TÉCNICO LISBOA

Automatic Parameter Tuning of Algorithms using Optimization

João Pedro Santos Azevedo

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Rodrigo Martins de Matos Ventura
Oscar Lima Carrion

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Jorge Nuno de Almeida e Sousa Almada Lobo

November 2017

Acknowledgments

First and the most importantly, I would like thank my family including my girlfriend for the love and continuous support they have always offered until today. A special thanks to my mother, father and grandparents for always giving me the very best they can offer.

Also a special thanks to professor Rodrigo Ventura for guiding me through this thesis, and for the opportunity of being part of the SocRob team that I am very grateful for. I also have to thank Oscar Lima for the patience and formidable provided lessons that were very useful for this thesis.

Resumo

O algoritmo Adaptive Monte Carlo Localization é uma famosa abordagem para a alcançar a localização de robôs usando um filtro de partículas. O AMCL tem alguns parâmetros que são configuráveis. Estes parâmetros podem melhorar a sua performance em troca de um aumento do consumo de recursos computacionais. Nesta tese é realizada a otimização com múltiplos objetivos dos parâmetros do Adaptive Monte Carlo Localization, usando o algoritmo Predictive Entropy Search for Multi-objective Bayesian Optimization. Para tal, é estendida uma *framework* usada para otimizações sobre apenas um objetivo. Com esta extensão, é possível ter como resultado final uma fronteira de Pareto, onde o utilizador poderá escolher os recursos desejados e obter a configuração de parâmetros ótima para uma situação em particular. Um dos critérios usados na otimização com múltiplos objetivos é o erro de posição do robô. Como o seu cálculo normalmente requer *hardware* de rastreamento caro, é proposto e validado um método para estimar o erro de posição e orientação 2D do robô. Este método usa o algoritmo Iterative Closest Point.

Palavras-chave: Optimização com Múltiplos Objectivos, Optimização dos parâmetros do algoritmo Adaptive Monte Monte Carlo Localization,Iterative Closest Point, Predictive Entropy Search

Abstract

The Adaptive Monte Carlo Localization algorithm is a well-known approach for performing the localization of a robot using a particle filter. The Adaptive Monte Carlo Localization have some parameters which boost the performance but at the expense of an increase in computational resources. In this thesis, the multi-objective optimization of the Adaptive Monte Carlo Localization algorithm parameters was performed, using a state of the art algorithm, the Predictive Entropy Search for Multi-objective Bayesian Optimization. In order to do so, a framework for single objective optimization was extended. The added output of this framework is a Pareto front where the user can select the desired resources and get the parameter set that is optimal for that particular setup. One of the objective functions used in the multi-objective optimization is the position error of the robot. Since its computation normally requires expensive tracking hardware, a method for estimating the robot's two dimensional pose error using the Iterative Closest Point algorithm, is proposed and validated.

Keywords: Multi-objective optimization robotics, Adaptive Monte Monte Carlo Localization parameter optimization, Predictive Entropy Search, Iterative Closest Point

Contents

List of Tables

List of Figures

Glossary

AMCL	Adaptive Monte Carlo Localization
CMAES	Covariance Matrix Adaptation Evolution Strategy
CPU	Central Processing Unit
EBG	Electromagnetic Band Gape
EGO	Efficient Globa Optimization
EHI	Expected Hyper-volume Improvement
ICP	Iterative Closest Point
KLD	
MCL	Monte Carlo Localization
MODES	Multi Objective Derandomized Evolutionary Strategy
MOEA/D	Multi Objective Evolutionary Algorithm based on Decomposition
MOEA	Multi Objective Evolutionary Algorithms
MOO	Multi Objective Optimization
NSGA	Non-dominated Sorting Genetic Algorithm
PDF	Probability Density Function
PESMO	Predictive Entropy Search for Multi-objective Optimization
PSO	Particle Swarm Optimization
ParEGO	Pareto Efficient Globa Optimization
RMSE	Root Mean Square Error
RMSE	Root Mean Squared Error
ROS	Robotic Operation System
SEPTOP	Solar Electric Propulsion Trajectory Optimization Program
SMAC	Sequential Model-based Algorithm Configuration
SVD	Singular Value Decomposition

Chapter 1

Introduction

1.1 Motivation

It is no longer science fiction, the idea that domestic robots will take part in our daily lives in a near future. The industrial and academic world are in rush to develop robots capable of helping humans in daily house keeping tasks. From the extraordinary amount of developed algorithms, most of them contain configurable parameters, that in most cases, are manually tuned by the hands of an expert technician. The problem occurs when these algorithms contain a significant amount of configurable parameters, or when the output of these algorithms is not consistent (meaning that repeated evaluations with the same parameters do not yield similar results), or even when its optimization requires expensive hardware not easily accessible to to general public of robotics enthusiasts. Multi objective optimization (MOO) algorithms answer to this problem, by generating a set of optimal solutions that allow the decision maker to analyze the trade offs between objectives and select a final preferred solution. In this case, each one of these optimal solutions will be a set of parameters that will optimize the robot's functionality. Multi objective optimizers have been used in the past in many fields of study, from the optimization of electromagnetic bandgaps (EBGs) [1] to the minimization of the prediction error and prediction time of neural networks [2]. It is interesting to study how these multi objective optimizers can be applied to address the parameter optimization problem.

1.2 Problem Statement

The main goal of this thesis is to optimize the parameter configuration of algorithms exploring its trade offs in the objective functions, by using MOO algorithms. These algorithms optimize problems involving multiple incommensurable and conflicting objectives. This problem aggravates when the objective functions are black-box functions or when they are in some sort of way expensive to evaluate. This is the case of the Adaptive Monte Carlo Localization (AMCL) algorithm, a stochastic nature algorithm, where to perform a reliable evaluation, the time needed is in the order of minutes. It is also not possible to perform more than one evaluation at one time. Since that the implementation of the AMCL algorithm we

want to optimize has 47 parameters, 22 of them configurable, the search space of the chosen algorithm is extremely high. Due to this feature, it is expected that the objective functions might be highly multi modal and rough. The majority of the MOO algorithms has not been tested under this situation, so it is also this thesis aim to analyze their performance under this stressful problem. The chosen objective functions were two: the position error given by the AMCL algorithm and the Central Processing Unit (CPU) time consumed by the AMCL process. These criteria have an opposite relationship, meaning that a higher precision of the AMCL estimated position will result in a higher CPU time consumption. For localization purposes, the user always wants to have a precise localization of its robot, and even if it is the case where he is not interested in lowering the CPU consumption, discovering the Pareto frontier of these objective functions will at least give the possibility to the user to have the precision he requires with the lowest CPU consumption. When this idea is applied for all modules of the robot, perception and manipulation for instance, a great save in CPU consumption might be achieved. This thesis contributes by extending a framework for optimizing robotic algorithms developed in [3], now taking into account more than one evaluation criteria. The selected algorithm to optimize is the AMCL algorithm, that recently has also been optimized under one objective [3], was the Predictive Entropy Search for Multi-objective Bayesian Optimization (PESMO). PESMO is a state of the art Bayesian method with proved efficiency when used for optimizing functions expensive to evaluate [2]. The framework can be easily be used to optimize other robotic algorithms with any other optimization algorithm. In this thesis, as output of this framework, it is desired a trustful Pareto front. A Pareto front is merely a trade off between the the desired objective functions, allowing to the user to chose a parameter set that best fits his needs. Experimental results of the MOO applied to AMCL are also provided.

In the second aim of this work, the important aspect is to remove the need of a ground truth sensor in order to optimize the AMCL algorithm. Generally, these ground truth sensors may be very expensive for regular users and robotics enthusiasts. In this thesis is proposed a method that uses the recorded laser scan data and the map of the environment to estimate the two dimensional pose error of the robot, allowing users that, for instance, have home made robots using for localization the AMCL algorithm, to optimize their location precision without a try and error process and without the use of expensive and many times unavailable tracking hardware. This can be achieved with an Iterative Closest Point (ICP) approach where the laser sensor data is compared to the map data to get an estimate of the pose error. The purpose of this estimate is to be used as evaluation function for the AMCL pose error, dismissing the use of expensive robot tracking hardware and allowing robotic hobbyists to optimize their AMCL algorithm inexpensively. This thesis contributes with the development of the proposed method, as well as experimental results for its validation.

1.3 Related Work

For more than three decades, MOO algorithms have been studied and successfully applied in many situations. In swarm and evolutionary computation, Nondominated Sorting-based Genetic Algorithm (NSGA) [4] and its second version NSGA-II [5] have been a reference in the field. The multi objective

evolutionary algorithm (MOEA) is based in a nondominated sorting approach using the fittest population of candidate solutions to achieve a Pareto front. In [6], we find MOEA/D, another MOEA, now based in decomposition. It decomposes a multiobjective optimization problem into a number of scalar optimization subproblems and optimizes them simultaneously. An extension to MOEA/D appears in [7], where the covariance matrix adaptation evolution strategy (CMAES) [8] is integrated into MOEA/D as the MOED/D-CMAES. In [1], there is Borg, one of the most powerful MOEAs, that combines many optimization techniques as the ϵ-dominance, ϵ-progress and randomized restarts into one optimization framework. There is also hybrid algorithms, as ParEGO [9], an extension of the single objective efficient global optimization (EGO) algorithm [10], where a combination of evolution strategies and of a model based approach to model the data are used to achieve great results in multi objective problems. In Bayesian optimization, we find ϵ-PAL, an active learning approach, that models the objectives functions as draws from a Gaussian process distribution. It also introduces interesting approaches on how to chose the promising points that might belong to the Pareto frontier, and also how to control this prediction accuracy. Still in Bayesian based approaches, the Predictive Entropy Search for Multi-objective Bayesian Optimization (PESMO) [2], is a very strong approach for expensive objective functions. Its strategy consists on the reduction of the Shannon entropy of the posterior estimate of the objective functions. It also introduces the concept of decoupling the objectives, allowing for the evaluation of one objective function separately.

These and other MOO algorithms have been used in many fields of study. In the aerospace industry, [11] used a Multi Objective Derandomized Evolutionary Strategy (MODES) to optimize an aircraft wing design. An Aircraft Control System Design was also optimized using a multi-objective evolutionary algorithm, presented in [12]. In [13], is developed an hybrid algorithm, combining NSGA with the Solar Electric Propulsion Trajectory Optimization Program (SEPTOP) in order to optimize low-thrust spacecraft trajectory profiles. In control theory, [14] used NSGA-II to perform a tri-objective optimization of an industrial greenhouse climate control system. In the electromagnetic and antennas field of study, [15] using a Multi objective Particle Swarm Optimizer, designing a time-delay equalizer metasurface for an EBG resonator Antenna. Borg, CMAES and MOEA/D, are all three also used to optimize an EBG structure. Other algorithms as ParEGO, ϵ-PAL and PESMO were also successfully used in test functions specifically designed to test MOO algorithms.

This thesis multi objective algorithm of choice was PESMO. This choice was made based in the following criteria:

- In contrast to most of the previously mentioned algorithms, PESMO is a model based approach and not an evolutionary algorithm, having a stronger statistical approach.
- PESMO is designed to perform a small amount of evaluation functions, specially adequate for cases where the evaluation is expensive. In contrast, evolutionary algorithms require a much greater amount of function evaluations to yield good results.
- PESMO is tested and compared against ParEGO, SMSego [16], EHI [17] and SUR [18] outperforming all in every study case.

- The computational cost of PESMO grows linearly with the number of objectives, having a lower computational cost than other model based related methods from the literature.

The algorithm we want to optimize is AMCL, implement in the ROS infrastructure[1] and has previously been optimized for a single objective case in [3]. In this case, SMAC [19], a single objective optimization method was used to optimize the position estimate of the AMCL algorithm. This thesis is in fact inspired in [3], aiming to be its extension. Besides the previously mentioned work, in [20], a single objective parameter optimization algorithm, the Particle Swarm Optimization (PSO) [21], was also applied to Monte Carlo Localization. Lastly, It is worth mentioning that multi-objective optimization methods can also be applied to the aerospace engineering industry. For instance, one could be interested in optimizing an aircraft structure in terms of thrust to drag ratio, or optimizing a wing's lift to drag ratio or even a trade off between the aircraft's volume and its Radar Cross Section (RCS).

1.4 Thesis Outline

After a brief summary in the present chapter of the problem statement and goals of this work, it is done in chapter 2, a review of the theoretical tools needed to the development of the suggested framework. The main theoretical tools are the AMCL, ICP and PESMO algorithms. Regarding the implementation of the suggested work, a system architecture full description is done in chapter 3. In chapter 4 is described how the experiment was realized. After that, all the results regarding the acquisition and validation of the Pareto front, as well as the validation of the proposed method for estimating the AMCL position error are exhibited and analyzed under chapter 5. Finally, conclusion are drawn in chapter 6, as well as suggestions for future work.

[1] http://wiki.ros.org/amcl

Chapter 2

Theoretical Background

2.1 Monte Carlo Localization

An essential part of robotics is the ability of the robot to find its location on a known map of the environment, using range sensor data and odometry sensor data. The problem can be considered as a Bayesian filtering problem and in order to solve it, the Monte Carlo Localization (MCL) algorithm is used. MCL estimates the pose (position and orientation) of the robot through a particle filter, where each particle represents a likely state, i.e, a possible robot pose. For 2D localization cases, the state $\mathbf{x}$ of the robot corresponds to a Cartesian reference frame position $[x, y]$ and an angular orientation θ, i.e., $\mathbf{x} = [x, y, \theta]^\intercal$.

2.1.1 Markov Location Algorithm

MCL is based in the basic Markov location algorithm [22]. Let $\mathbf{x_k}$ be the state of the robot at time-step k and $Z^k = \{z_k, i = 1, \ldots, k\}$ the sensor measurements up to the current time-step k. The idea is to construct the posterior density $p\left(\mathbf{x}_k | Z^k\right)$ of the current state conditioned on all measurements[23]. This is accomplished in two steps, the predictions step and the update step.

Prediction Step

In the prediction step, only a motion model is used to construct a predictive probability density function (PDF) $p\left(\mathbf{x}_k | Z^{k-1}\right)$ to predict the current state $\mathbf{x}_k$. The environment is assumed to be Markovian, meaning that the past movement measures are independent from the future ones, so the current state $\mathbf{x}_k$ will only depend on the previous state $\mathbf{x}_{k-1}$ and the motion applied $\mathbf{u}_{k-1}$[23]. The motion model is given by a conditional density function $p\left(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_{k-1}\right)$ specifying the probability that a movement or input action $\mathbf{u}_{k-1}$ executed at state $\mathbf{x}_{k-1}$ results in state $\mathbf{x}_k$[22].

Update Step

In this step, sensor reading are taken into account to obtain the posterior density $p\left(\mathbf{x}_k | Z^k\right)$. Assuming again a Markovian environment, and that the measurement model is given by a likelihood $p\left(\mathbf{z}_k | \mathbf{x}_k\right)$ specifying the probability of perceiving $\mathbf{z}_k$ when the robot is at state $\mathbf{x}_k$, the posterior density $p\left(\mathbf{x}_k | Z^k\right)$

is:

$$p\left(\mathbf{x}_k|Z^k\right) = \frac{p\left(\mathbf{z}_k|\mathbf{x}_k\right)p\left(\mathbf{x}_k|Z^{k-1}\right)}{p\left(\mathbf{z}_k|Z^{k-1}\right)} \tag{2.1}$$

The process is then repeated recursively and as the robot moves, converging the particles towards the correct pose of the robot.

2.1.2 MCL Algorithm description

The posterior PDF $p\left(\mathbf{x}_k|Z^k\right)$ can be represented in many ways, some of them described in [23]. MCL algorithm uses a sampling-based method, that consists in, at each time-step k, *represent the posterior density* $p\left(\mathbf{x}_k|Z^k\right)$ *by a set of particles* $S_k = \{s_k^i, i = 1, ..., N\}$ *that are randomly drawn from it*[23]. Each particle represents a possible state of the robot. Reformulating the two steps mentioned before we have:

Prediction Step

In the previous iteration, the posterior PDF $p\left(\mathbf{x}_{k-1}|Z^{k-1}\right)$ is represented by a set of particles $S_{k-1} = \left\{s_{k-1}^i, i = 1, \ldots, N\right\}$. The motion model is applied to each particle of S_{k-1} by drawing a sample from the density $p\left(\mathbf{x}_k|s_{k-1}^i, \mathbf{u}_{k-1}\right)$, to acquire a new particle $s_k^{'i}$ and constructing a new particle set $S_k^{'}$ that approximates a random sample from the predictive PDF $p\left(\mathbf{x}_k|Z^{k-1}\right)$. *We are able to do this because of the duality between the samples and the density from which they are generated*[23].

Update Step

To take sensor reading $\mathbf{z}_k$ into account, each particle $s_k^{'i}$ in $S_k^{'}$ is weighted by the likelihood $p\left(\mathbf{z}_k|s_k^{'i}\right)$. In words, by the probability of measuring $\mathbf{z}_k$ given $s_k^{'\,i}$. Then we draw, for $i = 1, ..., N$, N samples s_k^i to obtain S_k. This new set S_k approximates a random sample from the posterior density $p\left(\mathbf{x}_k|Z^k\right)$ because only the samples $s_k^{'i}$ with high likelihood are selected.

2.1.3 Adaptive (or Kullback-Leiber Distance Sampling) Monte Carlo Localization

The main idea of the Kullback-Leiber Distance (KLD) Sampling is to adapt the number of samples over time increasing the efficiency of the MCL algorithm. When the robot location is uncertain, a large number of particles is necessary to cover all the possible states. However, when the robot position is known with low uncertainty, a such large number of particles is unnecessary and inefficient. KLD-Sampling bounds the error introduced by the sampling-based approximation of the true posterior density $p\left(\mathbf{x}_k|Z^k\right)$ of the particle filter. This bound is achieved by calculating the number of samples needed so that, with probability $1 - \delta$, the Kullback-Leiber distance between the maximum likelihood estimate (MLE) of the sampling-based estimated posterior density and the actual true posterior density $p\left(\mathbf{x}_k|Z^k\right)$, does not exceed a user defined threshold ε. The number of samples n is:

$$n \doteq \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3 \tag{2.2}$$

Where $z_{1-\delta}$ is the upper $1-\delta$ quartile of the standard normal $N(1,0)$ distribution and k is the number of different bins with support (if its probability is above a threshold) of the discrete distribution from where the n samples are drawn (the posterior distribution in this case). The derivation of n and how the previous result can be applied to the MCL algorithm can be found in detail in [24].

