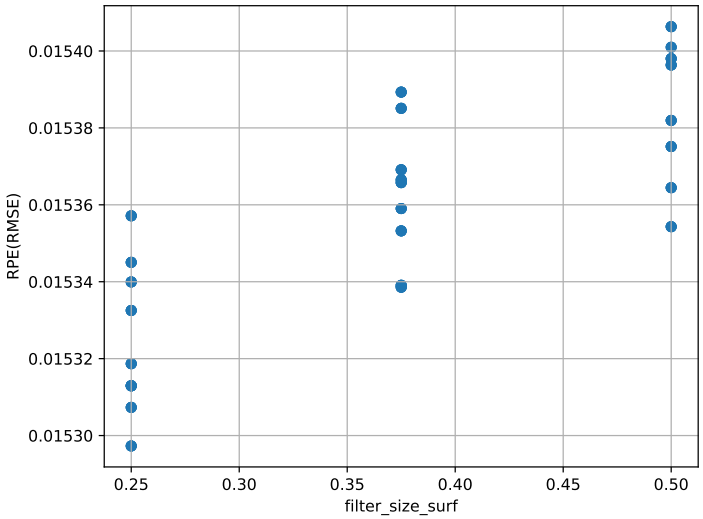


Figure 4.7: filter_size_surf scatter plot

indicate that, when all parameters except filter_size_map and filter_size_surf are held fixed, the observed performance is obtained locally. However, this provides limited insight into the globally optimal parameter configuration for sequence 1006_03 of the BotanicGarden dataset.

### 4.1.2 Random Search

For Random Search, the same parameter space was used, and the algorithm was run for 5 hours, as was the case with Grid Search. In total, both algorithms were run for a total of 100 iterations. Figure 4.8 illustrates the RPE per iteration of Random Search.

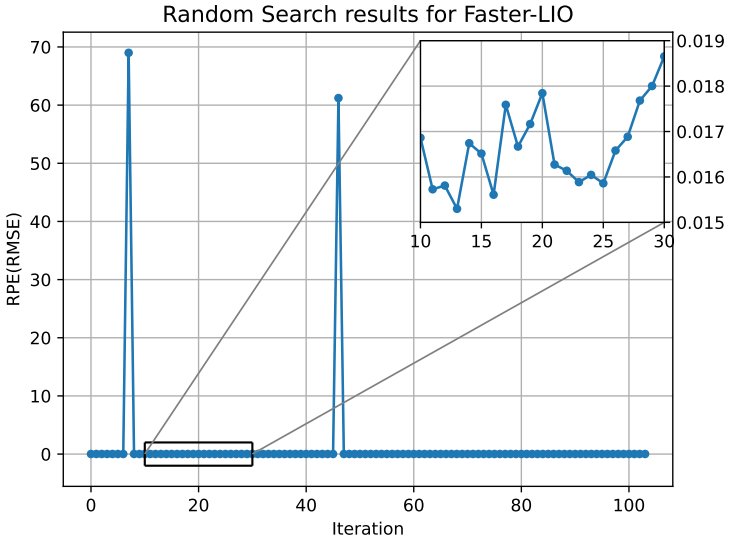


Figure 4.8: RPE per iterations of Random Search

For the scatter plots, the two outlier points were removed, to undo the scale distortion. As with Grid Search, the scatter plots for ivox_grid_resolution (Figure 4.9), filter_size_map (Figure 4.10) and filter_size_surf (Figure 4.11) are presented.

