machines

Review

Optimization Techniques in the Localization Problem: A Survey on Recent Advances

Massimo Stefanoni [1,*], Peter Sarcevic [2], József Sárosi [2] and Akos Odry [2]

[1] Doctoral School of Applied Informatics and Applied Mathematics, Obuda University, 1034 Budapest, Hungary
[2] Department of Mechatronics and Automation, Faculty of Engineering, University of Szeged, 6725 Szeged, Hungary; sarcevic@mk.u-szeged.hu (P.S.); sarosi@mk.u-szeged.hu (J.S.); odrya@mk.u-szeged.hu (A.O.)
* Correspondence: massimo.stefanoni@stud.uni-obuda.hu

Abstract: Optimization is a mathematical discipline or tool suitable for minimizing or maximizing a function. It has been largely used in every scientific field to solve problems where it is necessary to find a local or global optimum. In the engineering field of localization, optimization has been adopted too, and in the literature, there are several proposals and applications that have been presented. In the first part of this article, the optimization problem is presented by considering the subject from a purely theoretical point of view and both single objective (SO) optimization and multi-objective (MO) optimization problems are defined. Additionally, it is reported how local and global optimization problems can be tackled differently, and the main characteristics of the related algorithms are outlined. In the second part of the article, extensive research about local and global localization algorithms is reported and some optimization methods for local and global optimum algorithms, such as the Gauss–Newton method, Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), and so on, are presented; for each of them, the main concept on which the algorithm is based, the mathematical model, and an example of the application proposed in the literature for localization purposes are reported. Among all investigated methods, the metaheuristic algorithms, which do not exploit gradient information, are the most suitable to solve localization problems due to their flexibility and capability in solving non-convex and non-linear optimization functions.

Keywords: local optimization; global optimization; mathematical programming; single objective problem; multi-objective problem; metaheuristic algorithms; deterministic algorithms

check for updates

Citation: Stefanoni, M.; Sarcevic, P.; Sárosi, J.; Odry, A. Optimization Techniques in the Localization Problem: A Survey on Recent Advances. *Machines* **2024**, *12*, 569. https://doi.org/10.3390/machines12080569

Academic Editor: Dan Zhang

Received: 9 July 2024
Revised: 9 August 2024
Accepted: 13 August 2024
Published: 19 August 2024

1. Introduction

Optimization is a mathematical discipline which provides tools to select the best element from a set of possible alternatives based on some given criteria. In general, an optimization problem is one of maximizing or minimizing a real function by methodically selecting input values from a permitted set and figuring out the output values of the function itself. Optimization problems can be classified as continuous or discrete, and to solve them, several optimization techniques and approaches have been developed. These techniques can be divided into exact methods, approximate algorithms, metaheuristics, and greedy algorithms [1]. Each of them has pros and cons, and the user must choose the proper and most appropriate technique to solve the specific problem.

Nowadays, optimization, sometimes also referred to as mathematical programming, is used in every scientific field where it is necessary to deal with mathematical problems characterized by high complexity, constraints, interdependencies among variables, and a large space of solution [2]. Finance [3], chemical engineering [4], mechanical engineering [5], electrical engineering [6], and system and database design [7] are only a few examples where optimization has been applied in the last decades. In the field of localization, optimization methods have been widely used to solve localization problems pertaining to robots [8], smartphones [9], permanent magnets [10], vehicles [11], underground structures [12], urban utilities [13], and so on. By referring to this field, this paper aims to investigate and collect the main used optimization methods in the localization literature, such as the Gradient Descent method, Levenberg–Marquardt method, Genetic Algorithm (GA), Particle Swarm Optimization (PSO), etc., by reporting the concept of the used technique and an example of its application in the considered field. Furthermore, the paper also aims to provide in the initial part a general introduction to the classic optimization problem, providing the mathematical formulation, and, in the second part, a clear mathematical formulation of each investigated technique.

The rest of the paper is organized as follows. In Section 2, firstly, the single objective (SO) optimization problem is mathematically defined; then, the local vs global cases are presented, and at the end of the section, the multi-objective optimization (MO) problem with the related approaches is reported. In Section 3, some of the most interesting and important applications of the optimization methods in the field of localization are summarized and classified; for each of them, a brief explanation of the method and the inspired idea are reported. In Section 4, the conclusions are reported.