2.2 Iterative Closest Point

The Iterative Closest Point (ICP) algorithm is used to geometrically match and align two 3D data structures and was first introduced in [25]. This alignment process is also denoted as registration. Basically, each point of the data set that we want to align, also called source data set, is paired with the closest point of the other data set that we wish to align to, denoted as the model or target data set. An error metric is used to determine how good the alignment is. The process is iteratively repeated until converging and stopping at the desired stop criteria, outputting an optimal rotation and translation that minimizes the error metric between the model data set and the registered data set. The original ICP algorithm [25] [26] can be used in the following representations of data sets: (a) point sets (or point clouds) (b) line segment sets (c) implicit curves (d) parametric curves (e) triangle sets (f) implicit surfaces (g) parametric surfaces. Since in this work, the ICP algorithm will be used on point clouds, only the method for computing the closest point to a given point set is described. For more information about computing the closest point to a given point of the other possible representations of data sets, the reader can consult [25]. The euclidean distance $d(\vec{p}_1, \vec{p}_2)$ between two points $\vec{p}_1 = (x_1, y_1, z_1)$ and $\vec{p}_2 = (x_2, y_2, z_2)$ is:

$$d(\vec{p}_1, \vec{p}_2) = \|\vec{p}_1 - \vec{p}_2\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{2.3}$$

Let P be a point set um N_p points denoted $\vec{p}_i$, for $i = 1...N_p$. Then, $P = \vec{p}_i$ The distance, $d(\vec{p}, P)$ between a point $\vec{p}$ and the point set P is the minimum of the distances to the points of the point set P, i.e., the distance to the closest point of P:

$$d(\vec{p}, P) = \min_{i \in 1,\ldots,N_p} d(\vec{p}, \vec{p}_i) \tag{2.4}$$

For finding the optimal rotation quaternion $\vec{q}_R = [q_0 q_1 q_2 q_3]^\intercal$ and translation vector $\vec{q}_T = [q_4 q_5 q_6]^\intercal$ that would register a data set points $P = \{\vec{p}_i\}$ to a set of model point $X = \{\vec{x}_i\}$ in a n-dimensional problem, for $n \leq 3$, the quaternion based algorithm described at [27] is suggested. For $n > 3$, the singular value decomposition (SVD) is suggested. Since this is the method that is implemented in the system architecture of this work, a summary of it is found in section 2.2.1. Before stating the ICP algorithm, some used notations are explained:

- $\mathcal{C}$ denotes the closest points operator between two data sets
- Y denotes the resulting set of closest points operator $\mathcal{C}$

$$Y = \mathcal{C}(P, X) \tag{2.5}$$

- $\mathcal{Q}$ denotes the least squares registration between P and Y, resulting in the registration vector $\vec{q}$ and in the mean square point matching error d_{ms}:

$$(\vec{q}, d) = \mathcal{Q}(P, Y) \tag{2.6}$$

- $\vec{q}(P)$ denotes the point set P transformed by the registration vector $\vec{q}$

- τ denotes the change in the mean square error threshold

Algorithm 1 Iterative Closest Point Algorithm

1: **procedure** ICP($(\vec{q}, d)$)
2: $P_0 \leftarrow$ source data set with N_p points $\vec{p}_i$
3: $X \leftarrow$ model (target) data set with N_x supporting geometric primitives
4: $\vec{q} \leftarrow [1, 0, 0, 0, 0, 0, 0]^\intercal$
5: $k \leftarrow 0$
6: $P_k \leftarrow P_0$
7: Compute closest points: $Y_k = \mathcal{C}(P_k, X)$
8: Compute registration: $(\vec{q}, d_k) = \mathcal{Q}(P_0, Y_k)$
9: **if** $d_k - d_{k-1} \leq \tau$ **then**
10: Go to 1
11: **end if**
12: Apply registration: $P_{k+1} = \vec{q}(P_0)$
13: $k = k + 1$
14: Go to 7
15: **end procedure**

2.2.1 Optimal transformation using the Singular Value Decomposition method

The SVD is a factorization of a real or complex $\mathbf{m} \times \mathbf{n}$ matrix $\mathbf{M}$ in the form of $\mathbf{U\Sigma V}^*$, where $\mathbf{U}$ is an $\mathbf{m} \times \mathbf{m}$ real or complex unitary matrix, $\mathbf{\Sigma}$ is a $\mathbf{m} \times \mathbf{n}$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and $\mathbf{V}^*$ is an $\mathbf{n} \times \mathbf{n}$ real or complex unitary matrix. The diagonal entries $\mathbf{\Sigma}$ are denoted as the singular values of $\mathbf{M}$. For ICP applications, if $\mathbf{M}$ is real and $\mathbf{m} \times \mathbf{m}$, then $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}^*$ are real and also $\mathbf{m} \times \mathbf{m}$. $\mathbf{U}$ and $\mathbf{V}^*$ can be viewed as rotation matrices and $\mathbf{\Sigma}$ as a scaling matrix. So when we have a data set, and apply the matrix $\mathbf{M}$, a rotation, a scaling and another rotation are applied to data set. This can be implemented in the ICP algorithm since we are trying to find the best transformation between two data sets. Since only the rotation is calculated through SVD, firstly we need to align the two data sets to the same center. In order to do so, we find the centroid of each data set. The centroids of P and X are respectively:

$$\vec{\mu}_p = \frac{1}{N_p} \sum_{i=1}^{N_p} \vec{p}_i \tag{2.7}$$

$$\vec{\mu}_x = \frac{1}{N_p} \sum_{i=1}^{N_x} \vec{x}_i \tag{2.8}$$

Subtracting the corresponding center of mass to every point in the two point data sets, results into two new data sets centered in the same point, $X' = \{\vec{x} - \vec{\mu}_x\} = \{\vec{x}'_i\}$ and $P' = \{\vec{p}_i - \vec{\mu}_p\} = \{\vec{p}'_i\}$. Now the data is normalized and centered. Considering the cross-covariance matrix $\mathbf{C}$ between the two data sets:

$$\mathbf{C} = \sum_{i=1}^{N_p} p'_i {x'_i}^T \tag{2.9}$$

We want to find the rotation matrix $\mathbf{R}$ that maximizes the trace $\mathrm{Tr}\left[\mathbf{R} \cdot \mathbf{C}\right]$. The optimal rotation matrix $\mathbf{R}$ is:

$$\mathbf{R} = \mathbf{V}\mathbf{U}^{\intercal} \tag{2.10}$$

Where $\mathbf{V}$ and $\mathbf{U}$ are the SVD rotation matrices of $\mathbf{C}$. Since before finding the rotation matrix $\mathbf{R}$, the data sets P and X were centered, the translation matrix $\mathbf{T}$ is:

$$\mathbf{T} = \vec{\mu}_x - \mathbf{R}\vec{\mu}_p \tag{2.11}$$

2.3 Predictive Entropy Search for Multi Objective Optimization

The purpose of a MOO is to find the right balance among several objectives. The right balance is defined by the Pareto points, points that provide an optimal design in terms of more than one objective. Another common aim of many MOO is to achieve the first purpose with as few evaluations of the design space as possible.

Formally, an MOO algorithm seeks to simultaneously optimize n objective functions $f_1, \ldots, f_n : \mathcal{X} \to \mathbb{R}$ over a finite design space $\mathcal{X}$. The finite design space is a set of all possible inputs $\mathbf{x}$.

We say that a point $\mathbf{x}$ dominates other point $\mathbf{x}'$ if $f_i(\mathbf{x}) \leq f_i(\mathbf{x}') \forall i \in 1, \ldots, n$ and $\exists i \in 1, \ldots, n$ for which $f_i(\mathbf{x}) < f_i(\mathbf{x}')$.

For $\mathbf{x} \in \mathcal{X}$, $\mathbf{f}(\mathbf{x})$ is all the evaluations of the input $\mathbf{x}$ that belongs to the design space, i.e., $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$.

The objective space is then the image $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^n$. In words, is a subset of $\mathbb{R}^n$, consisting in all the evaluations of all the design space.

The Pareto set $\mathcal{X}^*$ is a subset of the design space $\mathcal{X}$, containing all non-dominated points of $\mathcal{X}$, i.e., the set such that $\forall \mathbf{x}^* \in \mathcal{X}^*, \forall \mathbf{x} \in \mathcal{X}, \exists i \in 1, \ldots n$ for which $f_i(\mathbf{x}^*) < f_i(\mathbf{x})$, i.e., $\mathbf{x}^*$ dominates $\mathbf{x}$.

The Pareto frontier is then the image $\mathbf{f}(\mathcal{X}^*) \subset \mathbf{f}(\chi)$, in words, is a subset of the objective space, consisting in all the evaluations of all the Pareto set.

PESMO is a MOO based on *predictive entropy search*. The idea is to, at each iteration of the algorithm, choose the new evaluation point $\mathbf{x}$ of the design space $\mathcal{X}$, that maximizes the information gained about the Pareto set $\mathcal{X}^*$. Each of the n objective functions f_i for $i \in 1, \ldots, n$ is an unknown black box function so they are probabilistically modeled as a Gaussian process (GP) prior, with i.i.d. zero mean Gaussian observation noise.

A MMO algorithm that has a model based approach needs an acquisition function, i.e., a function that

serves as the criteria for choose the next evaluation point of $\mathcal{X}$. Before defining the acquisition function of PESMO, let $D = \{(\mathbf{x}_n, \mathbf{f}(\mathbf{x}_n))\}_{n=1}^{N}$ be a set, with the evaluation point $\mathbf{x}_n$ from $\mathcal{X}$ chosen at the step n and its objective functions evaluation $\mathbf{f}(\mathbf{x}_n)$, at every iteration until iteration N. For PESMO, the acquisition function, known as *entropy search*, is the expected reduction in entropy $H(\cdot)$ of the posterior distribution over the Pareto set $\mathcal{X}$, i.e., $p(\mathcal{X}^*|D)$:

$$\alpha(\mathbf{x}) = H(\mathcal{X}^*|D) - E_{\mathbf{y}}\left[H(\mathcal{X}^*|D \cup \{(\mathbf{x}, \mathbf{y})\})\right] \tag{2.12}$$

where $\mathbf{y}$ is the value of the a priori independent GP models of the objective functions at $\mathbf{x}$. The expectation $E_{\mathbf{y}}[\cdot]$ is taken with respect to the posterior distribution for $\mathbf{y}$ given the GP models, $p(\mathbf{y}|D, \mathbf{x}) = \prod_{i=1}^{n} p(y_i|D, \mathbf{x})$. The next acquisition point $\mathbf{x}_{N+1}$ is the one that maximizes the acquisition function, i.e., $\mathbf{x}_{N+1} = \mathrm{argmax}_{x \in \mathcal{X}}\, \alpha(\mathbf{x})$.

The acquisition function is however, unfeasible because it is difficult to evaluate the entropy of the Pareto set $\mathcal{X}^*$ given D, i.e., $H(\mathcal{X}^*|D)$ in equation 2.12. So it is approximated to the *predictive entropy search*:

$$\alpha(\mathbf{x}) = H(\mathbf{y}|D, \mathbf{x}) - E_{\mathcal{X}^*}\left[H(\mathbf{y}|D, \mathbf{x}, \mathcal{X}^*)\right] \tag{2.13}$$

This new acquisition function is formulated from equation 2.12 due to the principle of mutual information. For this case, the information known about $\mathbf{y}$ is used to gain information about $\mathcal{X}^*$, so in equation 2.12, the roles of $\mathcal{X}^*$ and $\mathbf{y}$ are exchanged, leading to the equation 2.13. Now, the first term of equation 2.13 is the entropy of the predictive distribution $p(\mathbf{y}|D, \mathbf{x})$ and the expectation $E_{\mathcal{X}^*}(\cdot)$ in the second term is taken with respect to the posterior distribution for the Pareto set $\mathcal{X}^*$ given the observed data D. $H(\mathbf{y}|D, \mathbf{x}, \mathcal{X}^*)$ is the entropy of $P(\mathbf{y}|D, \mathbf{x}, \mathcal{X}^*)$ which is the predictive distribution of the objective function at $\mathbf{x}$ given the collected data D conditioned to the Pareto set $\mathcal{X}^*$. The deduction of how to evaluate both terms of equation 2.13 can be found in detail in [2]

2.4 Occupancy Grid

A classical approach for representing the physical world in 2D, is using an occupancy grid map. The occupancy grid map represents the environment as an evenly spaced field, each x-y coordinates of which, is a binary random variable whose value determines if the location is occupied with an obstacle or not.

In order to represent occupancy grids, the ROS package *nav_msgs*[1] is used. The *nav_msgs* package contains several ROS messages, services and actions used to interact with the navigation stack. An example of on occupancy grid is shown in figure 2.1

[1] http://wiki.ros.org/nav_msgs

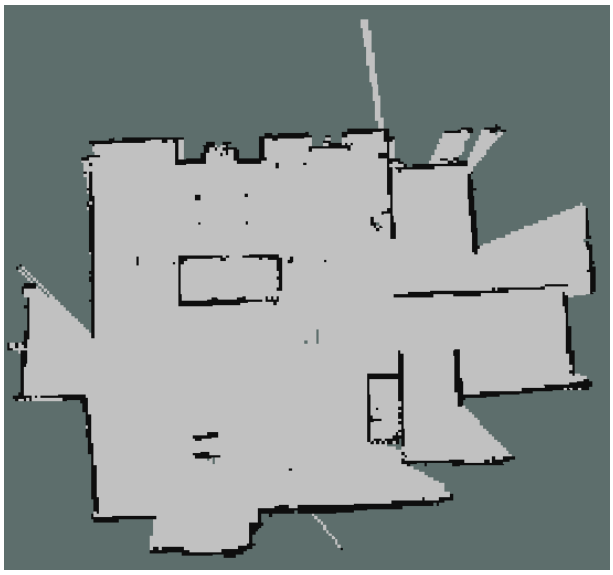


Figure 2.1: Example of an occupancy grid

2.5 Laser Scan

Distance sensors that allow your robot to detect and avoid obstacles tend to be both very useful in localization methods. To represent these laser scans, the ROS package *sensor_msgs*[2] is used. This package contains message definition for commonly used sensors, including laser range finders. An example of a laser scan representation is shown in figure 2.2

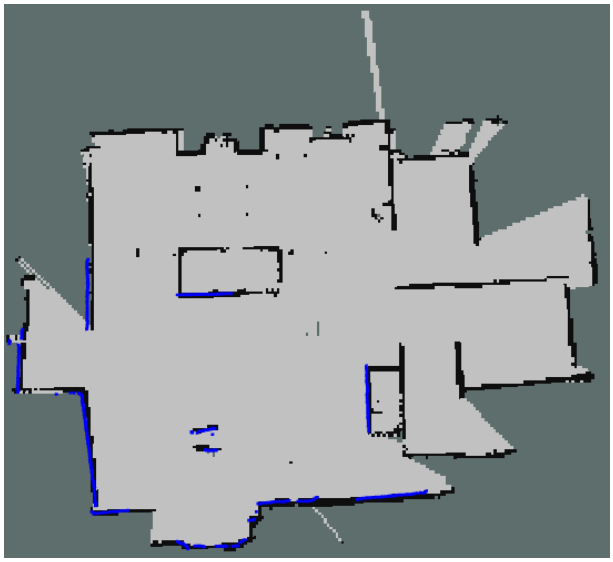


Figure 2.2: Example of a laser scan representation

[2]http://wiki.ros.org/sensor_msgs

2.6 Point Cloud

An usual option to represent the three-dimensional external surface of an object, is using a point cloud. A point cloud is a data structure in some coordinate system usually collected by stereo cameras or 3D scanners. Point cloud is the data structure used in this work for employing the ICP algorithm. An already developed C++ framework is used, which is the Point Cloud Library (PCL)[3]. Since we will be using 2D point clouds, its representation is similar to the laser scan displayed in the previous section 2.5.

[3] http://pointclouds.org/

Chapter 3

Multi-objective Optimization System Architecture

A valuable aspect of the proposed system architecture is that all modules were designed to be generic and completely independent from each other, so the user would only have to construct the link between the proposed architecture and an optimizer of his choice. The constructed modules were implemented as ROS packages and can easily be used or adapted for other study scenarios. The developed system contains all the packages described in the following sections of this chapter. In section 3.1 a global picture of the system architecture is shown and explained. In section 3.2, it is described how the map represented as an occupancy grid was converted into a point cloud. Section 3.3 explains the conversion of laser scans into point clouds. Section 3.4, which is the main module, describes how the occupancy grid map and the laser scan, both already converted into point clouds, are used to score the position of the robot. Section 3.5 is the link between the multi objective optimizer and the ROS infrastructure. Finally, in section 3.6, it is referenced all the other minor tools developed in the system, as the score accumulator and the simulation data recorder.

3.1 Description

The developed system can be divided in three main components: the multi objective optimizer MOO, the ROS instance server RIS and the client-server PESMO-ROS wrapper W that links both. The complete cycle of the system is described in algorithm 2.

