UNIVERSIDADE D
COIMBRA

Francisco Carvalho Tavares

AUTOMATED PARAMETER TUNING WITH RUST FOR EVALUATION OF SLAM METHODS

SUBTÍTULO TBD

Master's Dissertation in MEEC, supervised by Dr. David B. S. Portugal and Mário Cristovão and presented to the Faculty of Science and Technology of the University of Coimbra

November 2024

1 2 9 0

UNIVERSIDADE D
COIMBRA

Automated Parameter Tuning with Rust for Evaluation of SLAM Methods

Supervisor:

Prof. David Portugal

Co-Supervisor:

Eng. Mário Cristovão

Jury:

Prof. Jury1

Prof. Jury2

Prof. Jury3

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

5th of November, 2024

Acknowledgments

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Resumo

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Abstract

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

"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

DEA	Dissertation's Example Acronym
BO	Bayesian Optimization
SA	Simulated Annealing
SH	Successive Halving
HB	Hyperband
HPT	Hyperparameter Tuning
PSO	Particle Swarm Optimization
EA	Evolutionary Algorithm
VO	Visual Odometry
SLAM	Simultaneous Localization and Mapping
ICP	Iterative Closest Point
NDT	Normal Distributions Transform
RUSTLE	Reliable User-friendly and Straightforward Tool for Localization Experiments

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

SLAM is one of the most studied topics in robotics. Its purpose is to simultaneously estimate the robot's position and and map the robot's environment. SLAM methods often integrate data from devices like LIDAR, cameras, or ultrasonic sensors with algorithms like Kalman filters and particle filters, or more advanced aproaches, such as Visual and LIDAR SLAM. These methods are crucial for applications in robotics, augmented reality, and autonomous navigation.

The effectiveness of a SLAM method is usually evaluated using the Absolute Pose Error(APE) and the Relative Pose Error(RPE), meaning, the Absolute difference between the estimated pose and the ground truth and the difference between consecutive estimated poses and the respective ground truth, or the trajectory drift. SLAM's efficacy is dependent on various factors, such as sensor quality, available computational resources, loop closure and the 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.

One of the biggest hurdles in SLAM research is determining the best hyperparameter configuration for a given dataset. A tool like RUSTLE streamlines the 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 crutial component from frameworks like RUSTLE: in the literature, SLAM methods are usually not optimally tuned, or the comparisons are between methods that are almost optimally tuned and methods that are tuned just enough to give sufficiently satisfactory results, and so 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 thesis are:

- Develop an automatic HPT framework within RUSTLE to better optimize and compare the performance of various SLAM solutions.
- Get proficient at Rust programming, which not only will help with the development of the previously mentioned framework, but could also be of use in future works.
- Evaluate the efficiency of the developed optimization algorithms and compare them to each other.
- Summarize the developed work and identify lessons learned and potential future improvements.

1.3 Document overview

2 Background and Related Work

This chapter goes over 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 thesis.

2.1 SLAM

SLAM is a fundamental problem in robotics and computer vision, wherein a system builds a map of an unknown environment while simultaneously determining its own position within said map[1].

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 itself[1], which is implementation dependent.

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

- Sensor data collection
- Feature extraction, to identify key features like landmarks, corners or edges
- Data association, where features from different sensor readings are matches with one another to better understand motion
- Pose estimation, where the position and orientation of the device within the environment is determined
- Map building - Using the sensor data extracted previously, a map of the environment is constructed and dynamically updated
- Loop closure - By recognizing previously visited locations, it is possible to correct some errors, such as drift

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, Radar SLAM and multi sensor SLAM.

Visual SLAM

VSLAM uses cameras to understand an environment by detecting and tracking features over time[2]. Common subtypes of VSLAM are(according to the type of cameras 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).