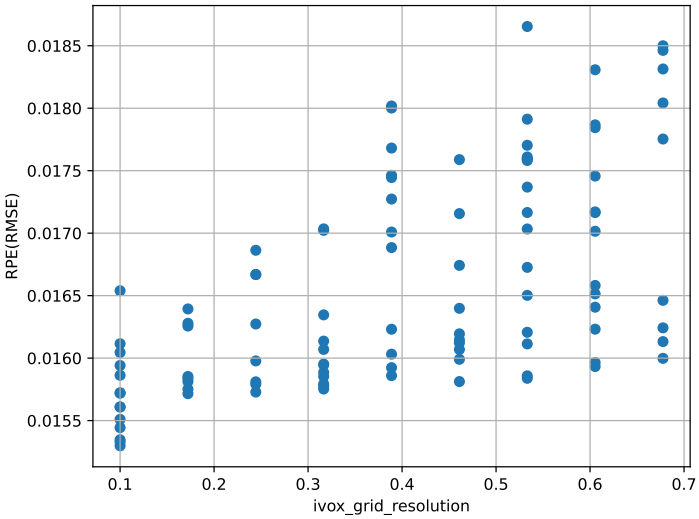


Figure 4.9: ivox_grid_resolution scatter plot

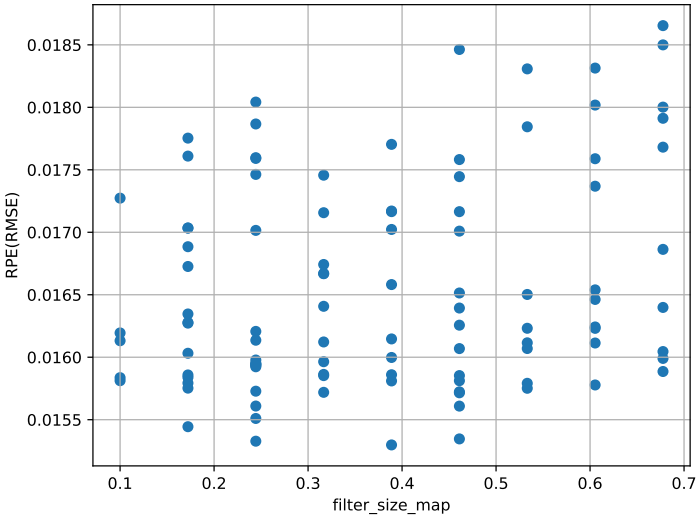


Figure 4.10: filter_size_map scatter plot

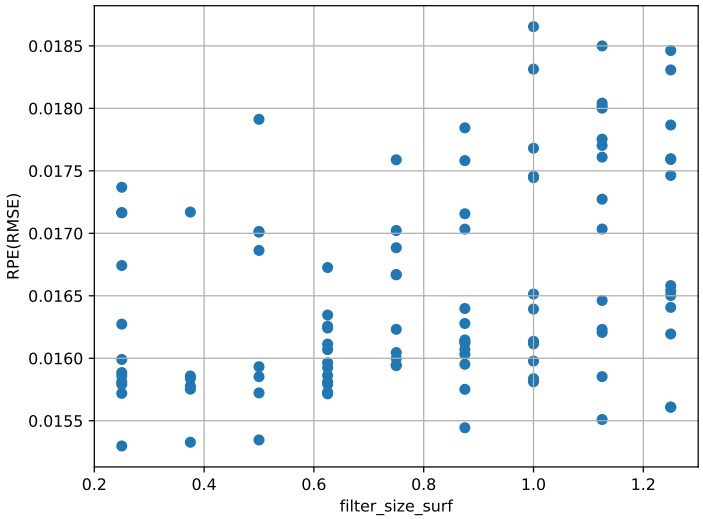


Figure 4.11: filter_size_surf scatter plot

The previous three figures provide a clearer understanding of how these parameters affect the RPE. In each case, all nine possible values are examined, highlighting the primary advantage of Random Search over Grid Search: given a fixed optimization constraint (in this case, a time limit of five hours) and a sufficiently large parameter space (157464 configurations), Random Search not only explores the space more effectively but also generally achieves superior results.

Table 4.5 sums up the results of the tuning process for Faster-LIO.

| | RPE | % difference to default configuration |
|---|---|---|
| Default configuration | 0.015037 | +0% |
| Grid Search | 0.015297 | +1.73% |
| Random Search | 0.015298 | +1.74% |
| Simulated Annealing | 0.014420 | -4.1% |

Table 4.5: Faster-LIO tuning results

## 4.2 Fast-LIVO2

As was the case with Faster-LIO, prior to parameter tuning, Fast-LIVO2 was evaluated using its default parameter configuration to establish a baseline for RPE. Table 4.6 presents the five-run average APE and RPE for Fast-LIVO2, obtained using the parameter values reported in Table 4.7.

| APE (RMSE) | RPE (RMSE) |
|---|---|
| 0.403815 | 0.045583 |

Table 4.6: Default parameter results (5 run average) for Fast-LIVO2

| | |
|---|---|
| point_filter_num | 1 |
| max_iterations | 5 |
| filter_size_pcd | 0.15 |
| filter_size_surf | 0.1 |
| half_map_size | 100 |
| patch_pyrimid_level | 4 |
| max_points_num | 50 |
| patch_size | 8 |
| max_layer | 2 |
| outlier_threshold | 1000 |
| sliding_thresh | 8 |
| voxel_size | 0.5 |

Table 4.7: Default parameter values for Fast-LIVO2

Before running Simulated Annealing, it is necessary to define the hyperparameters of the tuning algorithm.

Then, the following parameter space was devised. The two tables below present the parameter settings used for Simulated Annealing. The parameters are divided into two groups: discrete parameters (Table 4.8) and continuous parameters (Table 4.9).

| Parameter | Initial Value | Lower Bound | Upper Bound | delta max |
|---|---|---|---|---|
| point_filter_num | 100 | 5 | 150 | 20 |
| max_iterations | 70 | 40 | 95 | 15 |
| patch_size | 18 | 5 | 25 | 10 |
| patch_pyrimid_level | 18 | 4 | 32 | 7 |
| max_layer | 25 | 2 | 50 | 10 |
| max_points_num | 105 | 90 | 120 | 50 |
| half_map_size | 400 | 75 | 750 | 150 |
| sliding_thresh | 15 | 8 | 25 | 7 |
| outlier_threshold | 4000 | 750 | 7500 | 1500 |