Algorithm 2 Multi Objective Parameter Optimization for ROS Algorithms pseudo code

procedure MULTI OBJECTIVE OPTIMIZATION FOR ROS ALGORITHMS
  $n \leftarrow$ number of objective function evaluations
  for $i \leftarrow 1, n$ **do**
    MOO suggests a promising parameter set
    W loads suggestion into ROS parameter server
    W requests RIS an experiment run to evaluate the parameter set
    RIS starts evaluating the objective functions
    RIS launches algorithm
    RIS plays experiment and waits until finished
    RIS shutdowns algorithm
    RIS stops evaluation and responds to W with the results
    W forwards results to MOO
    MOO analysis results
  end for
  return Evaluated parameter sets with correspondent scores
end procedure

This architecture was inspired in a more generic automatic parameter optimization pseudo code, that can be found in [3]. The chosen third party MOO software was PESMO which is developed in Python. PESMO is fed with a configuration file where is discriminated the scoring functions, also denoted as objective functions, the parameters of the algorithm we want to optimize, the parameters domain and the maximum number of the objective functions evaluations. The algorithm we want to optimize is the AMCL algorithm, implemented in ROS. The objective functions are the CPU time consumption of the AMCL algorithm (measured in the number of ticks the CPU was used for processing the AMCL algorithm process) and the mean euclidean distance error between robot's pose given by AMCL and the ground truth pose given by the mo-cap system. PESMO starts by suggesting a set of AMCL parameters to evaluate. These parameters are then loaded into the ROS parameter server by the PESMO wrapper. The wrapper also triggers the instance server by requesting an experiment run with the suggested parameters, and waits for a response containing the objective functions results which are then forwarded to PESMO. The experiment run consists in playing the recorded robot sensor data and evaluate its AMCL position error as well as the CPU time consumption by AMCL. PESMO analysis the results, beginning a new iteration with a new promising parameter set suggestion. The optimization stops when the maximum number of experiment evaluation is reached. The output of the system will be all evaluated parameter sets with their correspondent scores. From this, one can easily plot the achieved Pareto frontier and choose a parameter set that best fits our needs. In figure 3.1 is shown a diagram with a global picture of the developed system architecture. As mentioned before, the evaluation functions are the CPU time consumption of the AMCL algorithm and the euclidean distance between the robot's AMCL pose and the ground truth pose. Since the ground truth pose is acquired by a mo-cap system, a new approach has

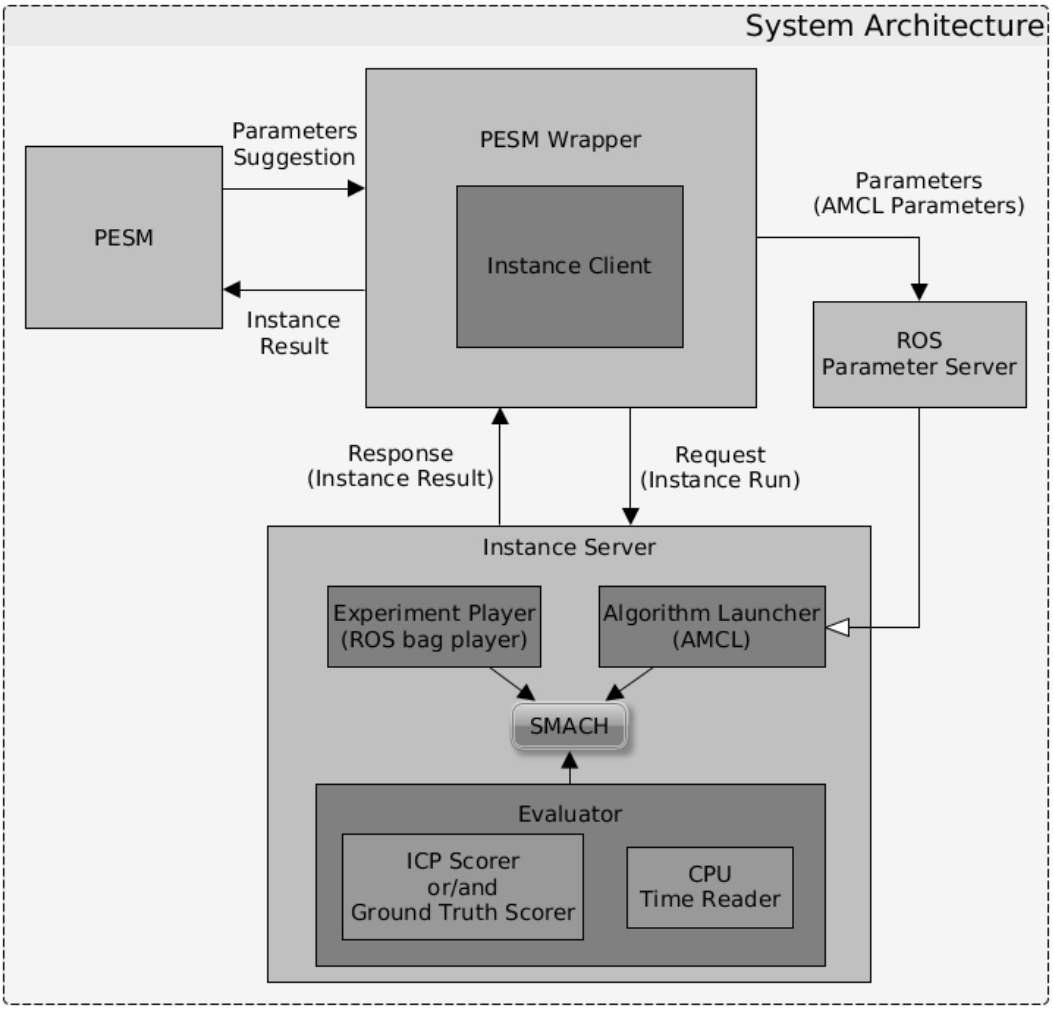


Figure 3.1: Diagram of the developed system architecture

been developed on how to estimate this error using only the recorded data. This approach can be used as the objective function renouncing the use of the mo-cap system. This approach is fully described in section 3.4 and a comparison with the ground truth data is presented in chapter 5.

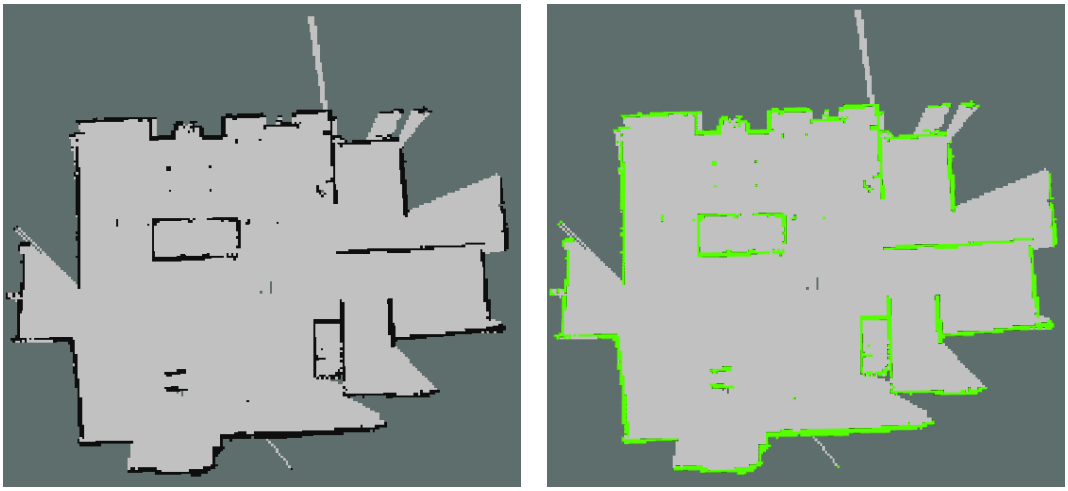
3.2 Occupancy Grid to Point Cloud Conversion

The primary content of this package is a method to convert an occupancy grid (a map in this case) into a point cloud. As seen in section 2.4, the occupancy grid map is represented as a 2D data structure. Logically, the resultant point cloud must also represent a 2D data structure. To do so, all its points will have as the z coordinate value, the z coordinate of the occupancy grid frame. To transform each cell into a map coordinate, firstly we need to know the grid height and width. Since the grid data is organized in row-major order, its width and height will tell us how to organize the point cloud in terms of rows and columns. The grid resolution will tells us how much a cell measures in the real world coordinates and the grid origin position will tell us where in the real world coordinates, the map is centered. This method can be described by the following algorithm:

Algorithm 3 Occupancy Grid to Point Cloud Conversion Algorithm

procedure CELLSTOCOORDINATES
  $w \leftarrow$ grid width ▷ In *cells*
  $h \leftarrow$ grid height ▷ In *cells*
  $x \leftarrow$ grid origin *x* coordinate ▷ In *m*
  $y \leftarrow$ grid origin *y* coordinate ▷ In *m*
  $r \leftarrow$ grid resolution ▷ In *m/cell*
  $c \leftarrow$ occupancy value threshold
  $d \leftarrow$ grid data array ▷ Contains the occupancy probability of each cell
  for $j \leftarrow 1, h$ **do** ▷ Iterate through the number of rows
    for $i \leftarrow 1, w$ **do** ▷ Iterate through the number of columns
      if $d[w * j + i] \geq c$ **then**
        $p_x \leftarrow x + i * r + r/2$ ▷ World map x coordinate
        $p_y \leftarrow y + j * r + r/2$ ▷ World map y coordinate
        $p_z \leftarrow 0$
        $PCL \leftarrow (p_x, p_y, p_z)$ ▷ Add p_x, p_y and p_z to the point cloud PCL
      end if
    end for
  end for
  return PCL
end procedure

The result of the *Occupancy Grid to Point Cloud Conversion* package is presented in figure 3.2. Sub figure 3.2a shows the initial occupancy grid map and sub figure 3.2b illustrates the conversion.



(a) Environment map in an occupancy grid format (b) Converted occupancy grid map into point cloud

Figure 3.2: Conversion of occupancy grid to point cloud

Only the completely black cells of the occupancy grid map, that represent a surely occupied cell, were converted into point cloud points, represented in a green color in sub figure 3.2b. It is possible to

observe that the green points overlap all the black cells, confirming a correct conversion.

3.3 Laser Scan to Point Cloud Conversion

In this module, one or multiple laser scans are converted into a point cloud. In order to do so, the ROS package *laser_geometry*[1], a package already developed, documented and publicly available in ROS repository is used. The *laser_geometry* package makes use of the *tf*[2] package that lets the user keep track of multiple robot coordinate frames over time. With this, each individual laser ray is appropriately converted, which is fundamental for a precise localization specially in tilting laser scanners and moving robots. The main difference between *laser_geometry* and the current work implementation is the extension for receiving and publishing trigger events, essential for controlling when this modules start and stops doing its purpose.

The result of the *Laser Scan to Point Cloud* conversion package are presented in figure 3.3 and figure 3.4. Both figures exhibit in a light blue color, the laser scan in a *Laser Scan* data format. In a darker blue color, the conversion into *Point Cloud* data format is displayed.

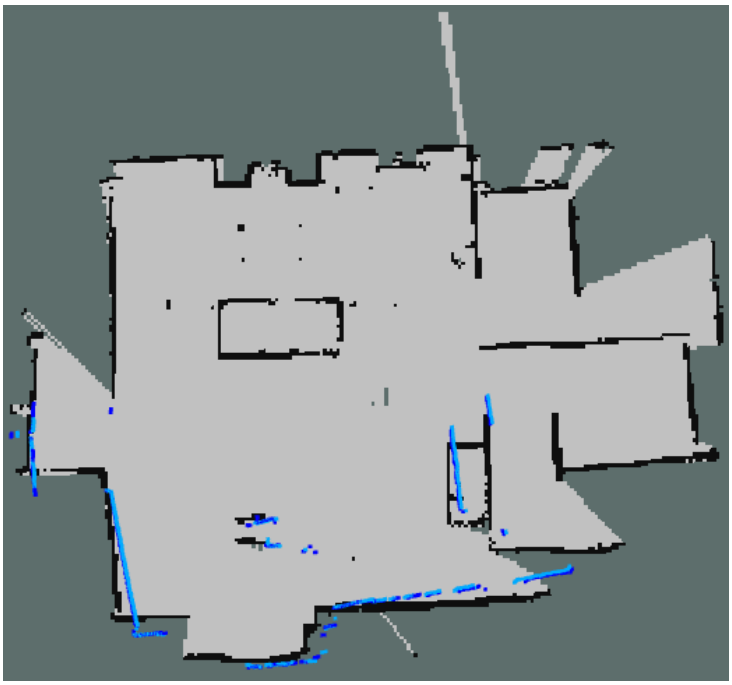


Figure 3.3: Conversion of laser scan to point cloud. Example of when the robot starts to get lost

[1] http://wiki.ros.org/laser_geometry
[2] http://wiki.ros.org/tf

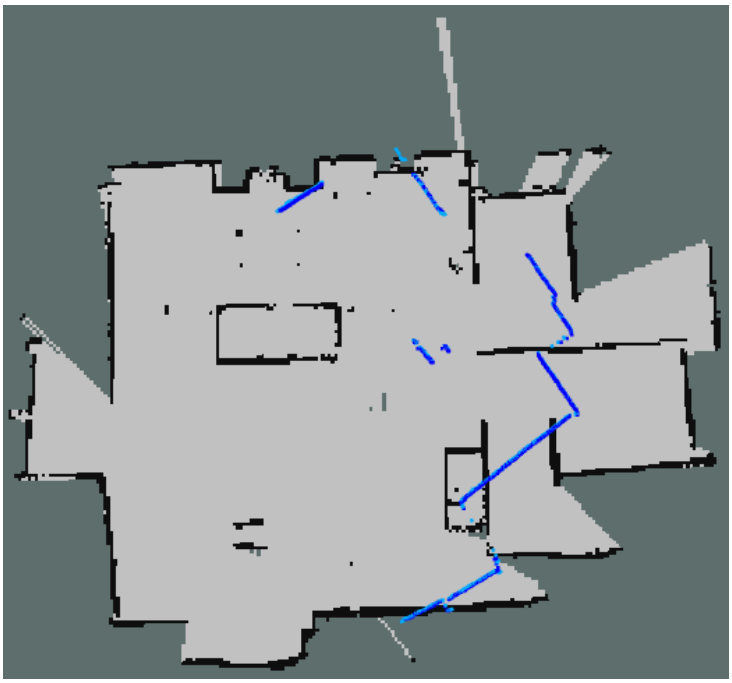


Figure 3.4: Conversion of laser scan to point cloud. Example of when the robot is completely lost

Again, we can observe that the points in a dark blue color overlap the lighter ones in terms of x-y coordinates, confirming a correct conversion in the x-y plane. In figure 3.5 we can also check the correct conversion in the z coordinate.

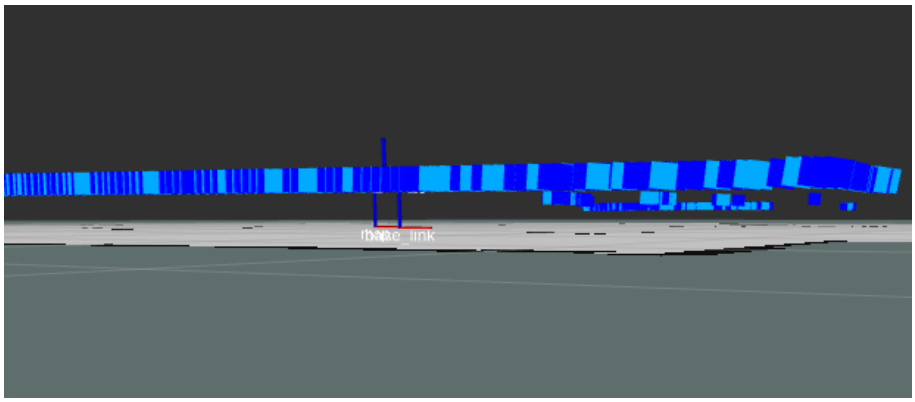


Figure 3.5: Z coordinate check of the conversion of laser scan to point cloud

Since the laser scans represent obstacles, which are in their majority walls, figure 3.3 displays an example when the laser scan is already distant from the black occupancy grid cells, that also represent the environment walls. This suggests that the robot does not have a accurate estimation of its pose. Figure 3.4 illustrates an instance when the robot is completely lost in the environment. The goal is to find the best transformation that fits the laser scans to the occupancy grid.

3.4 Point Cloud ICP Scorer

In this module, two points clouds are received and a spatial transformation that best align them is found. In the alignment process, using the ICP algorithm described in section 2.2, a fitness score is given. If this score is lower than the user defined threshold, an estimation of the pose error is computed, so this process can be used as objective function in the chosen multi objective optimizing algorithm PESMO, described in section 2.3. The ICP algorithm is implement using he Point Cloud Library, a standalone, large scale, open project for 2D/3D image and point cloud processing [3]. The output of the section 3.2 is the target (or fixed) point cloud, leaving the output of the section 3.3 as the source point cloud. The output of this package is therefore, the alignment fitness score, the estimated pose error both as an euclidean distance error and an angular error, the registered point cloud and the transformation that aligned both clouds (and transforms the source cloud into the registered cloud). In figure 3.6 is shown a diagram representing the proposed evaluation method.

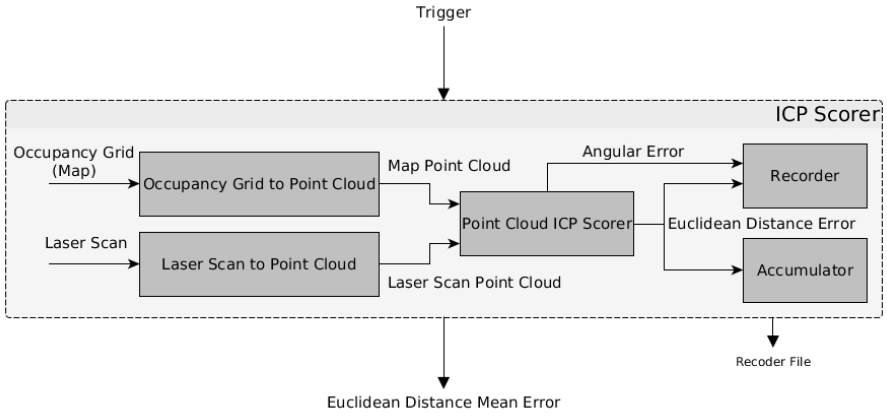


Figure 3.6: Diagram of the proposed ICP scorer method

Remembering section 2.2, the respective rotation matrix of the alignment transformation, is applied to the centroid of the source cloud. However, since the source cloud is angular aligned with the estimated robot orientation, and since the well localized robot should always have lasers readings align with the map, the computed rotation should be a good approximation of the true angular error. The estimated angular error is then equal to the yaw value of the rotation. The same reason applies for computing an estimated euclidean distance error. We can also simply compute it using the respective translation vector $T = [x, y, z]$ of the alignment transformation. The estimated euclidean distance error e is then:

$$e = \sqrt{x^2 + y^2} \tag{3.1}$$

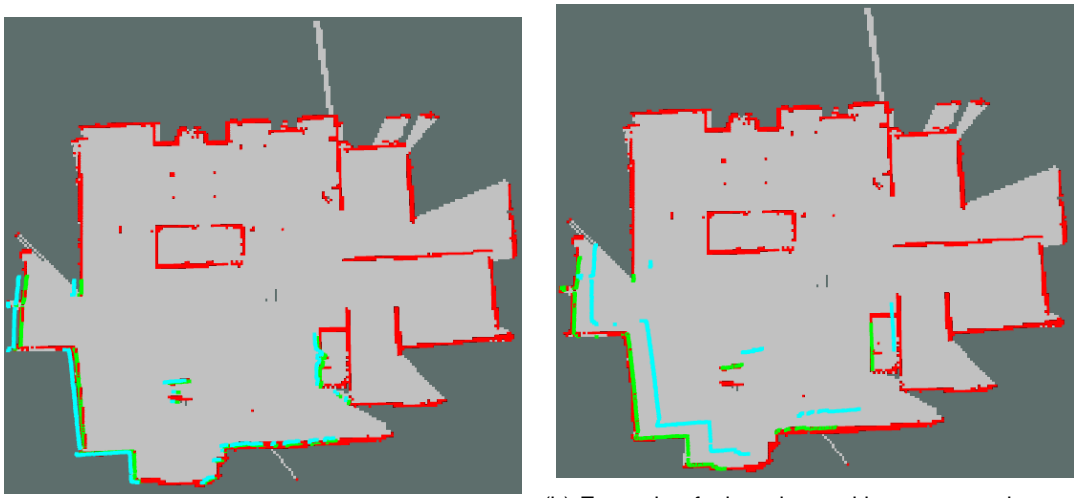
This method can also be described by the pseudo code of the following algorithm 4.