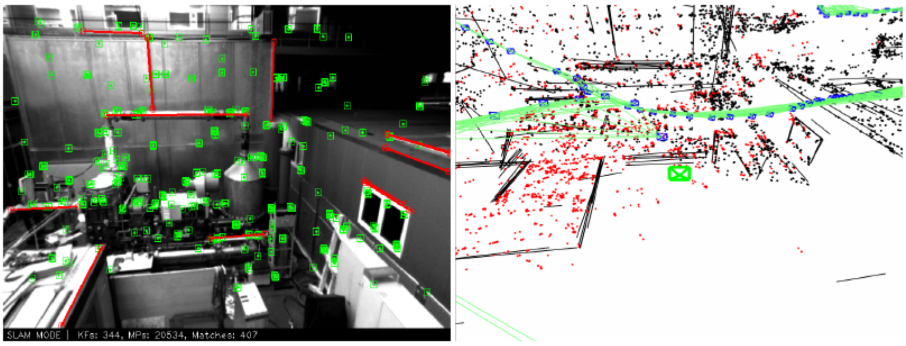


Figure 2.1: Visual SLAM

In terms of system parameters, one has to consider those related to the different parts of the system, such as:

- Feature detection and matching
  - feature matching threshold
  - feature detection sensitivity
  - scale factor(in case of pyramid-based detectors)
- Pose/Motion estimation
  - RANSAC threshold
  - Number of RANSAC iterations, which affect the robustness and speed of the algorithm
  - Minimum inliers, to control how many inliers before an estimated pose is accepted

- Mapping
  - Keyframe insertion threshold, which controlls how different a frame must be to be added as a keyframe
  - Map point culling threshold, the criteria for removing bad or unused key points
- Loop closure
  - Minimum time between loops
  - Loop detection threshold, level of confidence required to accept a loop

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

LiDAR SLAM

LiDAR SLAM uses, as the name suggests, LiDAR(Light detection and ranging) sensors, which unlike VSLAM, can measure distances directly using laser pulses. It is overall a more robust method for low light and featureless environments.[3]

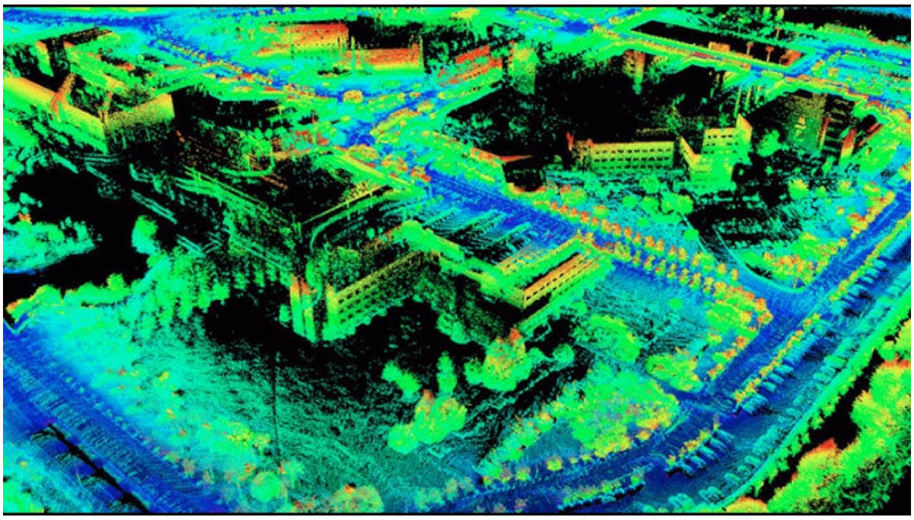


Figure 2.2: LiDAR SLAM point cloud example

In terms of system parameters, LiDAR solutions require preprocessing, introducing important parameters, such as Voxel grid size(Resolution for downsampling the point cloud) or the range threshold(range for keeping/eliminating points)[3]. As for the scan registration/motion estimation part of the method, according to the scan registration algorithm used(Iterative Closest Point (ICP), Normal Distributions Transform (NDT), or other), different parameters will be used[3].

Radar SLAM

Radar SLAM uses radar sensors that gather data by emitting electromagnetic waves and analyzing their reflections on surrounding objects. This is particularly useful in adverse weather conditions, where Visual SLAM doesn't work as well[4]. However, it also has some serious limitations. For starters, a radar can't capture detailed structures the way a camera or lidar can. There is also the problem of lower resolutions, when compared to technologies like LiDAR[5]. Due to these limitations, Radar is usually accompanied by other sensors, such as IMU or cameras of some type, making the overall SLAM system a **Multi sensor** system and taking advantage of the strengths of each sensor for a more robust map and trajectory estimation.