2. A Brief Overview of the Optimization Problem

2.1. Definition

Optimization methods are used to find a set of inputs to an objective function that results in the evaluation of the minimum or maximum of the function. Mathematically, the optimization problem is defined as follows [14]:

$$\begin{aligned} \text{minimize}: \quad & f_0(x) \\ \text{such that}: \quad & g_j(x) \leq 0 \quad j = 1, \ldots, m \\ & h_k(x) = 0 \quad k = 1, \ldots, p \end{aligned} \tag{1}$$

where $f_0(x)$ is a function, called an objective function, and defined as $f_0 : \mathbb{R}^n \to \mathbb{R}$, $x = (x_1; x_2; \ldots; x_n)^T$ is the optimization vector variable (or the design variables) of the problem, and $g_j(x)$ and $h_k(x)$ are the inequality and equality functions, respectively, that are constraint functions.

If for any z with $g_1(z) \leq 0, \ldots, g_m(z) \leq 0$ and $h_1(z) = 0, \ldots, h_k(z) = 0$, the relation $f_0(x^*) \leq f_0(z)$ is verified, then the vector x^* is the optimal solution of the problem and it has the smallest objective value among all vectors that satisfy the constraint functions. The maximum problem of a function $f(x)$ can be solved considering the minimum problem of $-f(x)$.

The objective function and the constraint functions can be linear or non-linear and an optimization problem can be called a linear program if the objective $(f_0, \ldots, f_m)$ and constraint functions satisfy the linear condition that can be expressed, for example, as follows:

$$f_s(\alpha x + \beta y) = \alpha f_s(x) + \beta f_s(y) \tag{2}$$

where the variable s $x, y \in \mathbb{R}^n$, and the coefficients $\alpha, \beta \in \mathbb{R}$. If the equality in (2) is not verified, the problem is a non-linear program [14]. The problems can also be classified as convex optimization problems if the following condition is satisfied:

$$f_s(\alpha x + \beta y) \leq \alpha f_s(x) + \beta f_s(y) \tag{3}$$

where $\alpha + \beta = 1$, $\alpha \geq 0$, and $\beta \geq 0$. The convex optimization can be considered a generalization of a linear programming because any linear program is also a convex optimization problem [14].

The solution to the problem is found inside the searchable design space that is subjected to the side constraints defined as $x_{iL} \leq x_i \leq x_{iU}$ with $i = 1, \ldots, n$, where variables are limited between the upper and lower bounds [15].

Optimization problems can have some or all of the design variables restricted to integer or discrete values, and they are referred to as integer or discrete optimization problems, respectively.

To find a solution for a class of optimization problems in relation to a given accuracy, a specific algorithm is needed; most of them are able to deal with the side constraints and the constraints function separately because the side constraints are managed directly by the algorithms themselves; unconstrained problems exist, but they can still have side constraints [15].

One or more equality and/or inequality constraints, with or without side restrictions, characterize a constrained optimization problem. For a generic problem, an equality constraint can only either be violated or satisfied, while an inequality constraint can be violated, active, or satisfied. An active inequality constraint, defined as $g_j(x) \leq 0$, is said to be active (or tight) when at a design point x^*, it is satisfied as an equality, i.e., $g_j(x) = 0$ [15].

Optimization techniques, or algorithms, use a specific procedure that is able to find the solution among design variable values that results in the best objective function value, and at the same time, it satisfies all the constraint functions (equality, inequality) and the side constraints; more than one optimum can be found, and they are referred to as local or relative optima.

2.2. A Comparison between Local and Global Extrema

The extreme minimum or extreme maximum of the objective function over the whole input search space is known as a global optimum. In general, finding a local optimum may be rather simple, whereas finding the global optimum may be more challenging, in particular for non-linear functions [16]. As an example, Figure 1 shows a generic function $f(x)$ which has three minima: two of them are of the local type, and one is the global minimum of the problem.

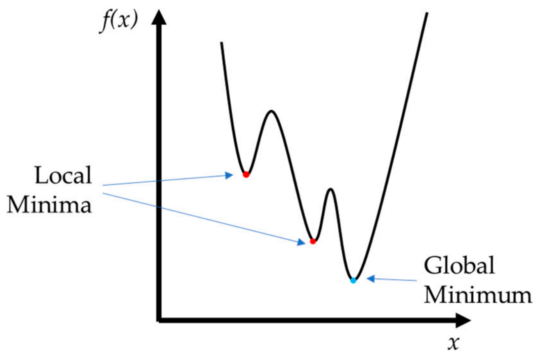

Figure 1. Local and global minima representation for a generic optimization problem.