[3] http://pointclouds.org/

Algorithm 4 Point Cloud ICP Scorer Algorithm

procedure SCORER
  $t \leftarrow$ ICP fitness score threshold
  $pcl\_t \leftarrow$ target point cloud
  $pcl\_s \leftarrow$ source point cloud
  $icp\_fitness\_score \leftarrow ALIGN(pcl\_t, pcl\_s)$
  if $icp\_fitness\_score \leq t$ **then**
    $tranformation \leftarrow GET\_TRANFORMATION(pcl\_t, pcl\_s)$
    $e \leftarrow COMPUTE\_ESTIMATED\_ERROR(transformation)$
    return e
  end if
end procedure

The *Point Cloud ICP Scorer* package results are shown in sub figures 3.7a and 3.7b. In red, is the target point cloud (occupancy grid), and in light blue is the source point cloud (laser scan). In green, is displayed the result of this module denoted as registered source point cloud.



(a) Example of when the robot is well localized

(b) Example of when the position accuracy is starting to deteriorate

Figure 3.7: Alignment of the target and source cloud through the ICP algorithm

It is possible to observe that the ICP algorithm is able to align the source cloud to the target cloud, since the registered cloud green points are overlapping the target cloud red points. In sub figure 3.7a, the robot belief of its position is well estimated since the source cloud is almost aligned with the target cloud before running ICP. The opposite situation occurs in sub figure 3.7b, where the robot belief of its position is no longer reliable. In both cases, since the alignment is visually good, it is expected a low ICP fitness score. The user can define the ICP fitness score threshold, which for cases above, the package will not consider the registration. It is also expected that the estimate of the euclidean distance between the robot estimated position and its true ground truth position, will be smaller in the first case. This extra verification is done in chapter 5.

3.5 PESMO-ROS Wrapper

This module is the bridge between the optimization algorithm and the ROS infrastructure. Its main function is to receive a request by the optimization algorithm with all the parameters values proposed to evaluation. This parameter set is redirected to the ROS infrastructure where they are uploaded to parameter serve. The wrapper also triggers the the instance server so it can start the experiment run. When the experiment finishes, the objective functions results are retrieved and redirected by the wrapper to PESMO. To encourage this module re-usability The wrapper was designed in such a way that the user would only need to specify in which ROS topics will be published the evaluation results. All the results data formats required by PESMO are internally handled by this module. Finally, there is also the option of multiple evaluations of the same parameter set. This is useful for non deterministic evaluation functions, where not a single evaluation but the average of multiple are a more reliable evaluation metric. This is important to handle stochastic nature algorithms, since PESMO does not support this option. There was a choice of modifying PESMO, however, since it is a third party software and since the developed wrapper is easily adapted to other optimization algorithms, this was the design choice that was made.

3.6 Other Modules

To conclude the system architecture, some more minor modules were developed. The main ones are the *Score Accumulator* and the *Recorder*. The *Score Accumulator* is an useful tool when the objective function is an average value of some computed data. In our case, one of the objective functions is the average position error of the robot during the experiment run. The *Recorder* module simply writes into an user defined file, the values of all computed pose errors during the experiment run. There is already a ROS tool for this function, but the designed *Recorder* was a simple tool useful for recording the data in such way that facilitated the data visualization and plotting that is presented in chapter 5.

Chapter 4

Experiment Setup

In this chapter is described the main components of the experiment and how was it realized. In section 4.1 and section 4.2, it is described the location and the robot used in the experiment. In section 4.3, a detailed description of the experiment and collected data is done. The configurable AMCL parameters and other parameters related to the ICP score method and the multi objective optimizer PESMO are described in sections 4.4 and 4.5 respectively.

4.1 Testbed

The testbed is an apartment-like environment based on the general design and specifications of the RoCKIn@Home testbed. The reader can consult the RoCKIn@Home rule book[1] for more details. The testbed is equipped with, among many other sensors and actuators, a Motion capture (Mo-cap) system. Mo-cap is the process of recording a live motion of objects and translating it into digital performance usable data for, in this case, robotic algorithm validation. The mo-cap provides the ground truth of the robot so the precision of the developed algorithm can be evaluated. The system used is composed by 12 OptiTrack(R) Prime 13 cameras with the following image sensor tech specs:

- Resolution: 1280 x 1024 (1.3MP)
- Pixel size: 4.8 µs x 4.8 µs
- Frame Rate: 30-240 FPS (adjustable)
- Latency: 4.2 ms
- Shutter Type: global
- Shutter Speed:
  - Default: 500 ps (0.5 ms)
  - Minimum: 10 ps ps (0.01 ms)

[1] http://rockinrobotchallenge.eu/rockin_d2.1.6.pdf

– Maximum: 3900 ps (3.9 ms) at 240 FPS

With the described setup, a ground truth precision under 3 mm can be achieved.

4.2 Robot

The robot consists in the MOnarCH robot[2] that was originally designed to to interact with children, staff, and visitors, engaging in edutainment activities in the pediatric infirmary at the Portuguese Oncology Institute at Lisbon (IPOL), Portugal. As described in [3], it has the following tech specs:

- Omni-directional mobile base equipped with for mechanic wheels actuated by four independent motors
- Two laser (5 and 30 m) range finders
- Two on-board computer with i7 processor
- 24 kg of weight with a maximum velocity if 2.5 m/s and 1 m/s^2 acceleration

More details about the robot can be found in [28].

4.3 Experiment description

The experiment consists in teleoperating the robot around an apartment-like test bed that was previously mapped. The robot movement consists in both linear and angular movements in order to simulate realistic robot movements and to have a more reliable error evaluation of the daily tasks these robots are being developed for. Firstly, an estimate of the ground truth pose of the robot provided by the mo-cap system is given to the AMCL in order to initialize the filter. Then, while the robot is being teleoperated around the test bed, the real robot pose given by the mo-cap system, the estimated AMCL pose and the laser and odometry readings are recorded into a data set. Since this data set will later be used for offline multi objective optimization of the AMCL parameters, additional data required to correctly offline rerun the AMCL algorithm was also recorded. The recorded data is then used to optimize the AMCL algorithm with the developed system. This recorded data has enough duration for being cropped into multiple parts. The idea is to have multiple data sets with different movements so one can tune AMCL in one training data set, and evaluate the tune robustness in another data set different from the training set.

4.4 AMCL Parameters Description

The AMCL algorithm implemented in ROS has a total of 47 parameters that can be customized. These parameters change the behavior of the AMCL algorithms in relation to the overall filter, the laser model

[2] http://monarch-fp7.eu/

and the odometry model. Some of these 47 parameters are not necessary to tune since they are related to, for instance, data for visualization purposes or the initial position of the robot. We end up with a subset of 22 configurable parameters shown in table 4.1. In this table is also shown the default parameter set, the data type and minimum and maximum value of each parameter.

Table 4.1: AMCL parameters configuration

	default	type	min	max
laser_max_beams	30	integer	5	100
min_particles	100	integer	5	980
max_particles	5000	integer	1000	10000
kld_err	0.01	float	0.005	0.5
kld_z	0.99	float	0.5	0.999
odom_alpha1	0.005	float	0.001	1
odom_alpha2	0.005	float	0.001	1
odom_alpha3	0.010	float	0.001	1
odom_alpha4	0.005	float	0.001	1
odom_alpha5	0.003	float	0.001	1
laser_z_hit	0.95	float	0.1	5
laser_z_short	0.1	float	0.05	1
laser_z_max	0.05	float	0.005	1
laser_z_rand	0.05	float	0.005	1
laser_sigma_hit	0.2	float	0.05	1
laser_lambda_short	0.1	float	0.005	1
laser_likelihood_max_dist	2.0	float	0.1	20
update_min_d	0.2	float	0.05	0.5
update_min_a	0.5236	float	0.1	1
resample_interval	2	integer	1	3
recovery_alpha_slow	0.01	float	0.001	0.01
recovery_alpha_fast	0.1	float	0.05	1

A complete description of all AMCL parameters can be found in the AMCL ROS wiki web page[3].

4.5 Additional Parameters Description

As mentioned previously in section 3.4, the ICP algorithm is used to align two point clouds. The ICP algorithm also has parameters that largely affect the alignment process, and its optimization could actually be treated as an additional single or even MOO problem. In table 4.2 are show the chosen ICP

[3] http://wiki.ros.org/amcl

parameters, based on a repeated visual analyses of the alignment of two point clouds, done before the MOO.

Table 4.2: ICP parameters configuration

	type	value
max_correspondance_distance	float	2.5
max_iterations	integer	200
transformation_epsilon	float	0.00000001
euclidean_fitness_epsilon	float	1

It is also important to refer, that only the estimates of the pose error with an associated ICP fitness score below 0.001 were considered reliable. This design choice is not optimal, since for larger errors in position, the ICP is not capable of a good alignment resulting in a bad estimate of the pose error. It was decided that for this cases, to not include this estimate in the calculation of the mean pose error. The remaining parameters were chosen so the ICP could be fast, abdicating of a successful alignment for very different target and source clouds. Finally, in table 4.3 it is shown the chosen PESMO parameters.

Table 4.3: PESMO parameters configuration

	type	value
grid_size	integer	1000
max_finished_jobs	integer	100
likelihood	string	GAUSSIAN

The *max_finished_jobs* parameter refers to the limit of objective function evaluations. Since PESMO was tested in [2] with around 200 evaluations, and since each evaluation in our case takes 30 seconds to conclude, the limit of 100 evaluation was chosen so the MOO would take around 6 hours to complete.

Chapter 5

Experimental Results

In this chapter the developed *Point Cloud ICP Scorer* package is validated and the results of the MOO are show and check for consistency. In section 5.1 is described the experiments for validating the developed package. In section 5.2 is shown the result of the MOO with the ground truth euclidean distance position error as objective function. In section 5.3 is shown the result of the MOO but now with the estimated euclidean distance position error as objective function. Finally, in section 5.4 is shown the results of another MOO, now with an attempt of overcoming the encountered difficulties in achieving a reliable Pareto frontier.

5.1 Point Cloud ICP Scorer Validation

Firstly, it is important to validate the created module that will be used as objective function. As mention in section 3.4, this module estimates the mean pose error of an experiment run. The mean pose error is defined by the euclidean distance between the AMCL and the ground truth position and by the difference between the orientation angle from AMCL and from ground truth. From this point on, for simplicity, replaying the recorded data and evaluating the objective functions will be denoted as a running a job. In order to validate this approach, three different experiments are realized. The first experiment consists in running a job with random AMCL parameters, and comparing the pose error estimated given by the *Point Cloud ICP Scorer* to the ground truth data. The results of this first experiment are presented in figures 5.1 to 5.6.

In figure 5.1 is represented with a blue line the estimated error, and with a green line the true error. It is possible to verify, that for this parameter configuration, the estimated error tends to follow the same behavior of the true error. In this case, for relatively small errors there is an over estimation and for bigger error an under estimation. The mean of the estimated error for this job was 0.255 m (25.5 cm) while the mean of the true error was of 0.245 m (24.5 cm), resulting in a difference of means of -0.010 m (-1 cm).

Figure 5.2 displays the difference between the estimated and the true position error during the job. This differences resulted in a root mean square error (RMSE) of 0.043 m (4.3 cm). We observe a

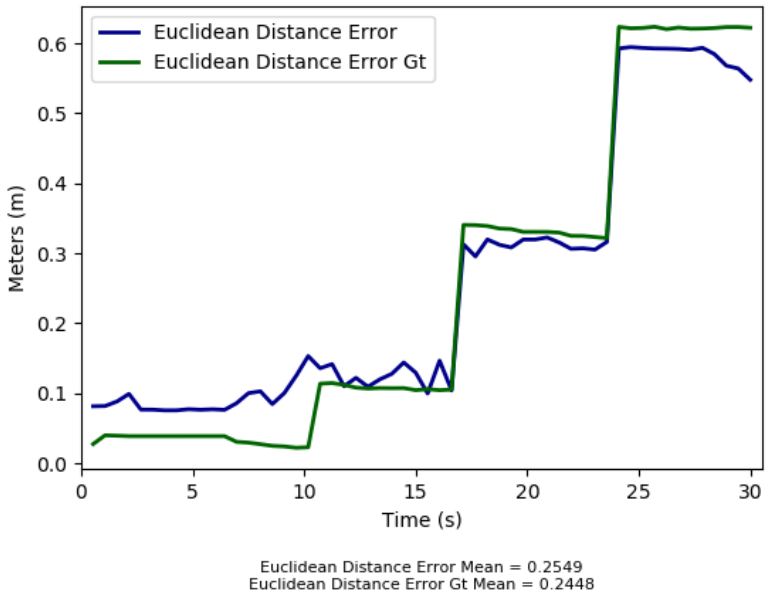


Figure 5.1: True and estimated euclidean distance error of a random job

maximum position error in absolute value of 0.13 m (13 cm) around second 10. We also verify that the difference between the two position errors tends to be higher as the jobs runs. As we seen in the previous figure 5.1, the robot is completely lost after 15 seconds being the true position error very high. We then can associate these higher ground truth errors with a greater difference between the estimated error and the true position error simply because the laser scans are no longer near the map walls, being more difficult for the ICP algorithm to align them and estimate the error. After a certain value of ground truth error, it is expected that the ICP algorithm no longer can align the laser scan to the occupancy grid, neither can estimate a position error. This is a known limitation of the proposed approach, and a suggestion for its attenuation might consist in optimizing the ICP algorithm.

In figure 5.3 we see the box plot of this experiment for the difference between the true and estimated euclidean distance error. Most of the error is contained between -0.04 m (-4 cm) and 0,025 (2,5 cm) resulting of a range between the second and fourth quartile of 6.5 cm. The medium in approximately 0 but there is a slightly bigger spread for negative difference values. In this situation, the overestimated error was larger than the case of when it was underestimated.

Now, the same analysis is done for the angular error. In figure 5.4 is shown with a blue line the estimated angular error and with a green line the true angular error. We can see that for this job and parameter configuration, both errors behave similarly. However, at around second 12, errors change in an opposite way. The mean of the estimated angular error was of -0.0628 rad (-3.60°) and the mean of the true angular error was of -0.054 rad (-3.09°) resulting in a difference of means of -0.0088 rad (-0.51°).

Figure 5.5 displays the difference between the estimated and the true angular error during the job. This differences resulted in a RMSE of 0.019 rad (1.08°). We observe a maximum angular error in

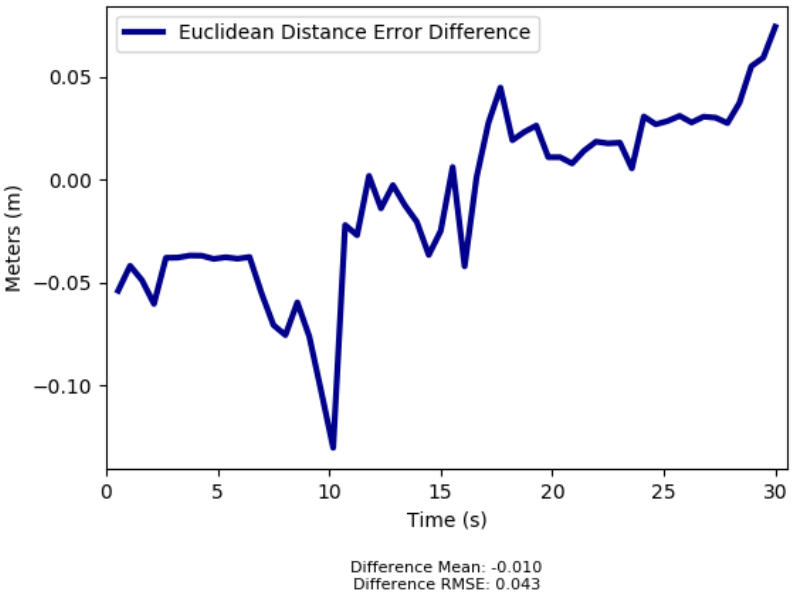


Figure 5.2: Difference between the true and estimated euclidean distance error of a random job

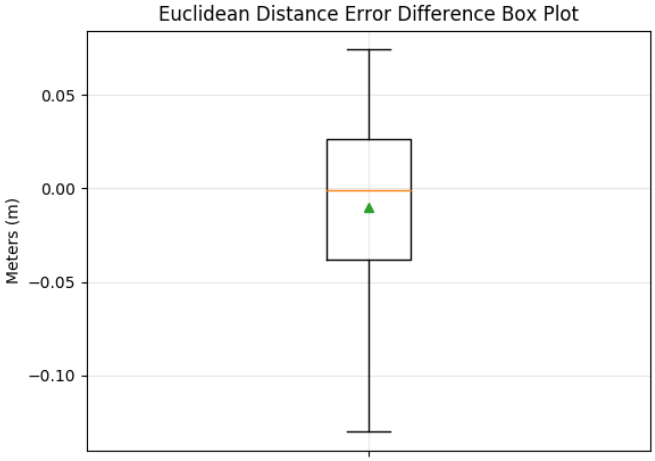


Figure 5.3: Box plot of the ifference between the true and estimated euclidean distance error of a random job

absolute value of 0.065 rad (3.72°) around second 12.

Finally, and concluding this experiment results, in figure 5.6 is presented the box plot of this experiment for the difference between the true and estimated angular error. Excluding the outlier points, data ranges between -0.015 rad (-0.86°) and 0.035 rad (2°) resulting in a total range of 0.04 rad (1.14°). The median is approximately 0.008 rad (0.46°),and the second quartile is approximately 0. This means that most of the difference is positive, suggesting a slightly underestimation of the angular error.