Table 4.8: Fast-LIVO2 discrete parameter settings

| Parameter | Initial Value | Lower Bound | Upper Bound | $\mu$ | $\sigma$ |
|---|---|---|---|---|---|
| filter_size_surf | 1.45 | 0.1 | 5.0 | -0.035 | 0.003 |
| voxel_size | 2.5 | 0.2 | 17.75 | -0.055 | 0.003 |

Table 4.9: Fast-LIVO2 continuous parameter settings

From Figure 4.12, it can be seen that, roughly for the first 70 iterations, reannealing allows the algorithm to escape local minima and achieve a better solution than the incumbent. After this point, the algorithm seemed to stagnate, with no further improvements observed until iteration 100, after which it halted its execution. Due to the considerable runtime of each simulated annealing run (approximately 2.5 minutes per iteration, amounting to 2–5 hours for 50–100 iterations), as well as the increased number of parameters of Fast-LIVO2 relative to Faster-LIO, some conclusions can be drawn. One, Simulated Annealing requires constant interaction and feedback from the tuner in order to adjust the algorithm hyperparameters, particularly the perturbation functions' settings, taking several runs before performance reaches acceptable levels.

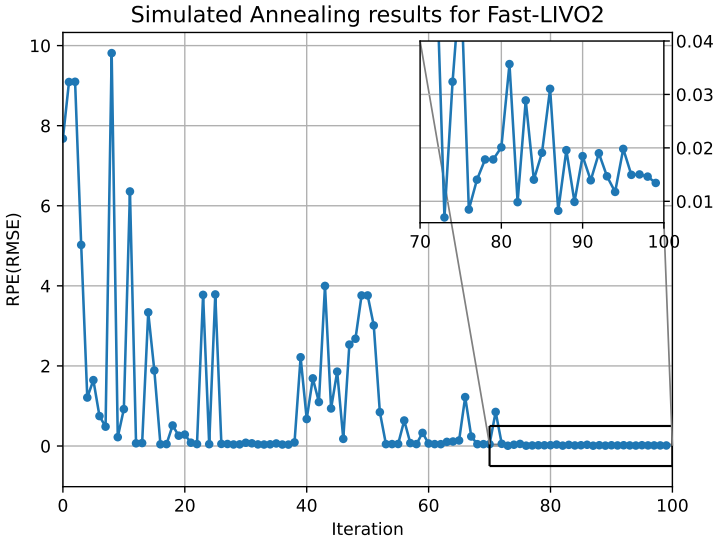


Figure 4.12: RPE per iteration of Simulated Annealing

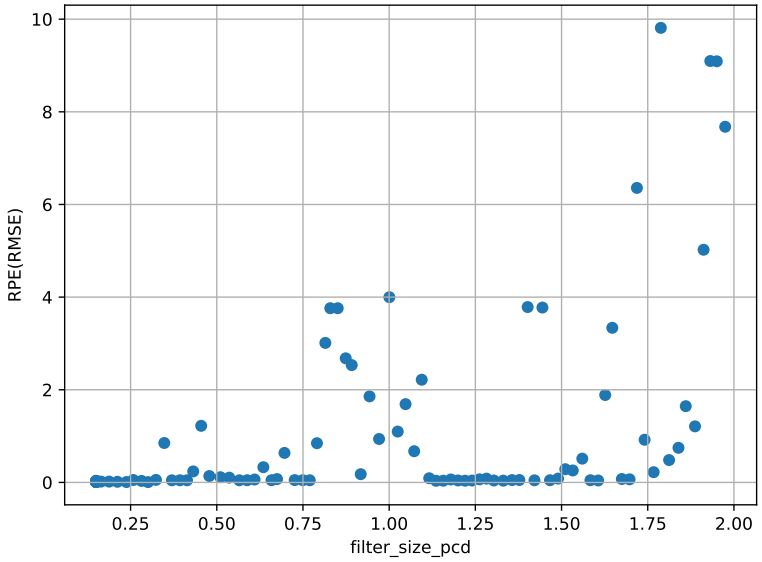


Figure 4.13: filter_size_pcd scatter plot

Figure 4.13 shows the scatter plot for the continuous parameter filter_size_pcd. It seems to indicate the optimal range for this parameter is between 0.2 and 0.75. This is not conclusive, however, as other tuned parameters' scatter plots displayed a similar pattern (see figure 4.14), indicating multiple parameters with a strong quasi-linear relationship with the RPE.

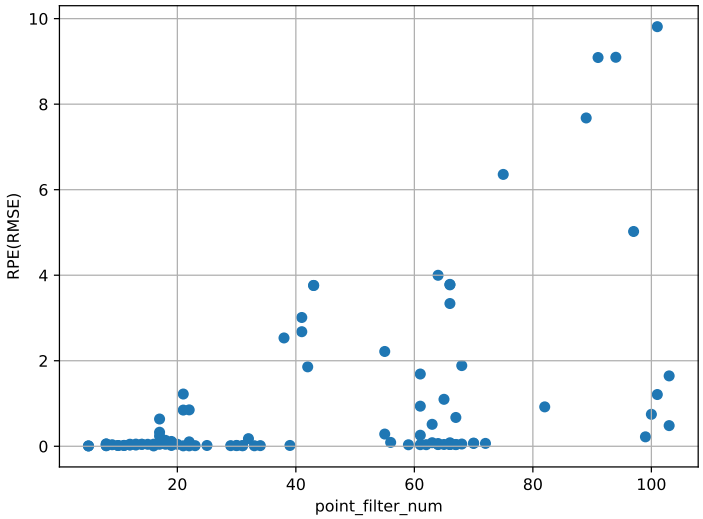


Figure 4.14: point_filter_num scatter plot

### 4.2.1 Grid Search

Similarly to Faster-LIO, Grid Search was run Fast-LIVO2 for 5 hours. Its RPE per iterations is shown in Figure 4.15.