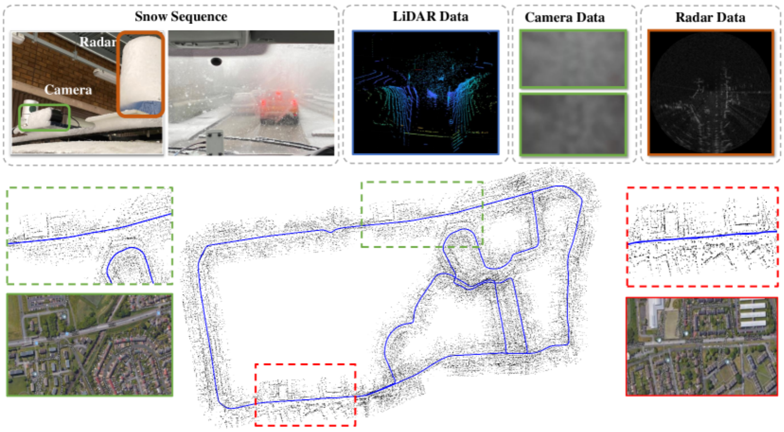


Figure 2.3: Multi sensor SLAM system based on radar

Multi sensor SLAM

Multi sensor SLAM is simply a category for the SLAM solutions which make use of several 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 a independent SLAM system.

A few common multi sensor SLAM sensor configurations include:

- camera + IMU(Visual Inertial SLAM)[6] Cameras have good spatial information but low temporal resolution, while IMUs have high temporal resolution. When IMU accumulates too much drift, the cameras help correct it, and the IMU helps when vision is low. Main disadvantage is the high sensitivity to visual degradation(fog, dark areas, etc).

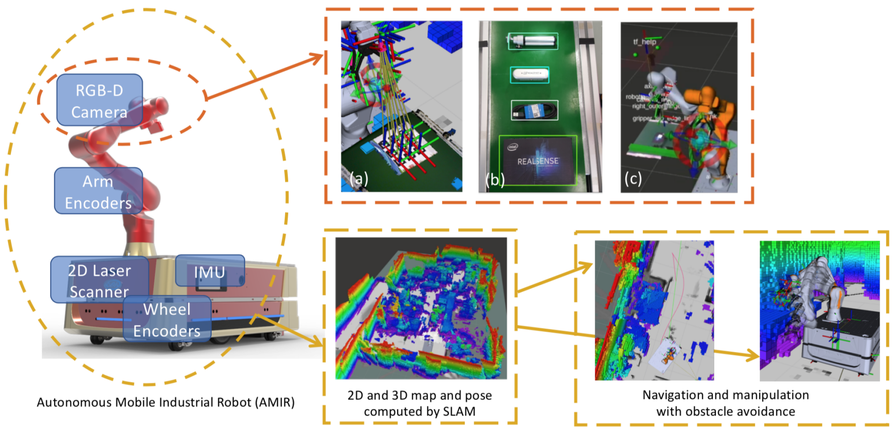


Figure 2.4: Multi sensor SLAM system for an industrial robot

- Lidar-Inertial SLAM (LI-SLAM)[7]. 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).

- Visual-Lidar-Inertial SLAM (VLI-SLAM)[8]. In this approach, each sensor fills the gaps of the other, combining rich visual features, accurate range data, and motion tracking. Its main advantage lie in the high robustness and accuracy across varying environments. However, It requires a complex calibration and synchronization process, is expensive and requires more computational power than more simple 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:

- Absolute Trajectory Error(**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 RMSE, as follows:

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

- Relative Pose Error(**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 ATE, the formula for the RMSE of the RPE is as follows:

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

- Resource usage(memory, CPU) - taking into account resource utilization is an important aspect when comparing different SLAM methods, due to the trade-offs between accuracy and memory usage. In some applications, it might not be worth it to use a more computationally intensive(but more accurate) SLAM solution.

- Benchmarking - 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. As for evaluation frameworks, one of the most widely used is EVO, which provides error metrics and visualization tools, so as to better compare the performance of different SLAM solutions.

2.2 Hyperparameter optimization techniques

Hyperparameter 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 predefining a parameter space and drop 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 HPO, which will be used as a baseline on this thesis' 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 Hyperparameter Optimization(HPO). It simply consists of an exhaustive search and evaluation of all possible hyperparameter combinations within a predefine parameter space[9]. In spite of the development of more specific algorithms in recent decades, Grid Search remains popular due to its simple implementation and trivial paralelization[9]. The main drawback is its computational cost, due to the curse of dimensionality being a serious problem in models with large numbers of hyperparameters[9]. 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.