The previous results are from an isolated case and can not guarantee the good function of the developed module for all possible parameter sets. In order to better evaluate the performance of this

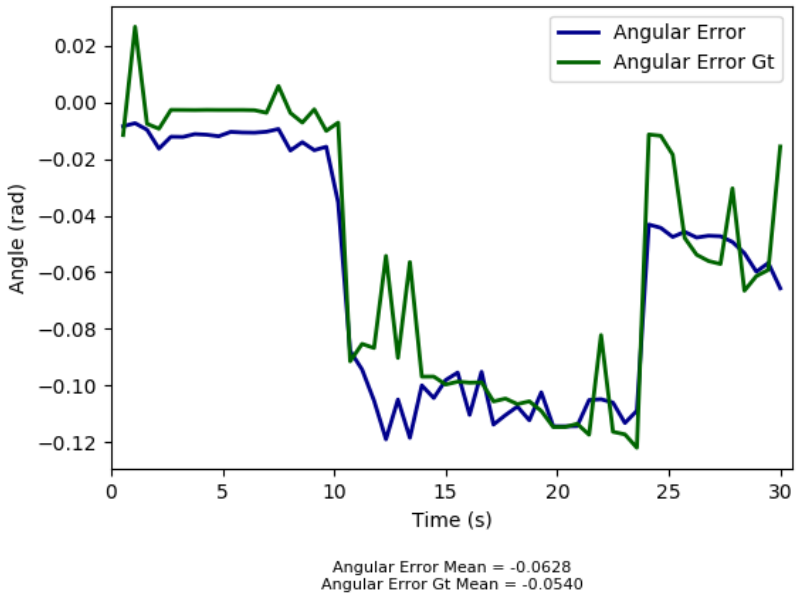


Figure 5.4: True and estimated angular error of a random job

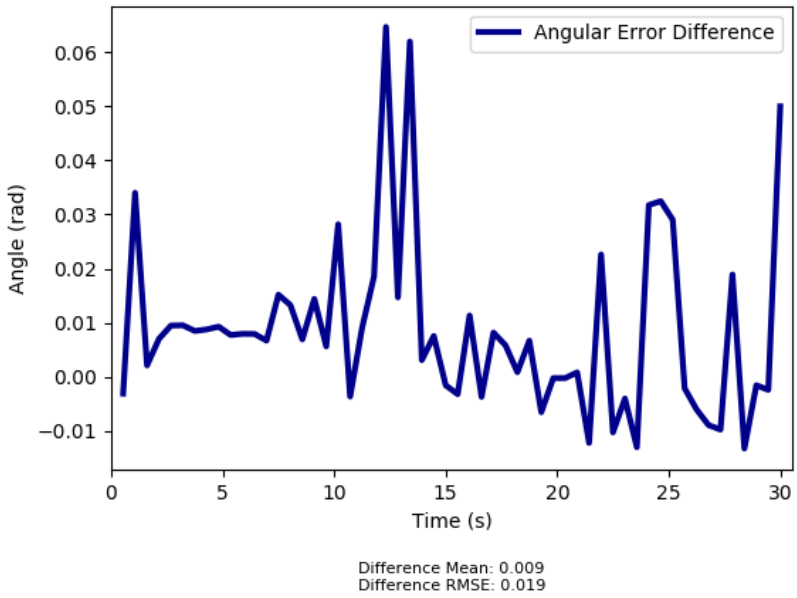


Figure 5.5: Difference between the true and estimated angular error of a random job

method, a get results statistically more important, a second experience is performed. It consists in running 50 jobs with a random parameter set and analyzing the results, comparing the estimated errors to the true errors. This experiment is presented in figures 5.7 to 5.12. In figure 5.7, is presented the mean position error of each one of the 50 jobs. The blue dots represent the estimated error while the green dots represent the true error. We observe that from these 50 jobs, there are cases where the true error is around 0.05 m (5 cm). There is also cases where the error is above 0.30 m (30 cm). This means

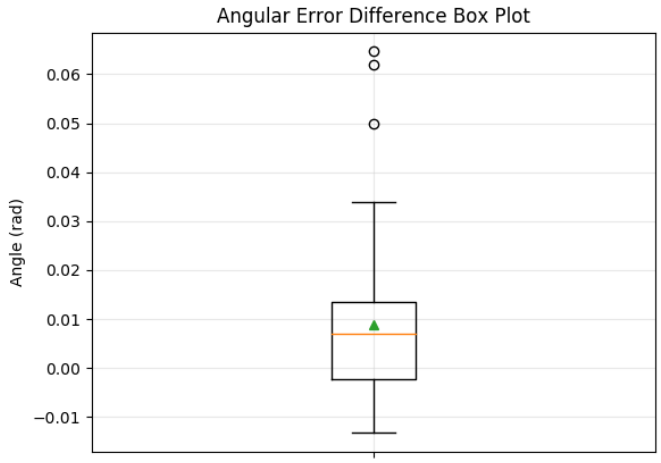


Figure 5.6: Box plot of the difference between the true and estimated angular error of a random job

that this experience covers situations between the two extremes, when the robots is well localized and when the robot is lost.

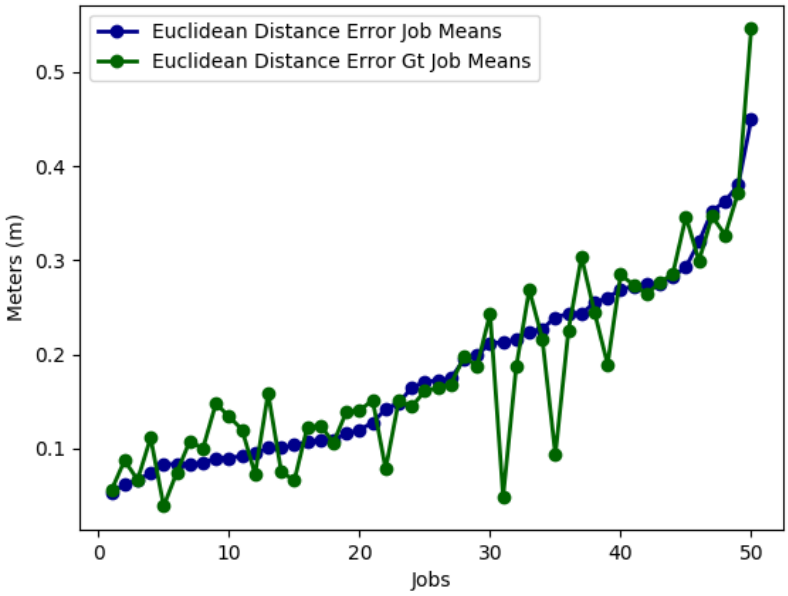


Figure 5.7: True and estimated mean position error for 50 random jobs

To better understand how close the estimated errors are from the true errors, in figure 5.8 is shown the difference between the true mean error and the estimated mean error of each job. These differences resulted in a RMSE of 0.079 m (7.9 cm), with a maximum difference of 0.16 m (16 cm) in absolute value and a minimum difference in the order of millimeters. With this information and remembering figure 5.7, we conclude that the estimated error is sometimes above and sometimes below the true error and that the biggest difference occurred for a true mean position error of 0.05 m (5 cm), while having 0.21 m (21 cm) as estimated. This is not a good result, since a parameter set with a mean true position error

of 5 cm is a very good result, so it has a strong possibility of belonging to the Pareto frontier. An over estimation of 16 cm would completely throw it away of the Pareto frontier. The opposite situation (large under estimation of the position error), did not occur in this experiment.

Figure 5.8: Position error difference for 50 random jobs

In figure 5.9 is displayed the box plot associated to the previous figure. There are indeed some outlier points, confirming that there are experiments where the proposed methods was not capable of performing a good position error estimation. Excluding these outliers, the difference ranges between -0.07 m (-7 cm) and 0.06 m (6 cm) with a median value of 0 m. This suggests that in the majority of cases, the proposed method will be capable of estimating the position error within 7 cm of error in its absolute value.

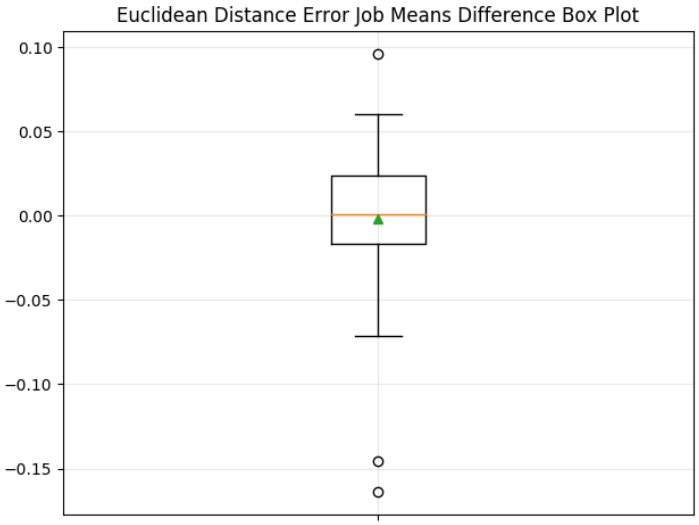


Figure 5.9: Box plot of the position error difference for 50 random jobs

Now, for the angular error, in 5.10 we see the mean angular error of each one of the 50 jobs. The blue dots represent the estimated error while the green dots represent the true error. We observe that from these 50 jobs, as for the position error, there are cases where the robot is well orientated, and cases where it is completely disoriented. At first sight, there's seem to be a good prediction by the developed package.

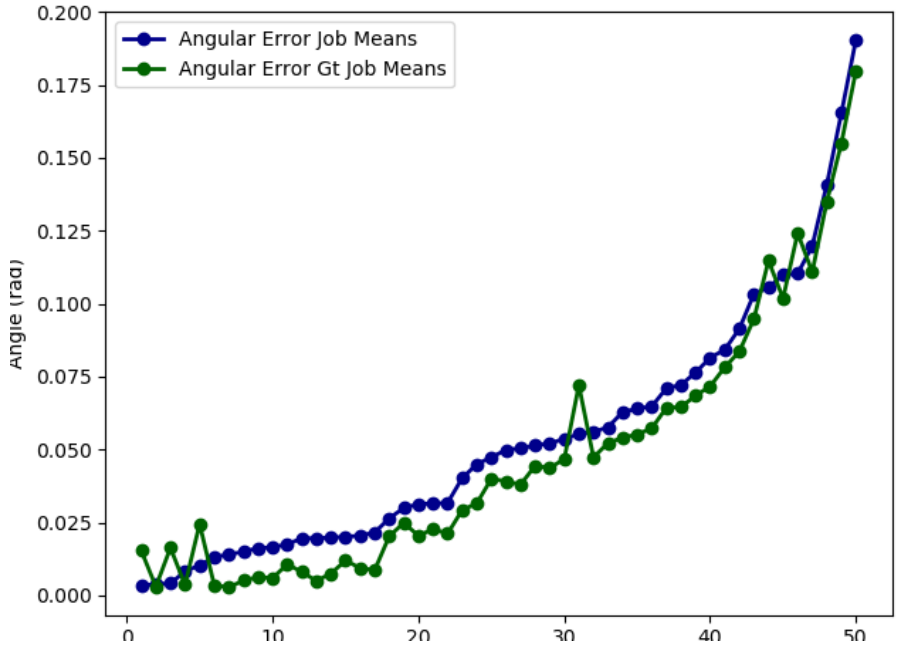


Figure 5.10: True and estimated mean angular error of 50 random jobs

To better analyze these results, in 5.11 is displayed the difference between the true mean angular error and the estimated mean angular error of each job. These differences resulted in a RMSE of 0.024 rad (1.38°), and a maximum difference in absolute value of 0.017 rad (0.97°). The minimum difference was of 0.005 rad (0.29°) in absolute value.

Finally, in figure 5.12 is shown the box plot of the previous figure data. As in the position error case, there is also one outlier point. The fourth quartile is very disperse, so approximately 75% of the differences are between 0.008 rad (0.46°) and 0.015 rad (0.86°). These are very small positive values, suggesting that the developed method has a good precision estimating the angular error, only with very small underestimation.

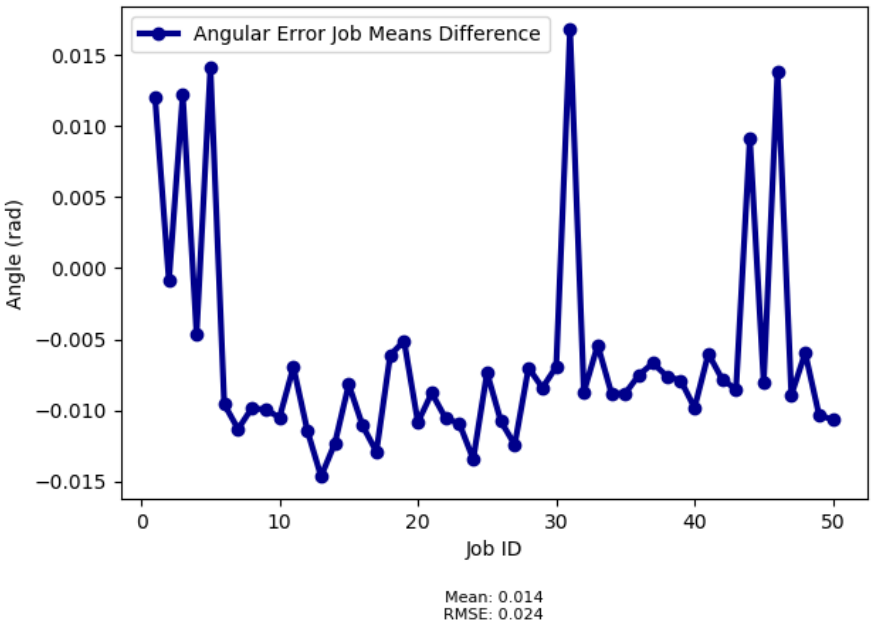


Figure 5.11: Angular error difference for 50 random jobs

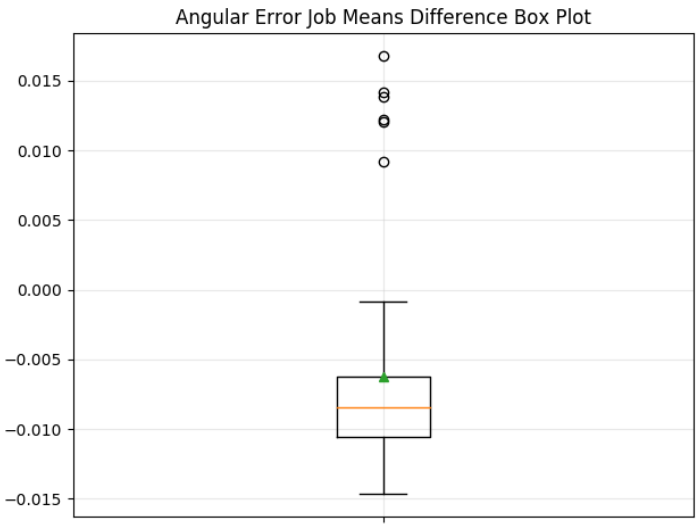


Figure 5.12: Box plot of the angular error difference for 50 random jobs

5.2 Ground Truth Pareto Frontier

The results of the MOO of AMCL are shown in figure 5.13. For this solution, the position error was computed based on the ground truth mo-cap data.

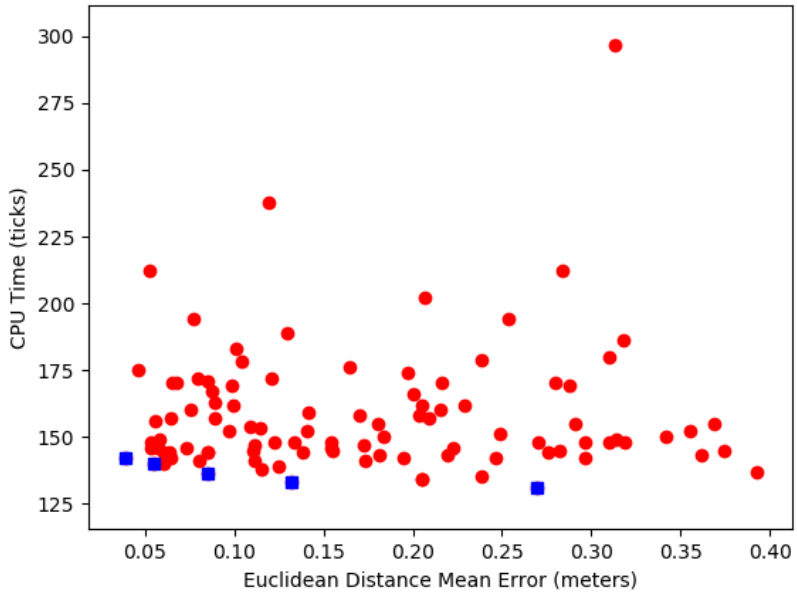


Figure 5.13: Pareto frontier: Ground truth position error - CPU time

Figure 5.14 is a zoomed portion of figure 5.13, showing only the region of interest, the Pareto frontier.

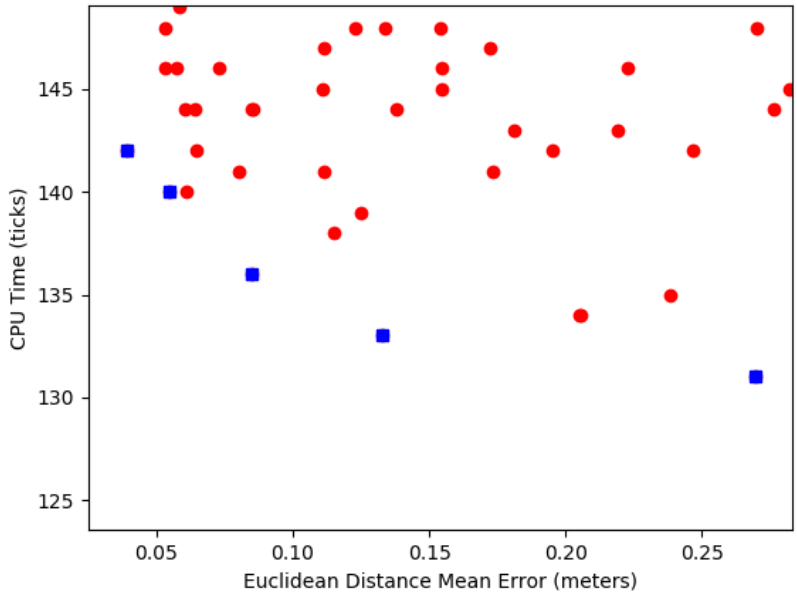


Figure 5.14: Pareto frontier Zoomed: Ground truth position error - CPU time

The Pareto frontier points and its associated objective functions evaluations are listed in table 5.1.

Table 5.1: Pareto frontier points. CPU Time vs Euclidean Distance Ground Truth Error

Point	**1**	**2**	**3**	**4**	**5**
Euclidean Distance Error GT	0.039	0.055	0.085	0.133	0.270
CPU Time	142	140	136	133	131

As table 5.1 indicates, PESMO was capable of finding a Pareto frontier with 5 points. The position error ranges from 0.039 m (3.9 cm) to 0.270 m (27.0 cm) and the CPU time consumption ranges from 131 ticks to 142 ticks. This results are acceptable since the first three points have an error below 9 cm, which are in fact usable parameter configurations. It is notable that the CPU time consumption of the Pareto front are very similar form one another. This rises the question on how conservative were the the limits applied to the design space listed under table 4.1, preventing the evaluations of certain regions of space.