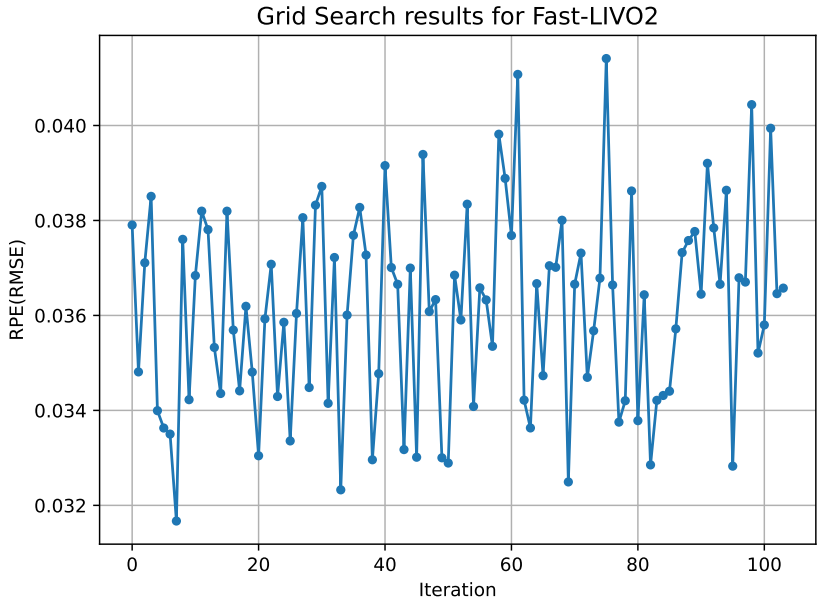


Figure 4.15: RPE per iteration for Grid Search

As with Simulated Annealing, several scatter plots are required to facilitate the interpretation of these results. Figures 4.16 and 4.17 present such scatter plots.

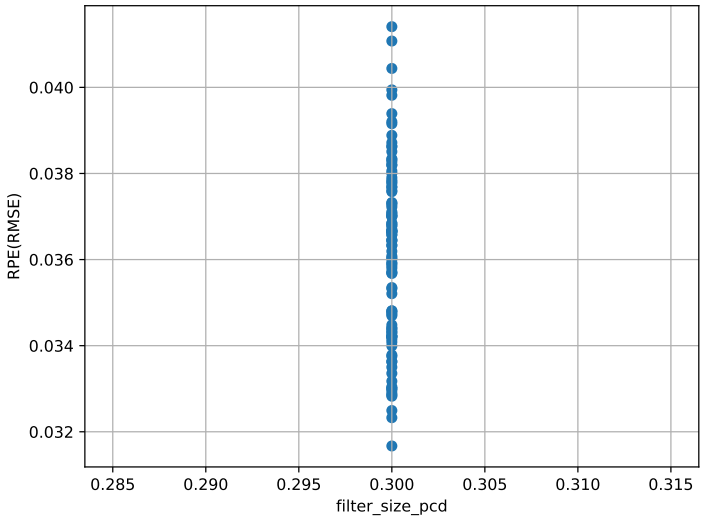


Figure 4.16: filter_size_pcd scatter plot

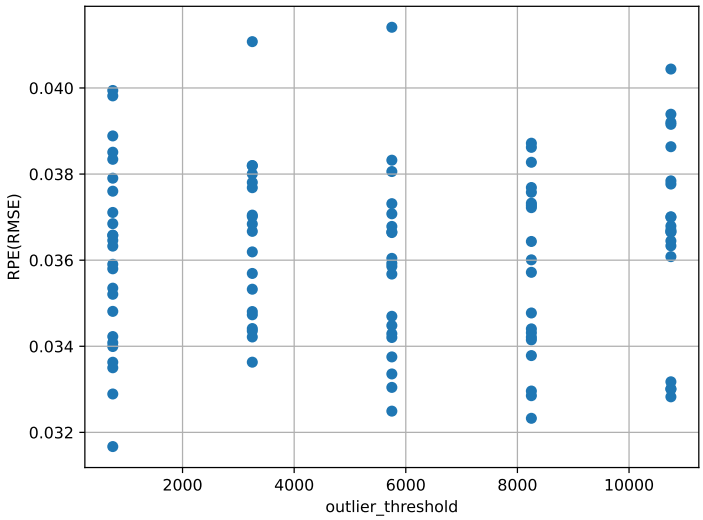


Figure 4.17: outlier_threshold scatter plot

The first thing that should be noted is that while in Faster-LIO seven parameters were tuned, in Fast-LIVO2 there are twelve parameters being tuned, and therefore, taking into account practical limitations, this means the parameter space is necessarily sparser and not detailed enough for a conclusive analysis. Despite that, Grid Search still manages to achieve an RPE value of 0.0316, which is about 30.5% lower than the default configuration.