It follows that optimization problems and algorithms can be classified into two types: local optimization and global localization algorithms. The former type of algorithm looks for local optimum solutions in a particular area of the search space, whereas the latter looks for the best solution for problems with multiple local optimums. Furthermore, there could be multiple local optimal points for an objective function, and if there is only one, that local optimum point is referred also to as the global optimum point. In case a global optimum does not exist for an objective function, the problem cannot be classified as an optimization problem.

In general, optimization problems are typically classified as multimodal or unimodal in terms of their modality. When a problem has more local optima solutions in addition to the global optimum, it is said to be multimodal; when it has only the global optimum as its local optima solution, it is said to be unimodal. Multimodal problems can sometimes be trickier and therefore more complex to solve [17].

The authors in reference [18] classify optimization problems as reported in Figure 2. It is highlighted that non-convex functions do not need to be multimodal because a non-convex function can have only one local optimum, and it follows that this is also the global optimum.

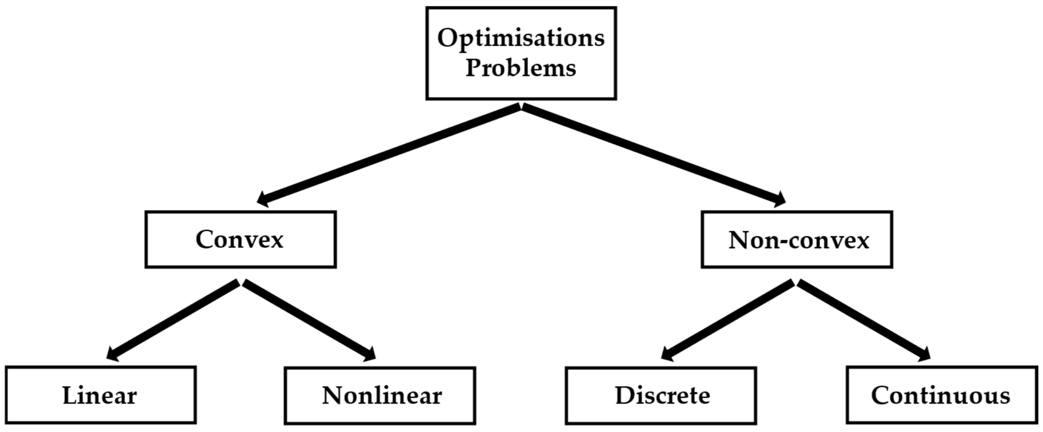

Figure 2. The classification of the optimization problems.

Local and global optimization algorithms deal with different problems; a local optimization technique is utilized when the objective function includes a single optimum or when the region of the global optimum is known; when the objective function response surface's structure is unknown or the function has local optimum points, a global optimization procedure is employed [19].

In order to discover the extrema of the function in a certain region of the search space or to come as close to them as possible, a local optimization method is developed to cross the considered region looking for the minima (or maxima) [14].

Global optimization techniques are suitable for situations where there are few variables and no necessity for fast computation time. They work with a single candidate solution or a population of them, from which new candidate solutions are generated iteratively and assessed using a predetermined function to see whether the newly generated variables are getting better or worse [14].

2.3. Local Optimization Algorithms

Most local optimization algorithms exploit the gradient information of the function to find the optimum solution. They work well for solving problems with lots of design variables, and fine-tuning their settings is rather simple. Furthermore, they are considered efficient compared to the number of function evaluations (or iterations) needed to find the solution. Among the drawbacks are that they can only locate a local optimum, they have issues solving discrete optimization problems, they require algorithms that are difficult to implement efficiently, and they suffer from numerical noise [15].

There are many types of gradient-based algorithms, and each of them is characterized by a different logic or idea to determine the search direction. In general, gradient-based algorithms typically use a two-step iterative process to reach the optimum: the first one is to exploit the gradient information to find a search direction S in which to move, and the second step is moving in the direction d; the last one is called one-dimensional or a line search and also provides the optimum step size, α_k. The two steps are repeated until no more progress is obtained and mathematically, they can be expressed as follows:

$$x_k = x_{k-1} + \alpha_k d \tag{4}$$

where k identifies the number of the k-th-step [20]. The gradient information is not always easily accessible in complex problems, and techniques such as Finite Difference Gradient Calculations, Automatic Differentiation, or Analytic/Semi-Analytic Gradient Calculations can be used to compute the gradient. Moreover, the first one provides gradient information that is accurate at the working precision; the second one provides only an approximation of the gradient, giving an accuracy that is related to the selected step size; the last one can be applied to linear finite element codes, and it is inexpensive [15].