5.2.1 Ground Truth Solution Validation

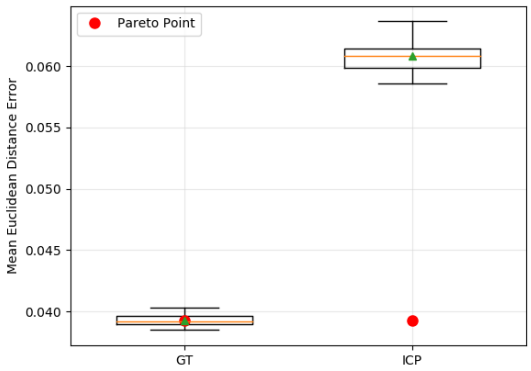
In order to validate and understand how consistent these parameter sets from the Pareto frontier are, each Pareto point is reevaluated 25 times with a different recorded data. With this, we can verify how these parameter sets behave in situations different from the training data. This test data was recorded in the same test bed, has the same duration but a different path with different movements. It is also convenient to verify, how the *Point Cloud ICP Scorer* would predict the position error for this Pareto points. Table 5.2 list the mean, standard deviation and RMSE results of evaluation the previous Pareto points 25 times with the test data. At bold are the previously shown Pareto points obtained with PESMO.

Table 5.2: Ground truth Pareto frontier validation results

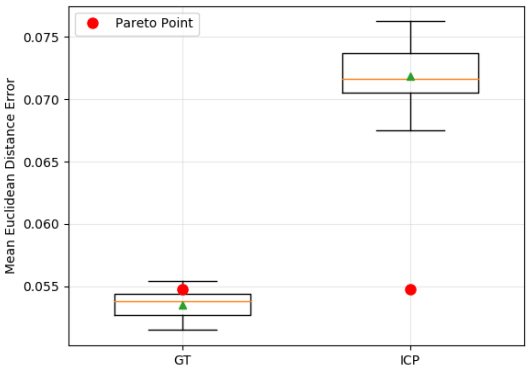
Point	**1**	**2**	**3**	**4**	**5**
Euclidean Distance Error GT	**0.039**	**0.055**	**0.085**	**0.133**	**0.270**
Mean GT	0.039	0.054	0.084	0.130	0.269
Mean ICP	0.061	0.072	0.081	0.165	0.246
Std GT	0.000	0.001	0.001	0.004	0.015
Std ICP	0.001	0.002	0.002	0.004	0.012
RMSE GT	0.000	0.002	0.002	0.005	0.015
RMSE ICP	0.022	0.017	0.004	0.033	0.026
CPU Time	**142**	**140**	**136**	**133**	**131**
Mean	146	146.96	146.08	136	133.32
Std	2.236	4.521	3.224	2.905	2.328
RMSE	5.441	8.299	10.583	4.205	3.286

Analyzing the previous table, it is convenient to separate the conclusions for each evaluation function. Firstly, the ground truth position error is very consistent, with a maximum difference of 0.003 m (3 mm) in point 4. With this results, one can be confident in saying that using the mo-cap system for evaluating the pose error, and using this evaluation as an objective function in single or MOO algorithms yields good results. Secondly, comparing the ground truth results to the *Point Cloud ICP Score* results, we see that the estimated position error had a maximum difference of 0.032 m (3.2 cm) in point 4 and a minimum difference of 0.003 m (3 mm) in point 3. The standard deviation and the RMSE of both previous

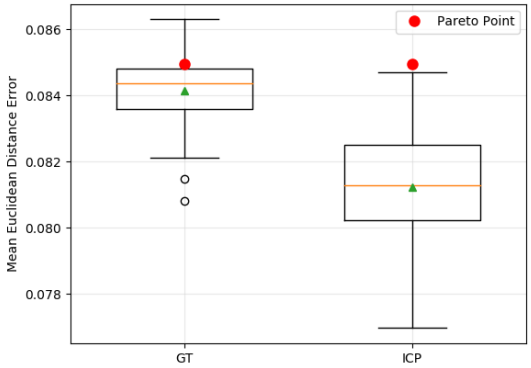
evaluation methods are in the order of millimeters, suggesting a consistency of the values along the 25 evaluations. The boxplots regarding this results are displayed in figure 5.15.[1].



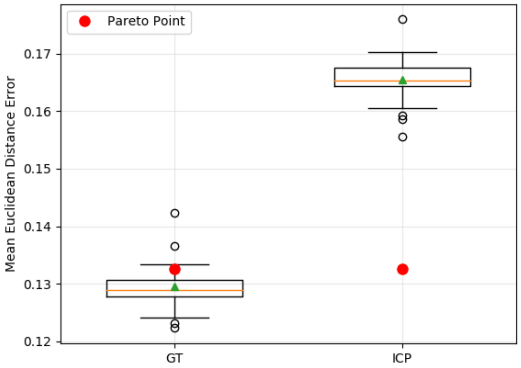
(a) Point 1



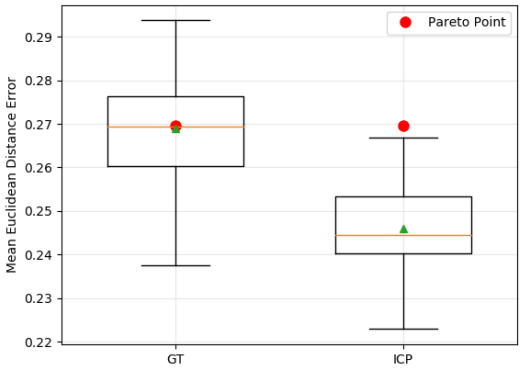
(b) Point 2



(c) Point 3



(d) Point 4



(e) Point 5

Figure 5.15: Boxplots for each Pareto point comparing the ground truth data (GT) to the estimated data (ICP)

With the help of table 5.3, we verify that the acquisition of the position error with the ground truth sensor yielded the best results except for point 4 and point 5, represented in sub figure 5.15d and

[1]The box plots of this picture have different scales to better compare the true mean euclidean distance error and the estimated. Comparison between plots can be done in the auxiliary of table 5.3

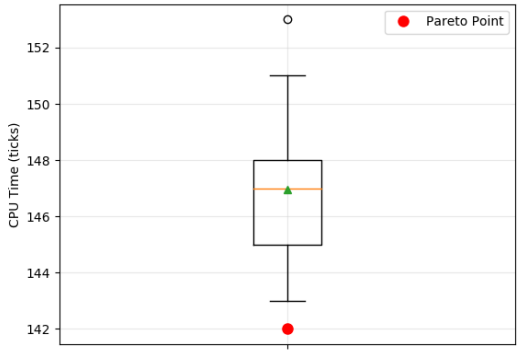
subfigure 5.15e, where there was some outlier points on point 4 and the maximum dispersion of 0.054 m (5.4 cm) between the lowest and biggest value on point 5. These two points are the the ones with higher position error and usually will not be used in the robot. The *Point Cloud ICP Scorer* method had a slightly lower dispersion in the results, with a maximum range of 0.044 m (4,4 cm) also for point 5. This point also had the biggest estimated position error given by this method and certainly would not be chosen. Even though this method predicted the position error with a error up to 3.2 cm, it was able to correctly predict the order of the Pareto points which is a good result.

Table 5.3: Ground truth Pareto frontier validation box plots data

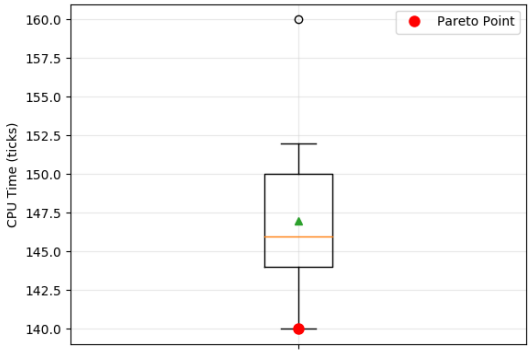
Point	**1**	**2**	**3**	**4**	**5**
GT Median	0.039	0.054	0.084	0.129	0.269
GT Lower Quartile	0.039	0.053	0.084	0.128	0.26
GT Upper Quartile	0.040	0.054	0.085	0.131	0.276
GT Minimum	0.038	0.051	0.082	0.124	0.238
GT Maximum	0.040	0.055	0.086	0.133	0.294
ICP Median	0.061	0.072	0.081	0.165	0.244
ICP Lower Quartile	0.060	0.071	0.080	0.164	0.240
ICP Upper Quartile	0.061	0.074	0.082	0.168	0.253
ICP Minimum	0.059	0.067	0.077	0.161	0.223
ICP Maximum	0.064	0.076	0.085	0.170	0.267
CPU Time Median	147	146	146	135	133
CPU Time Lower Quartile	145	144	143	134	132
CPU Time Upper Quartile	148	150	148	138	134
CPU Time Minimum	143	140	143	131	130
CPU Time Maximum	151	152	151	144	137

Lastly, still referring to table 5.2 for the CPU time, we detect that the results are not consistent since that the difference between the Pareto points CPU time and the reevaluation mean value reveals a wrong order of the points in terms of CPU time. For example, point 1 should be the point with the maximum value of CPU time, however, it was point 2 who yields that value. The maximum difference was of 10 ticks and the minimum difference was of 2 ticks. This results compromises the Pareto frontier in terms of the CPU time. To have a further look and better understand the CPU time results, figure 5.16 illustrates the box plots for each Pareto points.

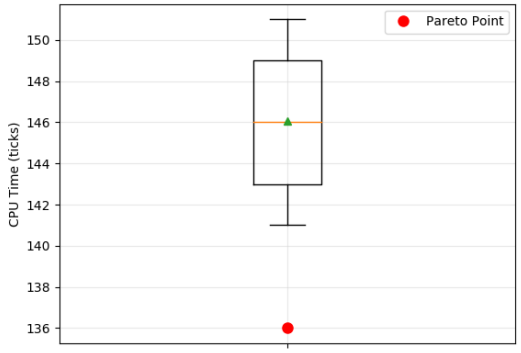
Again, with the help of table 5.3, we confirm that the CPU time evaluation function yielded results with a large dispersion of up to 13 ticks. In subfigures 5.16a and 5.16c the Pareto frontier point is actually in the outlier zone of the box plot, and for the remaining sub figures, the Pareto points fit in the first quartile confirming that the CPU time results are not reliable. A possible explication for these results are the stochastic nature of the AMCL algorithm and the variable amount of particles due to its adaptive approach. From run to run, the number of particles used by AMCL to estimate the robot pose might not be consistent, resulting in changes of the CPU time.



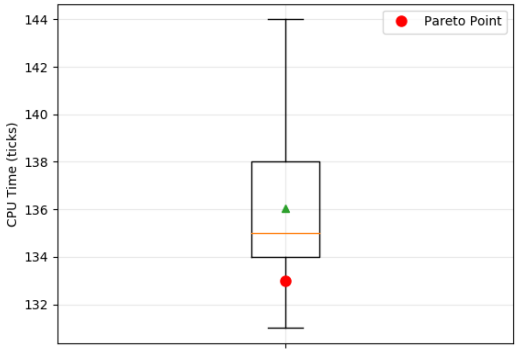
(a) Point 1



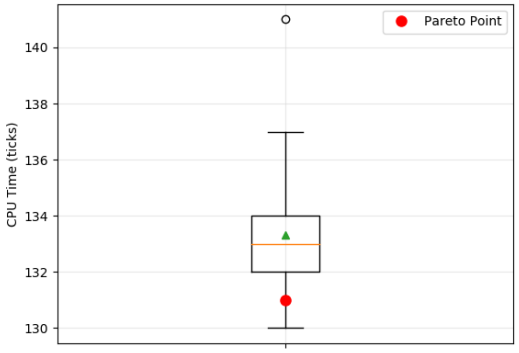
(b) Point 2



(c) Point 3



(d) Point 4



(e) Point 5

Figure 5.16: Boxplots for each Pareto point of the CPU time

5.3 Point Cloud ICP Scorer Pareto Frontier

The same analysis done in section 5.2 is now applied for the results of the MOO of AMCL now with the *Point Cloud ICP Scorer* method as evaluation function of the position error. The achieved Pareto frontier is presented in figure 5.17.

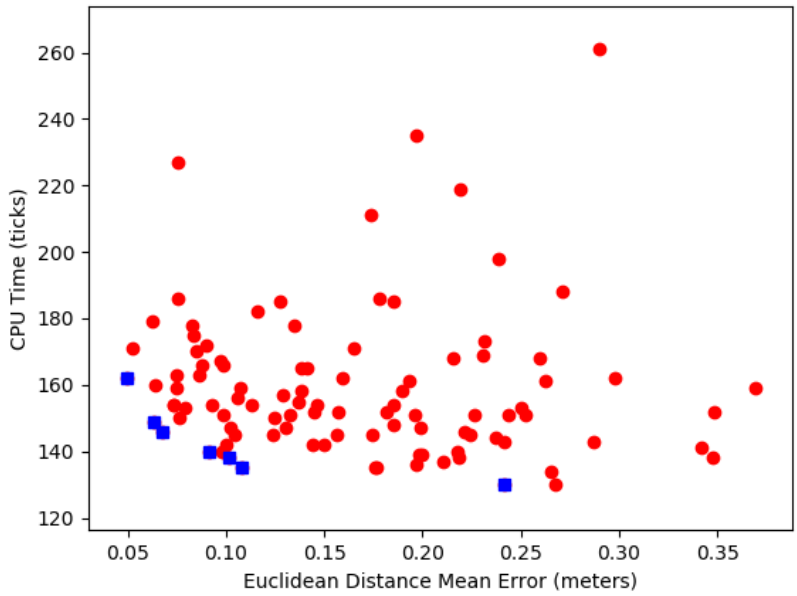


Figure 5.17: Pareto frontier: Ground truth position error - CPU time

Figure 5.18 is a zoomed portion of figure 5.17, highlighting the Pareto frontier.

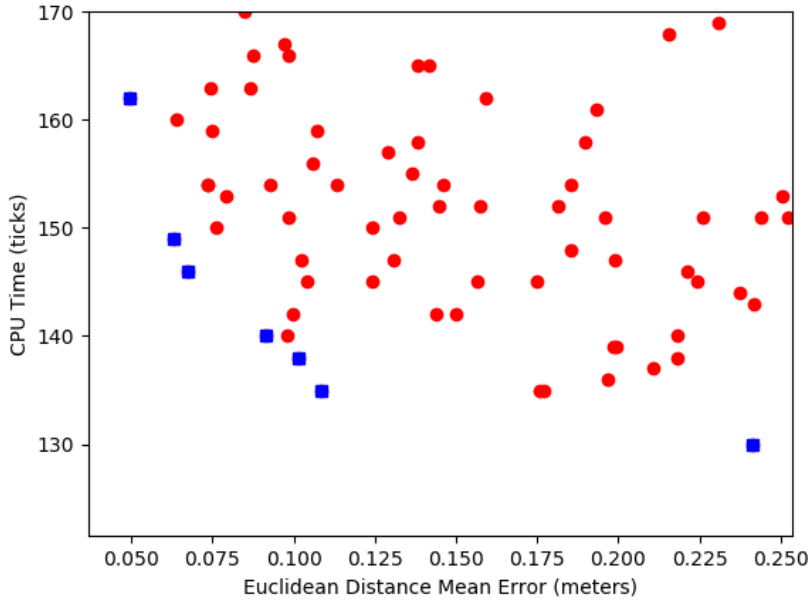


Figure 5.18: Pareto frontier zoomed: Estimated position error - CPU time

The Pareto frontier points and the respective objective functions evaluations are listed in table 5.4.

Table 5.4: Pareto frontier points. CPU Time - Estimated Euclidean Distance Error

Point	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Euclidean Distance Error ICP	0.049	0.063	0.067	0.091	0.102	0.108	0.242
CPU Time	162	149	146	140	138	135	130

As table 5.4 indicates, PESMO was also able of finding a Pareto frontier with these objective functions, now with 7 points. The position error ranges from 0.049 m (4.9 cm) to 0.242 m (24.2 cm) and the CPU time consumption ranges from 130 ticks to 162 ticks. Comparing with the results yielded in section 5.2, the minimum euclidean distance error was 1 cm bigger (4.9 cm instead of 3.9 cm) and the maximum euclidean distance error was 2.8 cm smaller (24.2 cm instead of 27 cm). For the CPU time, the bigger difference was that the point with minimum position error now has a bigger CPU time. This difference was of 20 ticks (162 ticks instead of 142 ticks).

5.3.1 Point Cloud ICP Scorer Solution validation

Our goal now is to validate and understand how consistent these parameter sets from the Pareto frontier are. Again, each Pareto point is reevaluated 25 times with a different recorded data. With this, we can verify how these parameters sets behave in situations different from the training data. This test data is exactly the same of the one used in the previous section. The idea is also to verify how would the mo-cap system evaluate these points, and compare it to the developed method. This is exactly what was done in the previous section, but in an opposite order, resulting in a cross check validation. Table 5.5 lists the mean, standard deviation and root mean square error results of evaluation the previous Pareto points 25 times with the test data. At bold are the previously shown Pareto points obtained with PESMO.

Table 5.5: Estimated Pareto frontier validation results

Point	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Euclidean Distance Error ICP	0.049	0.063	0.067	0.091	0.102	0.108	0.242
Mean GT	0.094	0.096	0.092	0.077	0.072	0.136	0.258
Mean ICP	0.050	0.065	0.067	0.098	0.104	0.115	0.245
Std GT	0.002	0.002	0.002	0.003	0.007	0.004	0.02
Std ICP	0.001	0.002	0.001	0.004	0.004	0.003	0.019
RMSE GT	0.044	0.033	0.025	0.014	0.030	0.028	0.026
RMSE ICP	0.001	0.002	0.001	0.008	0.005	0.007	0.019
CPU Time	162	149	146	140	138	135	130
Mean	164	151.44	153.48	139.56	138.96	142.6	138.4
Std	3.572	4.167	4.215	2.547	2.891	2.993	5.838
RMSE	4.238	4.829	6.151	2.585	3.046	8.168	10.229

The estimated position error is again very consistent, with millimetric order in standard deviation and RMSE and with a maximum difference of 0.007 m (7 mm) in point 4 and 6. Comparing these results to the respective ground truth values, we see that there was a maximum difference of 0.045 m (4.5 cm) for point 1. Checking this difference for the remaining points, we conclude that the proposed method was not able to find a completely true Pareto frontier. *Point Cloud ICP Scorer* predicted that point 1 would

have the lowest position error, while truly it was point 5 that had the lowest position error. This prediction error of 4.5 cm is consistent with what was concluded in section 5.1, that the proposed method would be able to estimate the true position error within 7 cm of error. It is also noted that the three estimations of the lower position error (point 1, 2 and 3), were all under estimations. As for points 4 and 5 (that generally would still be considered usable given the estimated error), they were overestimated. As expected, and even though that in this case the Pareto points had a maximum prediction error of 4.5 cm, we conclude that for the position error evaluation, a ground truth sensor data would be the best option, however, with the absent of it, the proposed *Point Cloud ICP Scorer* will give very similar results, usable if the user can deal with a prediction error up to approximately 7 cm. The box plots regarding this results are displayed in figure 5.19.