By looking at **equation 2.1**, it becomes clear this algorithm is not very scalable, due to the rapid increase in the number of configurations, which makes this the main hurdle in search based approaches. One way of getting around it is 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.

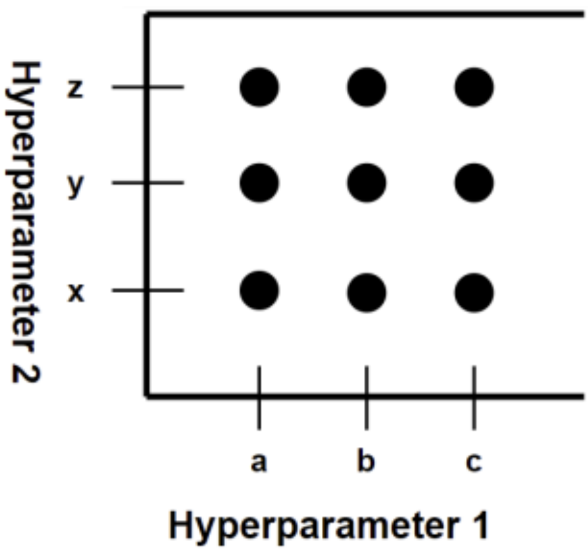


Figure 2.5: Graphical illustration of parameter space exploration in grid search

Random Search

Random Search is a variation of Grid Search. It randomly samples configurations in the aforementioned parameter space[10]. Both Grid Search and Random Search are very similar implementation wise, with one major difference: Random Search requires a budget be specified, whether it is time, number of configurations, etc.[11] The main advantage Random Search has over Grid Search is the faster convergence over a local or global optima[11], although this advantage gets slimmer the larger the parameter space.

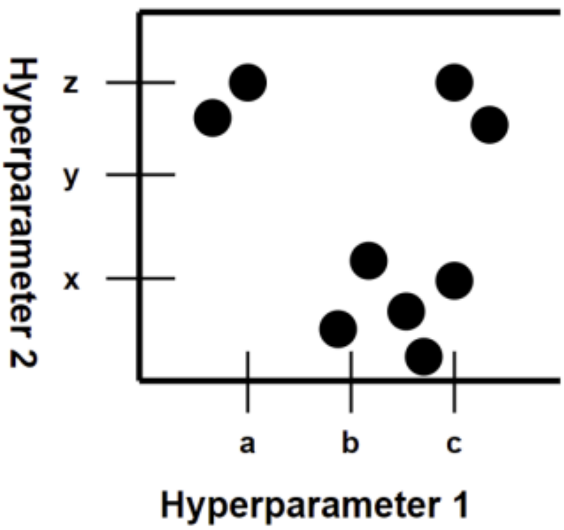


Figure 2.6: Graphical illustration of parameter space exploration in random search

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 unimportant and it is therefore treated as a black box. These kind of HPO techniques favor more complex optimization problems and clarify the relationship between algorithm performance and hyperparameter settings, which might prove to be a good option to pre select the most important hyperparameters to optimize when there are dozens or hundreds of parameters to optimize.

Bayesian Optimization

Bayesian Optimization (BO) is a probabilistic model-based approach that optimizes black box functions that are expensive to evaluate[12]. It's particularly useful when the objetive function lacks an analytic expression and its evaluations are very expensive, which is the case for SLAM methods.

This method has two components:

Algorithm 1 Bayesian Optimization

for $t = 1, 2, \ldots$ **do**
  Find $\mathbf{x}_t$ by optimizing the acquisition function over the GP: $\mathbf{x}_t = \mathrm{argmax}_{\mathbf{x}}\, u(\mathbf{x}|\mathcal{D}_{1:t-1})$.
  Sample the objective function: $y_t = f(\mathbf{x}_t) + \varepsilon_t$.
  Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ and update the GP.
end for

Figure 2.7: The Bayesian Optimization algorithm.

- Surrogate model that aproximates the objetive function, and is much easier to evaluate than the actual function[13]. Although multiple models can be used for this purpose, the most commonly used(and the standard) is Gaussian Processes(GP)[10].
- An aquisition function, that measures the value generated by evaluating the objective function at a given point, and guides the search for the next point to evaluate[13]. It balances exploration(areas with high uncertainty) with exploitation(areas with high predicted performance). A few common aquisition functions include Expected Improvement(EI), Probability of Improvement(PI) and Upper Confidence Bound(UCB).