### 4.2.2 Random Search

As before, for Random Search, the same parameter space was used, and the algorithm was run for 5 hours, as was the case with Faster-LIO. In total, both algorithms were run for a total of 100 iterations. Figure 4.8 illustrates the RPE per iteration of Random Search.

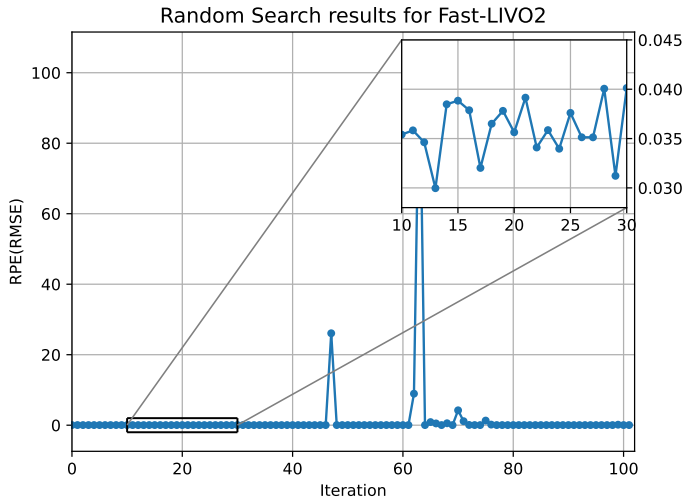


Figure 4.18: RPE per iterations of Random Search

As with Grid Search, the scatter plots for filter_size_pcd (Figure 4.19) and outlier_threshold (Figure 4.20) are presented.

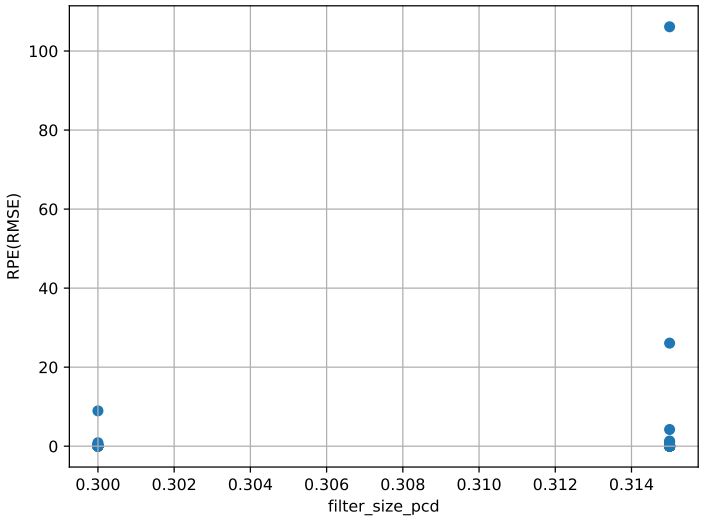


Figure 4.19: filter_size_pcd scatter plot

Unlike Faster-LIO, it is not possible to derive any meaningful conclusions from the Random Search scatter plots for Fast-LIVO2. Even though Random Search managed to achieve an RPE value 5.4% lower than Grid Search, mostly due to its pseudo-random nature, the increased number of tuned parameters for Fast-LIVO2, as mentioned previously, forces the tuner to design a low resolution search space, limiting the potential benefits.

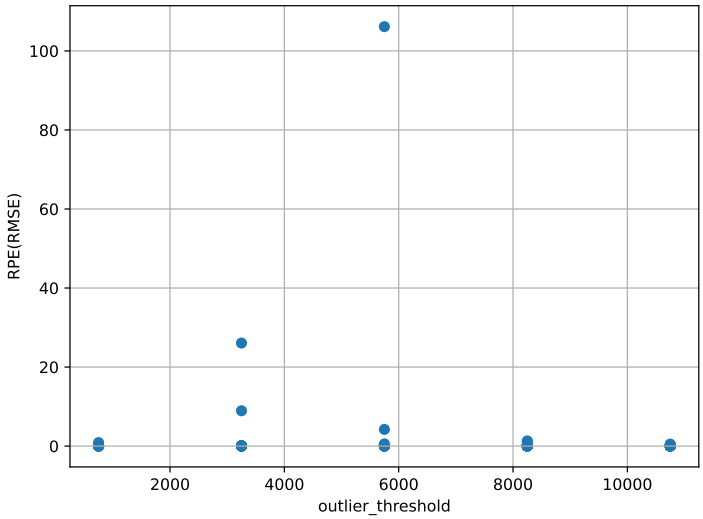


Figure 4.20: outlier_threshold scatter plot

Table 4.10 sums up the results of the tuning process for Fast-LIVO2.