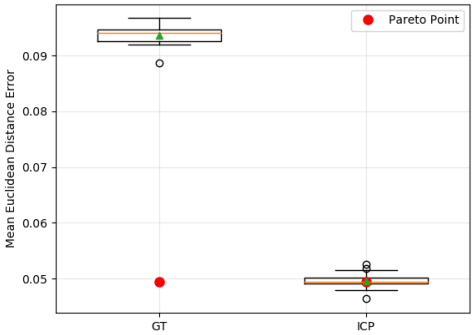
In table 5.6 is listed the data associated to the previously box plots, and we verify that the acquisition of the estimated position error was consistent except for point 4 and point 6, represented in subfigure 5.19d and sub figure 5.19f, where both point are in the outlier zone. The maximum dispersion was of 0.016 m (1.6 cm) also for point 4. We conclude that, as has happened in the previous section, there was a small over fit to the training data, since point 4 and 6 do not fall inside the respective box plots. We also conclude that even tough this method yields results as consistent as the ground truth method, it is not able to exactly predict the position error. It comes to the user to decide if its robotic application can handle an error of up to approximately 7 cm.

Table 5.6: Estimated Pareto frontier validation box plots data

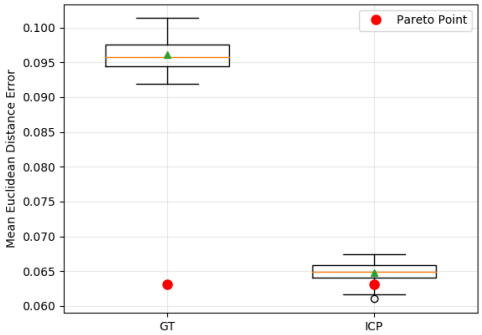
Point	**1**	**2**	**3**	**4**	**5**	**6**	**7**
GT Median	0.094	0.094	0.093	0.076	0.073	0.136	0.255
GT Lower Quartile	0.093	0.093	0.092	0.075	0.068	0.113	0.246
GT Upper Quartile	0.095	0.096	0.093	0.080	0.077	0.139	0.270
GT Minimum	0.092	0.089	0.091	0.073	0.062	0.13	0.217
GT Maximum	0.097	0.100	0.095	0.083	0.083	0.145	0.291
ICP Median	0.049	0.075	0.067	0.098	0.105	0.114	0.243
ICP Lower Quartile	0.049	0.074	0.066	0.095	0.101	0.112	0.233
ICP Upper Quartile	0.050	0.076	0.067	0.101	0.107	0.117	0.256
ICP Minimum	0.048	0.072	0.064	0.093	0.097	0.109	0.204
ICP Maximum	0.052	0.077	0.069	0.109	0.110	0.122	0.290
CPU Time Median	164	151	153	139	138	142	138
CPU Time Lower Quartile	161	150	151	138	138	141	135
CPU Time Upper Quartile	166	153	156	140	140	144	140
CPU Time Minimum	158	146	149	136	135	137	132
CPU Time Maximum	172	155	160	143	142	148	144

To finalize the analyses of this Pareto frontier, we record table 5.6 and give a look into figure 5.20.

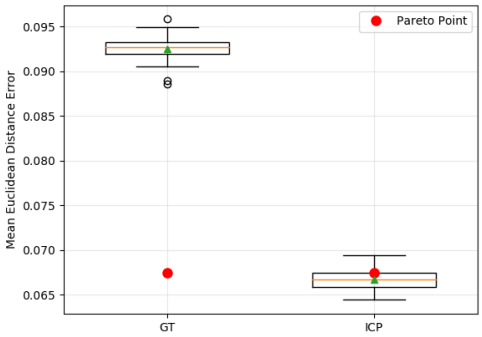
We see that the CPU time evaluation function results are as dispersed as in the previous section. This dispersion was up to 14 ticks for point 1 (subfigure 5.20a). In subfigures 5.20c. 5.20e and 5.20g the respective Pareto frontier point is in the outlier zone of the box plot, and for the remaining sub figures, the Pareto points fit in second or third quartile. We conclude that the CPU time objective function evaluations are not reliable, compromising the Pareto frontier, as has happened in the previous section.



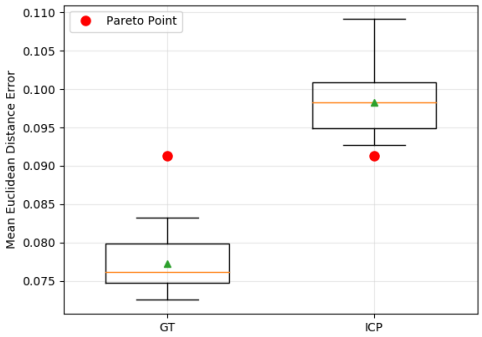
(a) Point 1



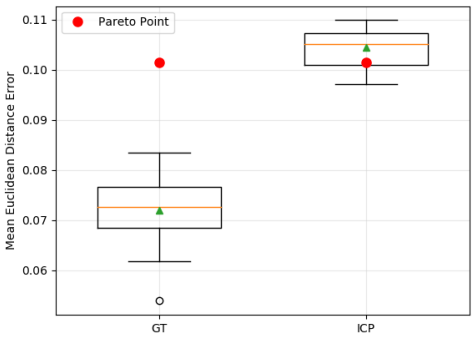
(b) Point 2



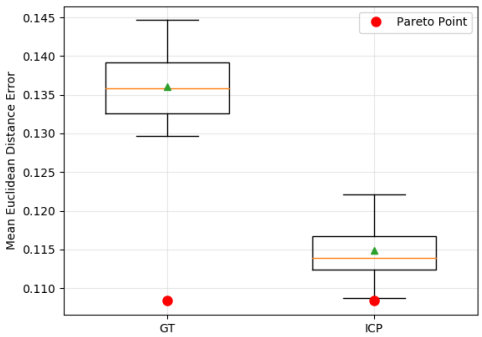
(c) Point 3



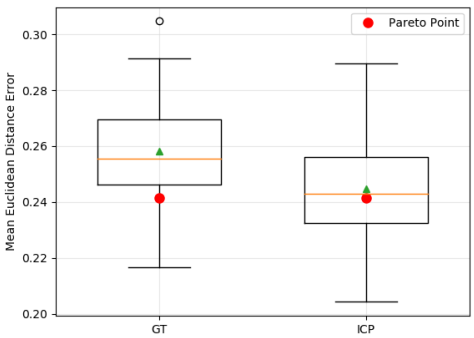
(d) Point 4



(e) Point 5

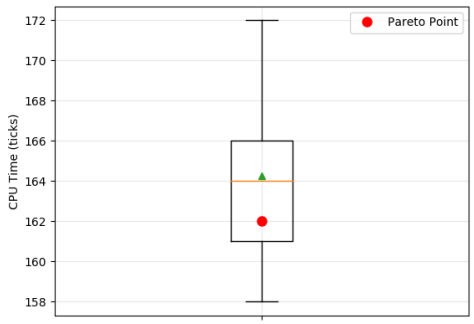


(f) Point 6

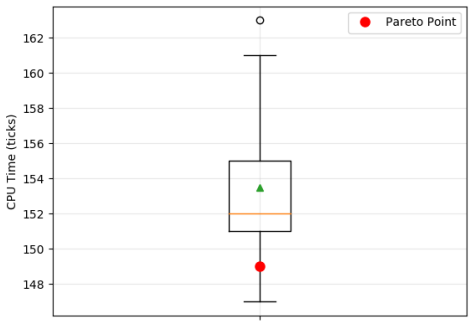


(g) Point 7

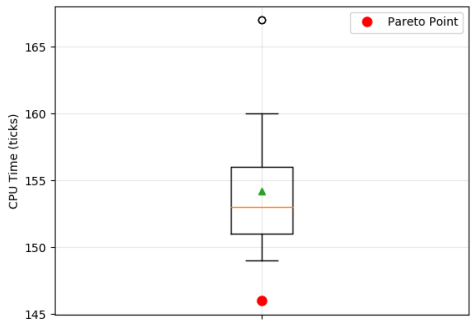
Figure 5.19: Box plots for each Pareto point comparing the estimated data (ICP) to the ground truth data (GT)



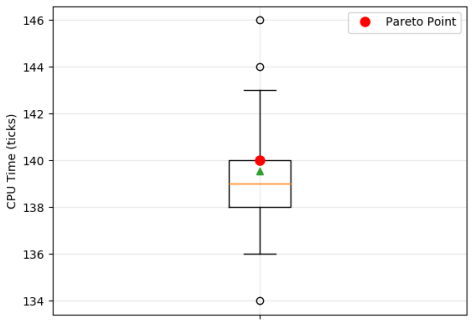
(a) Point 1



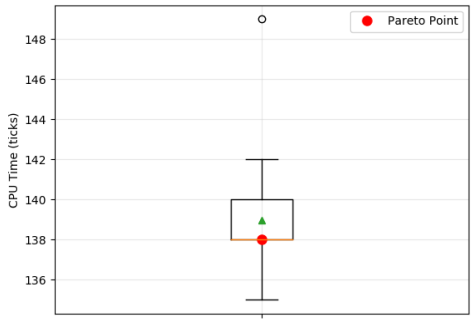
(b) Point 2



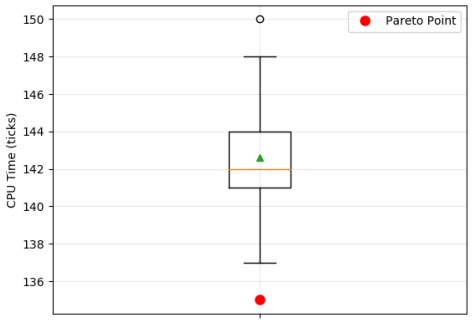
(c) Point 3



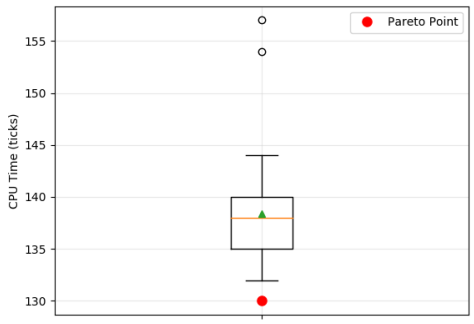
(d) Point 4



(e) Point 5



(f) Point 6



(g) Point 7

Figure 5.20: Box plots for each estimated Pareto point CPU time

5.4 Stronger Pareto Frontier

As concluded in the two previous sections, the achieved Pareto frontiers are compromised by the very stochastic nature of the CPU time consumption by the AMCL algorithm. To surpass this step back, a new optimization was done, now reevaluating 10 times each suggested point by PESMO. Since each job takes at least 30 seconds to conclude, plus the time taken by PESMO to perform the optimization process, the number of reevaluations was 10 simply to avoid a extremely high optimization time. The goal of this approach is to suppress the wrong prediction of the CPU time consumption, achieving a more reliable Pareto frontier, both when the position is computed based on ground truth sensor or when is estimated by the *Point Cloud ICP Scorer*. Since it was already verified that the acquisition of the position error using the mo-cap system yields very good results, only the *Point Cloud ICP Scorer* method was used for computing this new stronger and more faithful Pareto front. The obtained results are shown in figure 5.21.

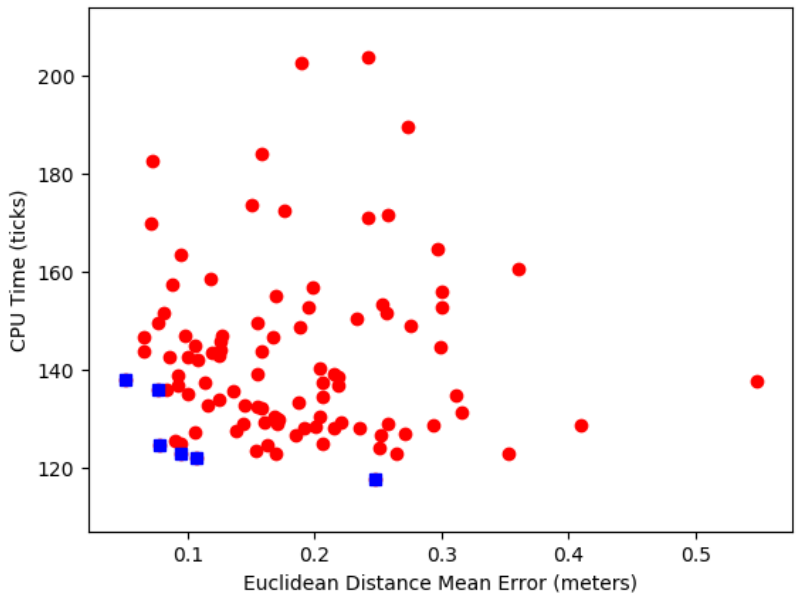


Figure 5.21: Stronger Pareto frontier: Estimated position error - CPU time

Figure 5.22 is a highlight of the achieved Pareto frontier.

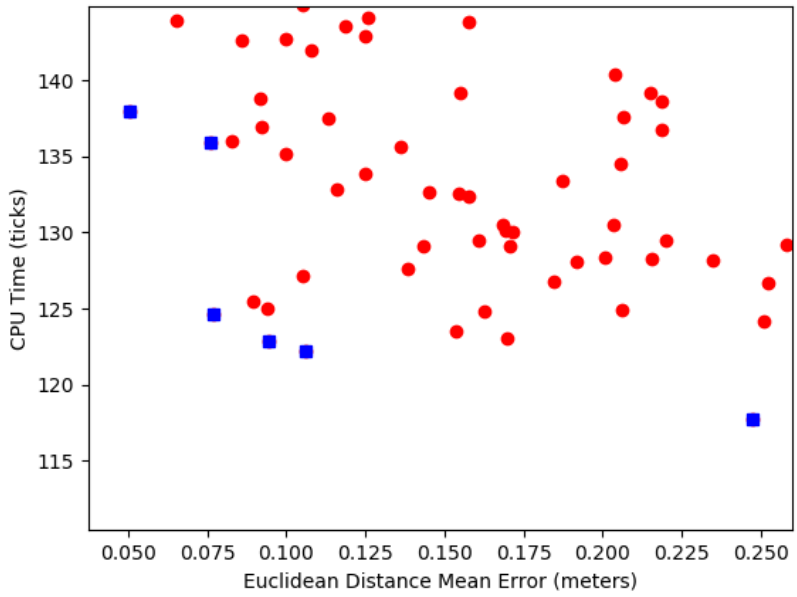


Figure 5.22: Stronger Pareto frontier zoomed: Estimated position error - CPU time

The Pareto points and respective objective functions evaluations are listed in table 5.7.

Table 5.7: Stronger Pareto frontier points. CPU time - Estimated Euclidean Distance Error

Point	**1**	**2**	**3**	**4**	**5**	**6**
Euclidean Distance Error ICP	0.050	0.076	0.077	0.094	0.106	0.248
CPU Time	138	135.9	124.6	122.9	122.2	117.7

Table 5.7 shows that 6 Pareto points were found. The estimated position error ranges between 0.050 m (5 cm) and 0.248 m (24.8 cm), while the CPU time consumption ranges from 117.7 to 138 ticks. The range of values for the estimated euclidean distance error is very similar to the ones obtain under section 5.3, which might evidence a lower and upper bound on the position error that the proposed method can compute. The lower bound is associated to the position error that will be always present in AMCL. The upper bound can be associated to the previously noticed limitation of the developed method, of not being able to compute a reliable position error estimation for high true position errors. When it comes to the range of values for the CPU time consumption, these were smaller than the one found under section 5.3. This might be a result of reevaluating 10 times each suggested point by PESMO.

In order to check how more reliable this new Pareto frontier is, each Pareto point is again reevaluated 25 times with a testing data.[2] Table 5.8 lists the mean, standard deviation and RMSE results of evaluation each Pareto point 25 times with the training data.

As we seen in the previous sections, the values obtained for the estimated position error are very consistent between the 25 evaluation, with low standard deviations and RMSEs. Looking at the *Mean GT* and *Mean ICP* values, that are the mean values of reevaluating the position error 25 times using the ground truth sensor and the *Point Cloud ICP Scorer* package, we notice that, for the same point, they

[2]This testing data is different than the one used in section 5.3 in order have more data available on how the Pareto frontiers that were found behave in different situations

Table 5.8: Estimated stronger Pareto frontier validation results

Point	**1**	**2**	**3**	**4**	**5**	**6**
Euclidean Distance Error ICP	0.050	0.076	0.077	0.094	0.106	0.248
Mean GT	0.135	0.068	0.092	0.101	0.226	0.194
Mean ICP	0.15	0.057	0.105	0.089	0.183	0.232
Std GT	0.002	0.002	0.003	0.001	0.003	0.006
Std ICP	0.004	0.057	0.002	0.002	0.002	0.005
RMSE GT	0.085	0.008	0.015	0.006	0.120	0.053
RMSE ICP	0.100	0.019	0.028	0.005	0.076	0.016
CPU Time	138	135.9	124.6	122.84	122.2	117.7
Mean	152.08	152.96	138.92	139.84	136.00	133.40
Std	2.712	2.441	3.006	2.648	3.286	4.118
RMSE	14.339	17.234	14.632	17.146	14.186	16.231

are very close to each other, with a minimum difference of 0.011 m (1.1 cm) in point 2 and a maximum difference of 0.043 m (4.3 cm) in point 5. It confirms once again, that the *Point Cloud ICP Scorer* method will give very similar results to the ground truth sensor, once again, with an error in the estimation under the maximum 7 cm predicted in section 5.3. It is also noticeable however, that the first point of the Pareto frontier was extremely over fitted to the training data, since in training it yielded an estimated error of 0.050 m (5 cm), and in testing it yielded an estimated error of 0.15 m (15 cm) and a true error of 0.135 m (13.5 cm). This result, invalidated this Pareto point. The box plots regarding this results are displayed in figure 5.23.