BO uses the following iterative process to optimize the objetive function:

1. Select the next point to be sampled with the aquisition function.
2. Evaluate the true function $f(x_1, x_2 \ldots x_n)$ 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[10].

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[14], such as APE or RPE in the case of SLAM optimization.

Population-based approaches are stochastic in nature. The specific techniques used in this thesis' work favor early exploration and become more exploitative over the course of the execution of the algorithm.

Simulated Annealing

Simulated Annealing (SA) is an optimization technique that mimicks the physical process of heating a metal and then cooling it slowly[15]. 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 and maximizing exploitation.

The method starts by assigning a single value to all hyperparameters, one that's supposed to be high enough to allow compreensive random search over the parameter space[16]. Then, it makes small changes to one parameter at a time and evaluates this new configuration, called a neighbor[15]. The aceptance of the neighbor as being the better solution depends on a probabilistic distribution, which in itself depends on the temperature and the difference between the current solution's and the neighbor solution's evaluation, like so:

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

where T is the current temperature and ΔE 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 α 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[16]. 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.

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[17]. Similar to Simulated Annealing, this technique requires a budget be specified(execution time, number of iterations, etc.).

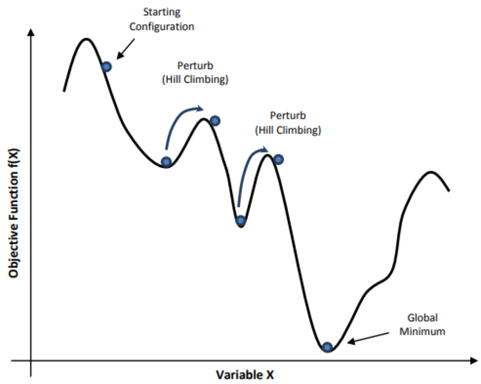


Figure 2.8: A graphic representation of the simulated annealing algorithm

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[18]. The hyperparameter configurations used in this method can be generated in a variety of different ways. One can define a parameter space similar to the one used in search-based approaches and use either Grid Search or Random Search to generate the desired number of configurations. There is also the possibility of manually selecting both the hyperparameters to optimize and their values. This latter approach, combined with prior knowledge-based sampling, might prove to be a better stretegy for selecting and generating the starting configurations, due to the sheer number of hyperparameters in most SLAM methods.

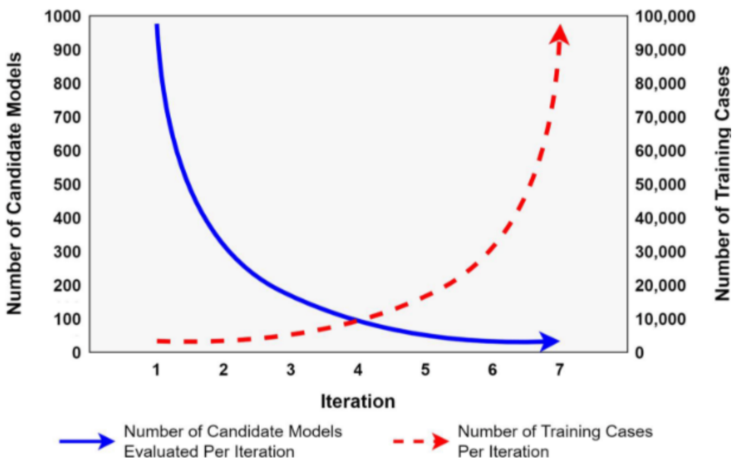


Figure 2.9: A graphic representation of the Successive Halving algorithm

Hyperband

Hyperband is a more efficient technique that builds on the Successive Halving algorithm[13?]. By dynamically allocating computational resources, Hyperband quickly identifies promising configurations and discards ill-performing ones early on, saving both time and CPU time.