2.4. Global Optimization Algorithms

Many problems can have multiple local optima (like the problem represented in Figure 1), and local globalization algorithms are able to find one of them without guaranteeing or saying if it is a local or global minimum point because all minima satisfy the Karush–Kuhn–Tucker (KKT) condition, which is a mathematical tool to verify if a solution is a minimum or not, but without specifying if it is a global one.

The found minima depend on the starting point, giving the first met minima as a result. To find other minima in the design space, a multi-start approach can be adopted, which starts from different points [21].

Global optimization algorithms have a far higher possibility of locating the global or near-global optimum. It is preferable to speak of these algorithms as having global properties because, in general, no algorithm can ensure convergence on a global optimum.

They fall into one of two categories: metaheuristic algorithms and deterministic algorithms [22].

2.4.1. Metaheuristic Algorithms

Metaheuristic (or stochastic) algorithms can be defined as computational intelligence paradigms which are especially used for solving complex optimization problems [23]. When precise optimization methods are unable to provide outcomes, they can provide satisfactory results. Metaheuristic algorithms can be classified into evolution-based, swarm intelligence-based, physics-based, and human behavior-related algorithms [24].

This class of optimization techniques is suitable for complex non-linear and discontinuous problems where classical optimization techniques might fail [25].

These algorithms, in contrast to the local methods, use a certain number of design points, which are often referred to as a population of individuals; they allow for determining the optimum without adopting any gradient information. These techniques frequently draw inspiration from various natural events, and some of their advantages include a higher likelihood of locating a global or nearly global optimum, robustness, and simplicity in implementation; moreover, they are also suitable for solving discrete optimization problems. On the other hand, some disadvantages include high computing costs, the limited size of the problem, weak constraint-handling capabilities, and difficulties in tuning problem-specific parameters [15].

These days, PSO and the GA are the two most often used metaheuristic algorithms. Additionally, evolutionary algorithms include genetic programming, Ant Colony Optimization (ACO), Simulated Annealing (SA), Differential Evolution (DE), Evolutionary Programming, Harmony Search, and others [22].

2.4.2. Deterministic Algorithms

Deterministic algorithms aim to find the global solution of an optimization problem by guaranteeing that the found solution is the real global minimum (or maximum) of the problem with a predefined tolerance. These algorithms usually deal with specific classes of problems, such as linear programming (LP), mixed-integer linear programming (MILP), non-linear programming (NLP), and mixed-integer non-linear programming (MINLP). An example of a deterministic technique is the DIRECT algorithm [26], which can find the global minimum of a multivariate function with simple bounds; it is a modification of the conventional Lipschitzian method where the need to specify a Lipschitz constant is eliminated.

In general, deterministic approaches can obtain advantages by exploiting the analytical properties of a problem to find the global solution; however, over the years, heuristic methods have shown to be more efficient and flexible than deterministic ones [18].

2.5. From Single to Multi-Objective Optimization

The case described by Equation (1) is referred to as a single objective optimization problem because the function to be optimized is only one. However, in many applications, the optimization of more than one objective function at the same time is requested, and this case, it is referred to as a multi-objective (MO) optimization problem. It is mathematically described as follows:

$$\begin{array}{lll} \text{minimize}: & f_m(x) & m = 1, \ldots, M \\ \text{such that}: & g_j(x) \leq 0 & j = 1, \ldots, J \\ & h_k(x) = 0 & k = 1, \ldots, K \end{array} \tag{5}$$

where m is the m-th function among the M functions to be optimized.

In MO optimization problems, two spaces are defined: the first is the decision variable space of the solution vector that contains all possible solutions x of the problem; the second is the multi-dimensional space of the objective function vector that contains the evaluations of each objective function; each x solution belonging to the decision variable space corresponds to a point in the objective function space. The spaces for a problem with a vector solution x with three components and two membership functions are represented in Figure 3 [27].