| | RPE | % difference to default configuration |
|---|---|---|
| Default configuration | 0.045583 | +0% |
| Grid Search | 0.031670 | -30.52% |
| Random Search | 0.029969 | -34.25% |
| Simulated Annealing | 0.006979 | -84.69% |

Table 4.10: Fast-LIVO2 tuning results

# 5 Conclusion

In this dissertation, three parameter optimization algorithms for SLAM systems were implemented, namely Grid Search, Random Search, and Simulated Annealing. It has five components: dataset definition, SLAM algorithm, tuning algorithm and parameter selection, and metric weights. The parameter selection requires a search space to be delimited. For Grid/Random Search, this mean defining upper and lower bounds, as well as a step size, for each parameter. For Simulated Annealing, additional settings related to perturbation functions and reannealing must be specified. Assigning weights to each metric (in this case, only the APE and RPE were considered) enables the research to optimize the selected SLAM algorithm with respect to a specific metric. All relevant components can be specified through an external configuration file.

For testing the implementations, the BotanicGarden dataset was used, specifically the 1006_03 sequence, and the chosen state-of-the-art SLAM solutions were Faster-LIO and Fast-LIVO2. As for metrics, both Faster-LIO and Fast-LIVO2 were optimized for the RPE. A few parameters were chosen for both algorithms, and their upper and lower bounds delimited accordingly. The tuning algorithms algorithms were then run and their performance compared. It was found that Grid and Random Search had a slightly worse performance than the default executions, while Simulated Annealing managed to achieve a 4.1% lower RPE in the case of Faster-LIO, and a 84.69% lower RPE in the case of Fast-LIVO2. Although Grid and Random Search are both baseline methods, their poor performance is partly explained by the fact that RUSTLE stores every SLAM data on a local database, thereby substantially increasing both disk usage and run time, imposing a practical limit on the granularity of the search space for both Grid and Random Search. Nevertheless, the implemented tuning algorithms are universal (agnostic to the SLAM solution they are optimizing) and can be a viable option for increasing the performance of any SLAM solution.

## 5.1 Future work

Three parameter tuning algorithms were adapted and implemented for SLAM optimization. These algorithms demonstrate potential for improving SLAM performance; however, there is still significant room for improvements.

### Tuning Algorithms

Section 3.2 proposed using Grid Search, Random Search and Simulated Annealing as the algorithms to fine tune SLAM. Due to time constraints, it was not possible to implement other, more efficient algorithms such as Bayesian Optimization or Successive Halving. Future work will involve implementing those, as well to fine tune the already implementated algorithms. Some examples of such improvements are tweaking the halting/reannealing conditions of Simulated Annealing, or adjusting the step sizes of a few parameters in Grid/Random Search for finder exploration.

Additionally, given the moderate success of Simulated Annealing, future work will involve studying and implementing additional meta-heuristic algorithms, such as Particle Swarm Optimization and Genetic Algorithms.

### Other SLAM Solutions

The implemented tuning algorithms were applied to Faster-LIO and Fast-LIVO2. In principle, the tuning process depends on the selected dataset rather than the specific SLAM algorithm. It was therefore assumed that the implemented optimization algorithms could be equally effective for other SLAM solutions. Future work will involve applying these algorithms to alternative SLAM frameworks (e.g., IG-LIO, LIO-SAM) and evaluating the relative effectiveness of parameter tuning across different SLAM methods.

# Bibliography


[1] S. A. K. Tareen, "Large scale 3d simultaneous localization and mapping (ls-3d-slam) using monocular vision," Master's thesis, Pakistan Navy Engineering College, Islamabad, Pakistan, 2019.

[2] D. Casado, "Introduction to visual slam: Chapter 1 —introduction to slam." `https://medium.com/@dcasadoherraez/introduction-to-visual-slam-chapter-1-introduction-to-slam-a0211654bf0e`, 2021.

[3] F. Andert, O. Böttcher, A. Mushyam, and P. Schmälzle, "Semantic lidar point cloud mapping and cloud-based slam for autonomous driving," in *2025 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 853–860, IEEE, 2025.

[4] R. C. Luo, S. L. Lee, Y. C. Wen, and C. H. Hsu, "Modular ros based autonomous mobile industrial robot system for automated intelligent manufacturing applications," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1673–1678, IEEE, 2020.

[5] E. I. M. Castillo, *Hyperparameter Optimization for SLAM: An Approach for Enhancing ORB-SLAM2's Performance*. PhD thesis, University of Alberta, 2022.

[6] 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.

[7] F. S. Caparrini, "Simulated annealing in netlogo." `https://www.cs.us.es/~fsancho/Blog/posts/Simulated_Annealing_in_NetLogo.md`, 2019.

[8] D. S. Soper, "Hyperparameter optimization using successive halving with greedy cross validation," *Algorithms*, vol. 16, no. 1, p. 17, 2022.