Hyperband works as follows:

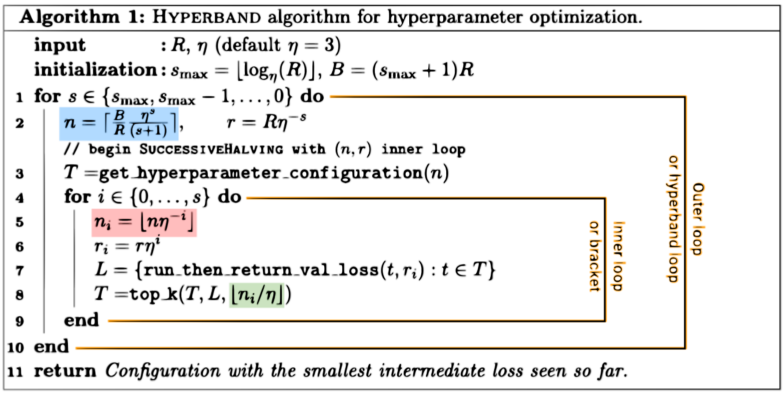
1. Define a total Budget B and the proportion of configurations to evaluate at each rung, η. Also, a maximum budget R for a single rung is required[19].

2. Generate a large number of configuration[19?]s.

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 / η configurations remain, instead of halting when only the best configuration remains. Then, increase the budget for the remaining configurations[19].

4. Repeat step 3 until maximum budget R is reached or all configurations are exhausted, meaning only the best one remains[19].

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, η (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: HB algorithm described in a more systematic way

2.2.4 HPT in SLAM

When applying HPT algorithms to fine tune a SLAM method's parameters, there are a few things to keep in mind.

First, not all parameters are tunable. Better yet, some parameters have fixed or default values and are treated as constants during the tuning process, such as a camera's intrinsic parameters in VSLAM, or an IMU's noise and Bias models in Visual-Inertial SLAM. This is an important aspect to keep in mind, especially when applying search based approaches, as it may provide a way to significantly reduce the number of configurations to be executed.

As it pertains to the effectiveness of tuning strategies, search based methods are usually considered baseline methods in research, and not very effective when dealing with very high dimensional parameter spaces, as due to the high stochasticity of grid and random search. In this regard, simulated annealing achieves better results. But in order to obtain near optimal results

2.3 Related Work

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

On search-based approaches, there has been some research on the topic. Particularly, authors have hand-selected and manually searched(brute force) the parameter space of a g-mapping SLAM solution[**?**]. In this specific case, four parameters were tuned: linear_update_step, angular_update_step, quantity of particle, and resampling threshold. 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[20]. In all 10 sequences, measured ATE significantly decreased, showing the main advantage of this simple tuning approach. It is also worth noting that while search based approaches are more suited for a very reduced number of parameters, it is best to use grid search than manual search(brute force). While both papers presented previously show that, with good knowledge of the system, one can significantly reduce the parameter space down to the very few and impactful hyperparameters and obtain a near optimal configuration with both approaches.

As far as model based approaches go, bayesian optimization 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 Odometry algorithm, without knowledge of the inner workings of the system, e.g the system is treated as a black box[21]. Although data augmentation was used to prevent overfitting, it was still possible to observe a noticeable decrease in the odometry drift error(in most tested sequences). Another article, although outside the scope of this thesis, used Bayesian Optimization to fine tune the extrinsic parameters of a camera in a Visual Inertial SLAM sytem[22].

With regards to population-based approaches, there isn't much research in SLAM itself. However, one article explored the application of both a particle swarm algorithm and an evolutionary algorithm to optimize the parameters of an RGB-D Visual Odometry system [14]. These metaheuristics managed to reduce the execution time from several days to just a few hours, which is its main advantage. However, it is also concluded that these optimized parameters also generalize to other sequences, so long as camera dynamics(its intrinsic parameters) and execution environment stays the same.

Article(s)	SLAM method	optimization method	Results
Z. Zheng[20]	Feature-based Visual Odometry	Grid Search	Across 8 optimized sequences, average ATE decreased, on average, 64.68%
. A. Putra and P. Prajitno[23]	G-mapping SLAM	Brute Force	Navigation time of a predefined path was reduced from 32 to 25 seconds
K. Koide et al[21]	Lidar Odometry	based on Bayesian Optimization	**With data augmentation**, in both tested environments, both components(translational and rotational) of the drift error decreased by at least 9.8%.
A. Kostusiak and P. Skrzypczyński[14]	RGB-D Visual Odometry	Particle Swarm Optimization	Both ATE and RPE decreased by well over 50%, and optimized parameters generalize well to other sequences
A. Kostusiak and P. Skrzypczyński[14]	RGB-D Visual Odometry	Evolutionary Algorithm	optimization duration about 5x shorter than Particle Swarm Optimization (PSO),