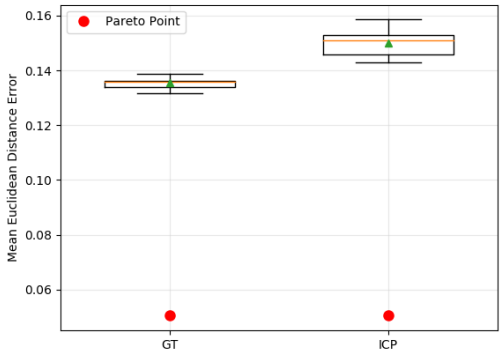
In table 5.9 is listed the results associated to the previous box plots. We notice a slight over fit to the training data, with a maximum difference of approximately 0.085 m (8.5 cm) for point 1, and a minimum difference of 0.005 m (5 mm) for point 4. This emphasizes the importance of a training data with a large variety of movements, to avoid over fit.

Table 5.9: Estimated stronger Pareto frontier validation box plots data

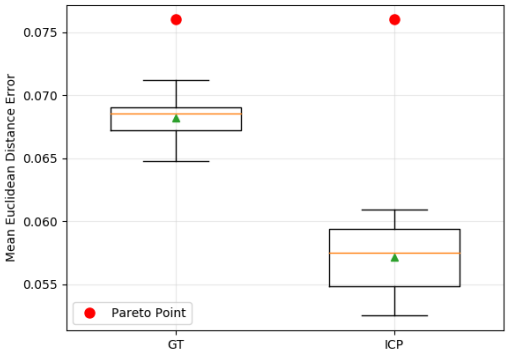
Point	**1**	**2**	**3**	**4**	**5**	**6**
GT Median	0.136	0.069	0.092	0.100	0.225	0.194
GT Lower Quartile	0.134	0.067	0.090	0.100	0.224	0.190
GT Upper Quartile	0.136	0.069	0.094	0.102	0.228	0.199
GT Minimum	0.098	0.065	0.087	0.098	0.219	0.182
GT Maximum	0.103	0.071	0.098	0.103	0.232	0.208
ICP Median	0.089	0.058	0.105	0.089	0.182	0.231
ICP Lower Quartile	0.088	0.055	0.104	0.088	0.181	0.229
ICP Upper Quartile	0.091	0.059	0.106	0.091	0.184	0.237
ICP Minimum	0.086	0.053	0.102	0.086	0.179	0.225
ICP Maximum	0.093	0.061	0.110	0.093	0.187	0.241
CPU Time Median	153	152	138	139	136	133
CPU Time Lower Quartile	150	151	137	138	134	131
CPU Time Upper Quartile	154	154	140	141	138	135
CPU Time Minimum	148	149	133	136	130	128
CPU Time Maximum	160	158	142	143	140	139

Lastly, in figure 5.24 are the box plots for the CPU time consumption of each Pareto point.

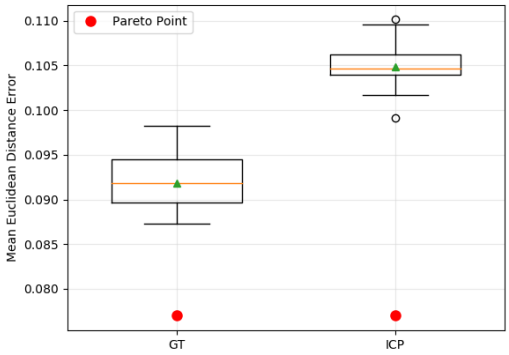
Remembering table 5.8, we notice that even though that again, the reevaluation of the CPU time under the testing data does not give the same results to the ones given by the Pareto frontier, the order



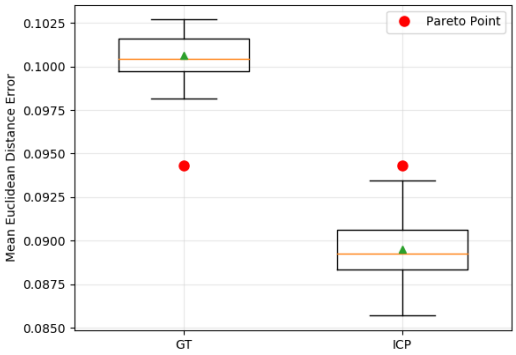
(a) Point 1



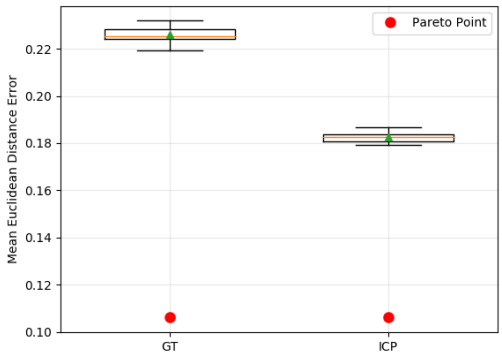
(b) Point 2



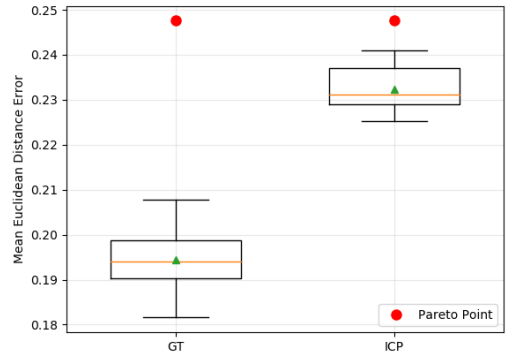
(c) Point 3



(d) Point 4



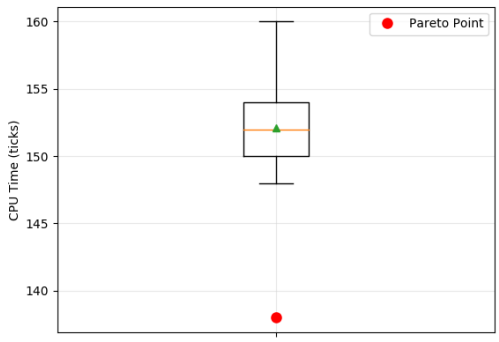
(e) Point 5



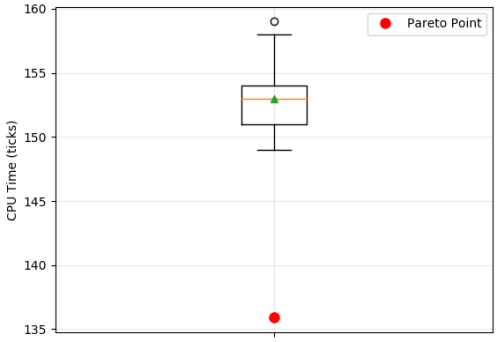
(f) Point 6

Figure 5.23: Estimated stronger Pareto frontier point 6 position box plots

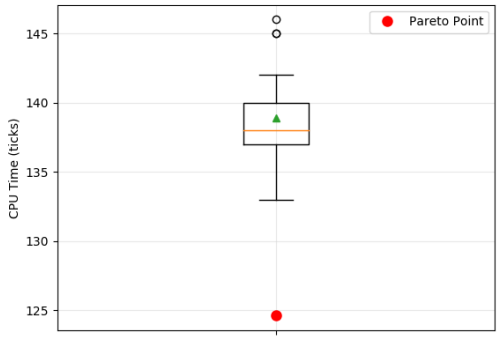
of the Pareto points remained the same, meaning that Pareto point with lowest CPU time still got the lowest CPU time in the reevaluation using the training data. The same for the point with highest CPU time, and the intermediate ones. This result is an improvement from the previous sections. The fact that the values are different between the Pareto points and its reevaluations are also related to the different amount of particles AMCL might need in the testing data, in relation to what it needs in the training data. This is due to different movements and locations the robot is.



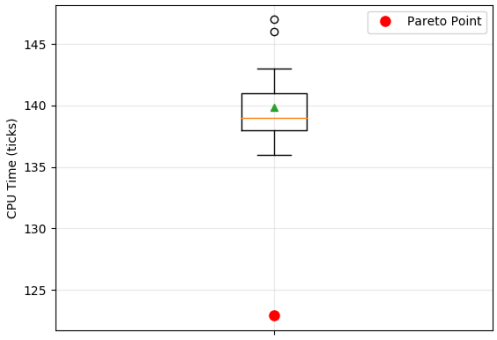
(a) Point 1



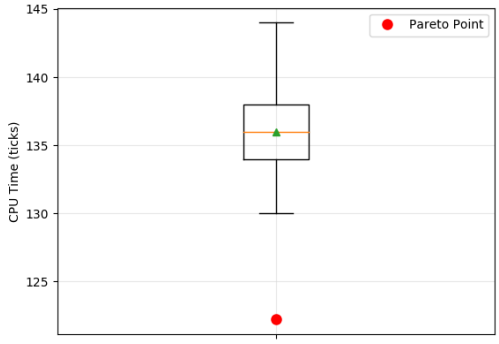
(b) Point 2



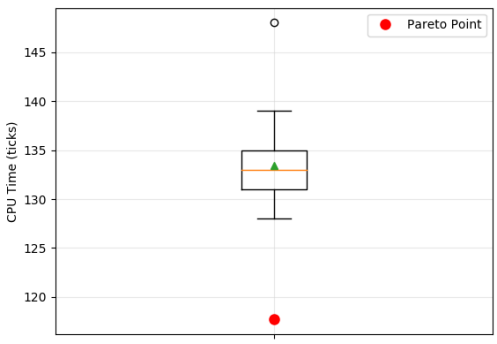
(c) Point 3



(d) Point 4



(e) Point 5



(f) Point 6

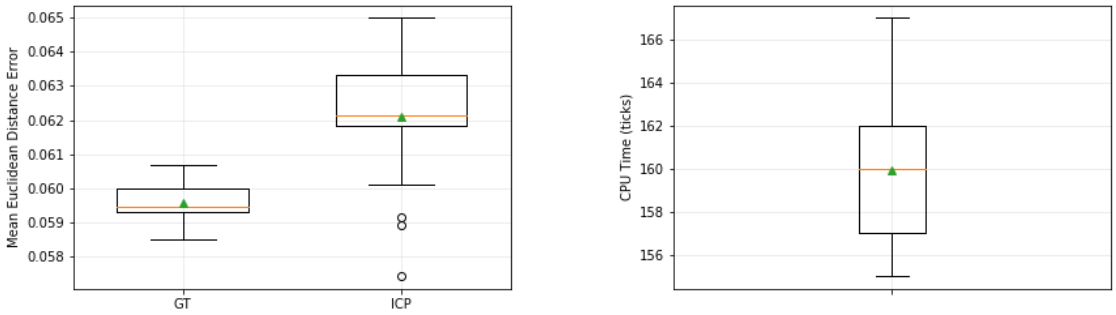
Figure 5.24: Estimated stronger Pareto frontier point 6 CPU time box plots

5.4.1 Default parameters

In one final comparison, the default parameters are also evaluated 25 times under the testing data. The results are exhibited in figure 5.25.

The data associated to the previous figure is listed in table 5.10.

Remembering table 5.8, all Pareto points found outperform the default parameter set in terms of CPU time consumption. However, none of the Pareto points outperform the default parameter set in terms of position error. The Pareto point lowest true position error has a value of 0.068 m (6.8 cm) while the



(a) Default parameters position error box plot

(b) Default parameters CPU time box plot

Figure 5.25: Default parameters box plots

Table 5.10: Default parameters box plots data

GT Mean	0.060
GT Std	0.001
GT Median	0.059
GT Lower Quartile	0.059
GT Upper Quartile	0.060
GT Minimum	0.059
GT Maximum	0.061
ICP Median	0.062
ICP Lower Quartile	0.062
ICP Upper Quartile	0.063
ICP Minimum	0.060
ICP Maximum	0.065
CPU Mean	159.92
CPU Std	3.30
CPU Time Median	160
CPU Time Lower Quartile	157
CPU Time Upper Quartile	162
CPU Time Minimum	155
CPU Time Maximum	167

default parameter set has a true position error value of 0.060 m (6 cm), being 8 mm lower.

Chapter 6

Conclusions and Future Work

From the experimental results of this theses, the following conclusions are taken:

- The chosen MOO algorithm PESMO was able to find a Pareto front of the studied problem. The Pareto front presented a large range of values, specially in terms of the mean euclidean distance error which ranged from very usable values of 0.004 m (4 mm) to values larger than 0.30 m (30 cm). For the CPU time consumption, there was only a difference of up to 40 ticks between the Pareto points with the best a worst mean euclidean distance error. This low difference, might raise the question if the chosen design space was to limited, restricting PESMO of finding Pareto points with low CPU time consumption. It's noticeable however, that for this case, finding these points with lower CPU time consumption might lead to mean euclidean distance error values no longer usable, since in general, the user is interested in position error values below 10 cm.

- The methods used for computing the position error are both very consistent when repeated evaluations are performed. The CPU time consumption results are not as consistent due to the adaptive method of the AMCL, which changes the amount of particles used from one evaluation to another. Due to this aspect and in order to find a more reliable Pareto frontier, it is important to reevaluate each suggestion given by PESMO multiple times and average the results.

- PESMO was also able of finding Pareto points that dominate the default parameter set in CPU time consumption. Using the estimated position error as objective function, a Pareto point with 0.068 m (6.8 mm) of mean true position error and 135.9 ticks of CPU time consumption was found, dominating in terms of CPU time the default parameter set by 24 ticks, but being dominated by 8 mm in terms of position error. Also all Pareto points found outperform the CPU time consumption of the default parameter set.

- The developed method to estimate the position error, *Point Cloud ICP Scorer*, was able to give estimations with a difference of up to 7 cm in relation to the true position error. From all the evaluation performed, this was the worst case scenario. For all the Pareto points of all the Pareto frontiers, the results were slightly better, since the estimations of the position error had a difference to the true position error of up to 4.5 cm.

6.1 Achievements

The main purpose of this work was to design a system that would automatically tune an algorithm in respect to multiple objectives and provide the user freedom to choose the parameters that would best fit the his needs. An already developed framework was extended for also perform MOO. PESMO algorithm was implemented in this framework as is easily used to optimize other algorithms. An optimization of the AMCL algorithm with respect to the position error and CPU time consumption was executed, outperforming the default parameter set in one objective function.

For the second goal of the work, it was proposed and validated a method to estimate the AMCL position error without the use of external tracking hardware. This developed method presented results very similar to the ground truth values, being considered very usable in most situation. This method still has room for improvement.

6.2 Future Work

A proposal for future work would be the optimization of the ICP algorithm to get better results out of the *Point Cloud ICP Scorer*. This proposed package could be extended to not only estimate the pose error in terms of euclidean distance and orientation angle, but to also decompose the position error in x and y Cartesian components. This is useful to compute an estimation of the pose correction one needs to apply to the pose given by the AMCL. With this pose correction, one could implement a method to, when the position error is above certain threshold, send a pose estimation to AMCL, restarting the filter process. If this correction is good enough, AMCL would be able recover the robot's position when he is lost. One application could be, for instance, to solve the kidnap problem.

Lastly, in terms of MOO, it is suggested to study how, not only PESMO but also other MOO algorithms, behave in the optimization of algorithm parameters with more than two objective function. An idea would be to optimize the energy consumption of the robot. Since PESMO presented good results in this problem with a design space of high dimension, something that had never been tested before, it is also interesting to check if for lower dimensional design spaces the results would be better.

Bibliography

[1] D. Hadka and P. Reed. Borg: An auto-adaptive many-objective evolutionary computing framework. *Evolutionary Computation*, 21(2):231–259, 2013.

[2] D. Hernández-Lobato, J. M. Hernández-Lobato, A. Shah, and R. P. Adams. Predictive entropy search for multi-objective bayesian optimization. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1492–1501. JMLR.org, 2016.

[3] R. V. Oscar Lima. A case study on automtic parameter optimization of a mobile robot localization algorithm. Instituto Superior Técnico, Universidade de Lisboa, 2016.

[4] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr 2002.

[6] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, Dec 2007.

[7] T. C. Wang, R. T. Liaw, and C. K. Ting. Moea/d using covariance matrix adaptation evolution strategy for complex multi-objective optimization problems. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 983–990, July 2016.

[8] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evol. Comput.*, 15(1):1–28, Mar. 2007. ISSN 1063-6560.

[9] J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, Feb 2006.

[10] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec 1998.

[11] B. Naujoks, L. Willmes, T. Bäck, and W. Haase. *Evaluating Multi-criteria Evolutionary Algorithms for Airfoil Optimisation*, pages 841–850. Springer Berlin Heidelberg, 2002.

[12] S. F. Adra, A. I. Hamody, I. Griffin, and P. J. Fleming. A hybrid multi-objective evolutionary algorithm using an inverse neural network for aircraft control system design. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 1–8 Vol.1, Sept 2005.

[13] V. Coverstone, J. Hartmann, and W. Mason. Optimal multi-objective low-thrust spacecraft trajectories. 186:387–402, 06 2000.

[14] M. Mahdavian, S. Sudeng, and N. Wattanapongsakorn. Multi-objective optimization and decision making for greenhouse climate control system. In *2016 International Conference on Information Science and Security (ICISS)*, pages 1–5, Dec 2016.

[15] A. Lalbakhsh, M. U. Afzal, and K. P. Esselle. Multiobjective particle swarm optimization to design a time-delay equalizer metasurface for an electromagnetic band-gap resonator antenna. *IEEE Antennas and Wireless Propagation Letters*, 16:912–915, 2017.

[16] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. *Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted-Metric Selection*, pages 784–794. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[17] T. Wagner, M. Emmerich, A. Deutz, and W. Ponweiser. *On Expected-Improvement Criteria for Model-based Multi-objective Optimization*, pages 718–727. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[18] V. Picheny. Multiobjective optimization using gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, 25(6):1265–1280, Nov 2015.

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). Technical Report TR-2010-10, University of British Columbia, Department of Computer Science, 2010. Available online: http://www.cs.ubc.ca/˜hutter/papers/10-TR-SMAC.pdf.

[20] A. Burchardt, T. Laue, and T. Röfer. *Optimizing Particle Filter Parameters for Self-localization*, pages 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[21] J. Kennedy. *Particle Swarm Optimization*, pages 760–766. Springer US, Boston, MA, 2010.

[22] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*., July 1999.

[23] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1322–1328 vol.2, 1999.

[24] D. Fox. Kld-sampling: Adaptive particle filters. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 713–720. 2002.

[25] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[26] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, Oct. 1994. ISSN 0920-5691. doi: 10.1007/BF01427149. URL `http://dx.doi.org/10.1007/BF01427149`.

[27] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[28] J. Messias, R. Ventura, P. Lima, J. Sequeira, P. Alvito, C. Marques, and P. Carriço. A robotic platform for edutainment activities in a pediatric hospital. In *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 193–198, May 2014.