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[10] B. Zhou, C. Zheng, Z. Wang, F. Zhu, Y. Cai, and F. Zhang, "Fast-livo2 on resource-constrained platforms: Lidar-inertial-visual odometry with efficient memory and computation," *IEEE Robotics and Automation Letters*, 2025.

[11] Y. Liu, Y. Fu, M. Qin, Y. Xu, B. Xu, F. Chen, B. Goossens, P. Z. Sun, H. Yu, C. Liu, *et al.*, "Botanicgarden: A high-quality dataset for robot navigation in unstructured natural environments," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2798–2805, 2024.

[12] R. Beņkis, E. Grabs, T. Chen, A. Ratkuns, D. Čulkovs, A. Ancāns, and A. Ipatovs, "A survey and practical application of slam algorithms," in *2024 Photonics & Electromagnetics Research Symposium (PIERS)*, pp. 1–10, IEEE, 2024.

[13] P.-Y. Lajoie, B. Ramtoula, F. Wu, and G. Beltrame, "Towards collaborative simultaneous localization and mapping: a survey of the current research landscape," *Field Robotics*, vol. 2, pp. 971–1000, 2022.

[14] I. A. Kazerouni, L. Fitzgerald, G. Dooly, and D. Toal, "A survey of state-of-the-art on visual slam," *Expert Systems with Applications*, vol. 205, p. 117734, 2022.

[15] Y. Li, J. An, N. He, Y. Li, Z. Han, Z. Chen, and Y. Qu, "A review of simultaneous localization and mapping algorithms based on lidar," *World Electric Vehicle Journal*, vol. 16, no. 2, p. 56, 2025.

[16] H. Taheri and Z. C. Xia, "Slam; definition and evolution," *Engineering Applications of Artificial Intelligence*, vol. 97, p. 104032, 2021.

[17] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A comprehensive survey of visual slam algorithms," *Robotics*, vol. 11, no. 1, p. 24, 2022.

[18] C. Harris, M. Stephens, *et al.*, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, pp. 10–5244, Manchester, UK, 1988.

[19] T. Lindeberg, "Feature detection with automatic scale selection," *International journal of computer vision*, vol. 30, no. 2, pp. 79–116, 1998.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[21] Y. Liao, Y. Di, K. Zhu, H. Zhou, M. Lu, Y. Zhang, Q. Duan, and J. Liu, "Local feature matching from detector-based to detector-free: a survey," *Applied Intelligence*, vol. 54, no. 5, pp. 3954–3989, 2024.

[22] W. Chen, L. Zhu, Y. Guan, C. R. Kube, and H. Zhang, "Submap-based pose-graph visual slam: A robust visual exploration and localization system," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6851–6856, IEEE, 2018.

[23] Q. Picard, S. Chevobbe, M. Darouich, and J. Didier, "A survey on real-time 3d scene reconstruction with slam methods in embedded systems. arxiv 2023," *arXiv preprint arXiv:2309.05349*.

[24] Z. Li, F. Ye, and X. Guan, "3d point cloud reconstruction and slam as an input," *arXiv preprint arXiv:2112.12907*, 2021.

[25] Y. Chen, Y. Zhou, Q. Lv, and K. K. Deveerasetty, "A review of v-slam," in *2018 IEEE International Conference on Information and Automation (ICIA)*, pp. 603–608, IEEE, 2018.

[26] A. Fontan, J. Civera, and M. Milford, "Anyfeature-vslam: Automating the usage of any chosen feature into visual slam," in *Proceedings of the Robotics: Science and Systems*, vol. 2, 2024.

[27] R. Raguram, J.-M. Frahm, and M. Pollefeys, "A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus," in *European conference on computer vision*, pp. 500–513, Springer, 2008.

[28] D. Yan, W. Tuo, W. Wang, and S. Li, "Illumination robust loop closure detection with the constraint of pose," *arXiv preprint arXiv:1912.12367*, 2019.

[29] X. Yue, Y. Zhang, J. Chen, J. Chen, X. Zhou, and M. He, "Lidar-based slam for robotic mapping: state of the art and new frontiers," *Industrial Robot: the international journal of robotics research and application*, vol. 51, no. 2, pp. 196–205, 2024.

[30] D. Zhu, Q. Wang, F. Wang, and X. Gong, "Research on 3d lidar outdoor slam algorithm based on lidar/imu tight coupling," *Scientific Reports*, vol. 15, no. 1, p. 11175, 2025.

[31] J. Jorge, T. Barros, C. Premebida, M. Aleksandrov, D. Goehring, and U. Nunes, "Impact of 3d lidar resolution in graph-based slam approaches: A comparative study," in *2024 7th Iberian Robotics Conference (ROBOT)*, pp. 1–6, IEEE, 2024.

[32] H. Wang, Y. Yin, and Q. Jing, "Comparative analysis of 3d lidar scan-matching methods for state estimation of autonomous surface vessel," *Journal of Marine Science and Engineering*, vol. 11, no. 4, p. 840, 2023.