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 HPT. Most notably:

- Lack of diversity in optimizing strategies. As presented earlier, most of the research use either a search based approach or a model-based approach(similar to BO). In the case of the former, it is understandable. For SLAM systems where there is a high degree of knowledge of the inner working of the sensors and how different parameters affect the performance of the solution, it is possible to manually set and stick with default values for some of the hyperparameters, and can then apply a simple baseline(search-based) approach to the tuning of the remaining hyperparameters. As for the latter, it is 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 [14], there isn't much research on population-based approaches to optimizing SLAM, whether variations of particle swarm algorithms, evolutionary algorithms, or more generally speaking, meta heuristics.

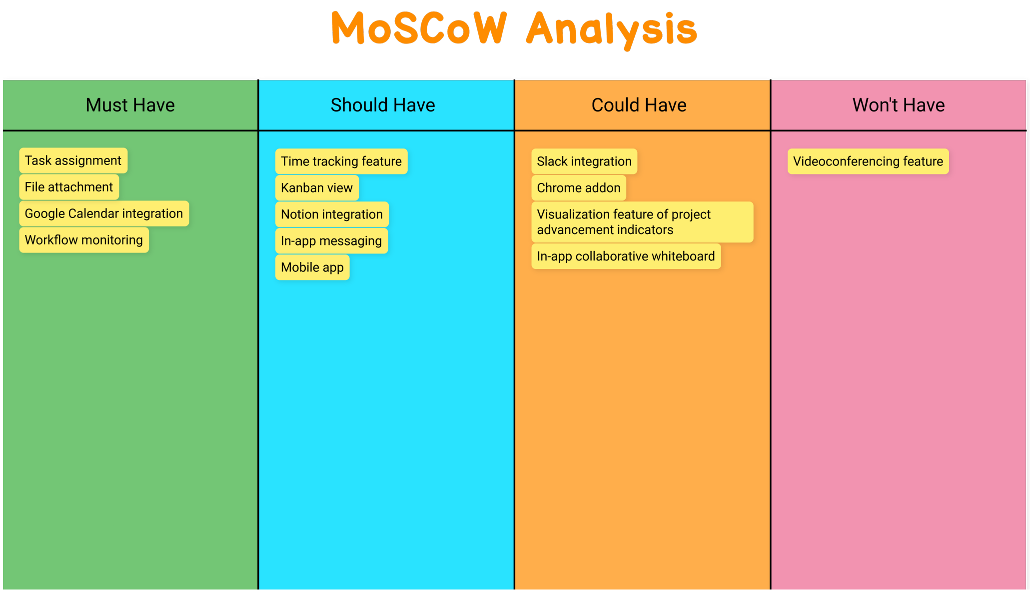
- Lack of a framework to automatically compare and optimize SLAM method, allowing for an even playing field in SLAM research. although RUSTLE, the tool which will built and improved upon during the course of this thesis, already allows for a somewhat organized approach to testing and optimizing complete SLAM solutions, it lacks any automatic optimizing feature, meaning the only way to currently approach HPT in RUSTLE is a brute force algorithm.

2.3.2 Statement of contributions

Given that academic research often focuses on a single optimization method for a particular SLAM system for an even more specific use case, the main contribution of this thesis is to widen the spectrum of optimization approaches to SLAM, as well as allow for the automatic testing(optimization) of many different SLAM solutions in an organized and systematic manner.

3 Methodology

This chapter will explain the methodology and requirements for the SLAM tuning module, developed for RUSTLE. For each implemented tuning technique, a high level overview of its architecture is provided. Test settings, including test algorithms and dataset choices, are also justified.



3.1 RUSTLE

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, stream-lined 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 more complete assessment of the algorithm's performance.

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

Algorithm executions are run as tests. Each test's configuration is passes as an YAML file, and 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 one wants to execute.

As for language choice, Rust was chosen mostly due to simplicity of development and the ease with which one can write highly performant code. Other languages, such as c++ and python, also have libraries for working with Docker and ROS, as well as libraries for database hosting. The difference with Rust in this regard is that to use these libraries it is only needed to include them as dependencies, and use them in the codebase, as the download, compilation and linking of these libraries is all automatically handled by Rust's build tool, cargo. As for performance, Rust is known for its compile-time enforced memory safety(at the cost of some control over memory), as well as built in type system guarantees, which prevent common concurrency issues like data races.