[33] N. Stathoulopoulos, V. Sumathy, C. Kanellakis, and G. Nikolakopoulos, "Why sample space matters: Keyframe sampling optimization for lidar-based place recognition," *arXiv preprint arXiv:2410.02643*, 2024.

[34] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, "Visual and visual-inertial slam: State of the art, classification, and experimental benchmarking," *Journal of Sensors*, vol. 2021, no. 1, p. 2054828, 2021.

[35] Z. Liu, H. Li, C. Yuan, X. Liu, J. Lin, R. Li, C. Zheng, B. Zhou, W. Liu, and F. Zhang, "Voxel-slam: A complete, accurate, and versatile lidar-inertial slam system," *arXiv preprint arXiv:2410.08935*, 2024.

[36] K. Liu, "A lidar-inertial-visual slam system with loop detection," *arXiv preprint arXiv:2301.05604*, 2023.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The international journal of robotics research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[38] M. Grupp, "evo: Python package for the evaluation of odometry and slam.." `https://github.com/MichaelGrupp/evo`, 2017.

[39] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[40] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The journal of machine learning research*, vol. 13, no. 1, pp. 281–305, 2012.

[41] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. arxiv," *arXiv preprint arXiv:2107.05847*, 2021.

[42] A. Bissuel, "Hyper-parameter optimization algorithms: a short review," 2020. `https://medium.com/criteo-labs/hyper-parameter-optimizationalgorithms-2fe447525903`.

[43] A. Morales-Hernández, I. Van Nieuwenhuyse, and S. Rojas Gonzalez, "A survey on multi-objective hyperparameter optimization algorithms for machine learning," *Artificial Intelligence Review*, vol. 56, no. 8, pp. 8043–8093, 2023.

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

[45] R. Elshawi, M. Maher, and S. Sakr, "Automated machine learning: State-of-the-art and open challenges," *arXiv preprint arXiv:1906.02287*, 2019.

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

[47] A. Kostusiak and P. Skrzypczyński, "On the efficiency of population-based optimization in finding best parameters for rgb-d visual odometry," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 13, 2019.

[48] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits and Devices magazine*, vol. 5, no. 1, pp. 19–26, 1989.

[49] O. Ghasemalizadeh, S. Khaleghian, and S. Taheri, "A review of optimization techniques in artificial networks," *International Journal of Advanced Research*, vol. 4, no. 9, pp. 1668–86, 2016.

[50] D. S. Soper, "Greed is good: Rapid hyperparameter optimization and model selection using greedy k-fold cross validation," *Electronics*, vol. 10, no. 16, p. 1973, 2021.

[51] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE transactions on evolutionary computation*, vol. 24, no. 2, pp. 201–216, 2019.

[52] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International conference on machine learning*, pp. 1437–1446, PMLR, 2018.

[53] I. A. Putra and P. Prajitno, "Parameter tuning of g-mapping slam (simultaneous localization and mapping) on mobile robot with laser-range finder 360 sensor," in *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pp. 148–153, IEEE, 2019.

[54] Z. Zheng, "Feature based monocular visual odometry for autonomous driving and hyperparameter tuning to improve trajectory estimation," in *Journal of Physics: Conference Series*, vol. 1453, p. 012067, IOP Publishing, 2020.

[55] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Automatic hyper-parameter tuning for black-box lidar odometry," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5069–5074, IEEE, 2021.

[56] Z. Chen, *Visual-inertial slam extrinsic parameter calibration based on bayesian optimization.* University of Colorado at Boulder, 2018.

[57] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, *et al.*, "Slambench2: Multi-objective head-to-head benchmarking for visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3637–3644, IEEE, 2018.

[58] S. Kroboth, "argmin: Numerical optimization in rust," 2024. `https://github.com/argmin-rs/argmin`.

[59] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-lio: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4861–4868, 2022.

[60] S. Wang, H. Huang, J. Yang, J. Bian, and S. Jiang, "Triangle descriptor loop detection method based on faster-lio," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 48, pp. 183–189, 2024.

[61] T. Wang, S. Li, and Y. Chen, "Lidar odometry: Recent advancements and remaining challenges," *Journal of Real-Time Image Processing*, 2024.

[62] B. Zhou, C. Zheng, Z. Wang, F. Zhu, Y. Cai, and F. Zhang, "Fast-livo2 on resource-constrained platforms: Lidar-inertial-visual odometry with efficient memory and computation," 2025.

[63] J. Będkowski, M. Matecki, M. Pełka, K. Majek, P. Lekston, A. Kostrzewa, J. Markiewicz, S. Łapiński, M. Własiuk, and K. Mrozowski, "The benchmark of lidar odometry algorithms utilised for a low-cost mobile mapping system," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 48, pp. 25–32, 2025.

[64] Y. Liu, Y. Fu, M. Qin, Y. Xu, B. Xu, F. Chen, B. Goossens, P. Z. Sun, H. Yu, C. Liu, L. Chen, W. Tao, and H. Zhao, "Botanicgarden: A high-quality dataset for robot navigation in unstructured natural environments," *IEEE Robotics and Automation Letters*, vol. 9, p. 2798–2805, Mar. 2024.