3.2 Development

Development follows an incremental approach, focusing on small incremental features, well thought and tested. Each big feature, such as random search tuning, is divided into smaller "implementable" sub-features. The reason for this is to make it easier to reason about how to implement the feature, as well as, during development, to make it easier to test each sub-feature.

3.3 Functional requirements

The system must allow users to define all relevant settings for the tuning process using a YAML configuration file. This configuration file will serve as the main interface for specifying the tuning method, SLAM algorithm, and dataset to be used. In addition, it will enable the user to indicate which parameters are to be optimized, with one or more values per parameter. Additionaly, the code must ensure type safety, meaning, for example, that a parameter which is supposed to be a String cannot have a u64 passed as a value. Also, the user must have the option of specifying only the parameters he wants to optimize.

For grid and random search tuning methods, the configuration file must support flexible representations of parameter values. Each tunable parameter can be defined as a single value, representing a fixed setting, or as an array containing multiple candidate values to be tested. For numerical parameters(integers and floats), a third specification pattern must be supported: a linearly spaced vector. This vector will be described by a three-element array in the form [**start, step, number_of_elements**], which the system will interpret to automatically generate a sequence of values. This feature will simplify the definition of evenly spaced search spaces for numerical parameters and ensures consistency across different tuning runs. It also makes it significantly less cumbersome to define large sequences of values.

Every tuning algorithm has, besides the parameter space, some method specific hyperparameters that control its behavior, such as the budget(grid search) or the cooling factor(SA). Each of these parameters must have a default value, so the user doesn't need to fine tune the tuning algorithm's behavior. As for the budget **B**, a common hyperparameter used in tuning algorithms, it must be either time based, or iterations based(maximum number of iterations to run).

For easier development and integration with the overall code structure of RUSTLE, the tuning of an algorithm's hyperparameters is treated as a test. Each configuration is run as a **simple test**, already defined in the code, changing hyperparameters between iterations. This methodology will allow for greater re-usage of already written code, saving on development time.

When running a tuning test, each configuration and its metrics(results) must be stored for later processing. The tuning method's hyperparameters must be stored as well, mostly for benchmarking purposes.

For benchmarking different tuning algorithms, there must be a command as simple as **tune benchmark -algo xxxxx -dataset xxxxx**, which runs a few preselected tuning algorithms, and displays a few performance metrics.

Whenever possible, configurations should be run concurrently.

The code written should be future proof, allowing for easier future integrations of more tuning methods into the software.

3.4 Non Functional requirements

3.5 Testing

3.5.1 Metrics

Write something about evalutation functions to compare configurations and obtain the better one.

3.5.2 Algorithms

Write something about the algorithms tested(point-lio, ig-lio, etc), but emphasize the tuning algorithms apply to any SLAM method.

3.5.3 Datasets

Mention the datasets used during development, but stress that, as in the SLAM methods used, the tuning algorithms generalize to any dataset.

Bibliography

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

[2] 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.

[3] 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.

[4] Z. Hong, Y. Petillot, A. Wallace, and S. Wang, "Radar slam: A robust slam system for all weather conditions," *arXiv preprint arXiv:2104.05347*, 2021.

[5] M. Holder, S. Hellwig, and H. Winner, "Real-time pose graph slam based on radar," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1145–1151, IEEE, 2019.

[6] 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.

[7] 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," 2024.

[8] K. Liu, "A lidar-inertial-visual slam system with loop detection," 2023.

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

[10] A. Bissuel, "Hyper-parameter optimization algorithms: a short review," *Criteo tech blog, Medium. Disponível em: https://medium. com/criteo-labs/hyper-parameter-optimizationalgorithms-2fe447525903. Acedido em*, 2020.

[11] 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.

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

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Z. Zheng, "Feature based monocular visual odometry for autonomous driving and hyperparameter tuning to improve trajectory estimation," vol. 1453, no. 1, p. 012067, 2020.

[21] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Automatic hyper-parameter tuning for black-box lidar odometry," pp. 5069–5074, 2021.

[22] Z. Chen, "Visual-inertial slam extrinsic parameter calibration based on bayesian optimization," 2018.

[23] 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,” pp. 148–153, 2019.

Appendix A

Sample Appendix

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.