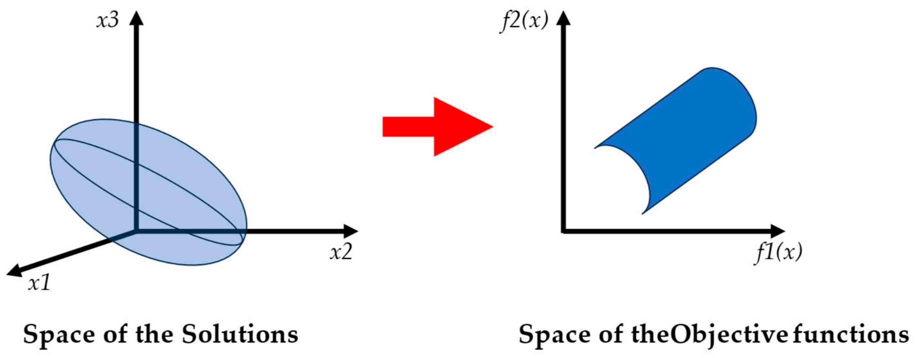

Figure 3. Representation of the two spaces for a generic MO problem.

If every objective function and solution region in a MO problem are convex, the problem is said to be convex too, and the convexity is crucial to solving the optimization problem.

The methods to find the solution to MO problems can be classified into two different types: the Pareto method and the scalarization method [28]. The former is used when performance indicators and desired solutions are distinct, and in such a way, the Pareto approach is applied to generate a compromise solution, or trade-off, that can be represented as a Pareto optimal front (POF). Unlike in the latter, a performance indicator component method is exploited to create a scalar function that is integrated into the fitness function [29].

The process of finding the optimal solution can be assisted by using metaheuristic algorithms such as GA, PSO, ACO, and so on.

2.5.1. Pareto Method

From a mathematical point of view, a MO problem with n objective functions solved by using the Pareto method is expressed as follows:

$$\begin{array}{l} f_{1,opt} = \min f_1(x) \\ f_{2,opt} = \min f_2(x) \\ \ldots \\ \ldots \\ f_{n,opt} = \min f_n(x). \end{array} \tag{6}$$

The method keeps the elements of the solution vectors independent and exploits the concept of dominance with which a solution can be considered a Pareto optimal solution (POS) or a non-dominated solution if around the considered solution there is no way of improving any objective without degrading at least one other objective; on the contrary, a solution is said to be a Pareto-dominated solution (PDS) if it can be improved without worsening at least one other objective. As an example, Figure 4 shows the solutions in the space of the objective functions where POSs are represented by red dots, whilst the PDSs are represented by red squares. Furthermore, in agreement with the representation in Figure 4, it is possible to give the following definitions [29]:

- Anchor point: the best value provided by an objective function.
- Utopia point: a point which is obtained by intersecting the best values provided by the objective functions.
- Pareto optimal front (POF): the line where the POSs lay.

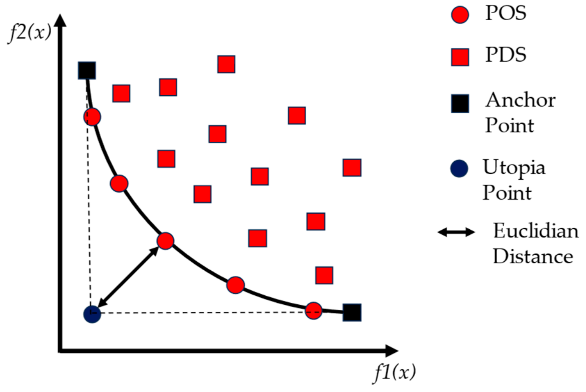

Figure 4. The space of the objective functions for a generic problem with two objective functions.

Once the POF has been obtained, the MO problem is solved by computing which of the solutions on the POF has the shortest Euclidean distance. The following equation is used:

$$d_E = \min\sqrt{\left(\frac{Q_1 - Q_1^*}{Q_{1norm}}\right)^2 + \left(\frac{Q_2 - Q_2^*}{Q_{2norm}}\right)^2} \tag{7}$$

where (Q_1^*, Q_2^*) are the coordinates for the utopia point related to f_1 and f_2, respectively, (Q_1, Q_2) are the coordinates of the solution point on the POF, and Q_{1norm} and Q_{2norm} are normalization factors based on the minimum values of each function.

In general, by considering a MO problem with two objective functions, four types of problems can be obtained and are given as follows:

$$\text{Problem 1}: \begin{cases} f_{1,opt} = \min f_1(x) \\ f_{2,opt} = \min f_2(x) \end{cases}, \tag{8}$$

$$\text{Problem 2}: \begin{cases} f_{1,opt} = \min f_1(x) \\ f_{2,opt} = \max f_2(x) \end{cases}, \tag{9}$$

$$\text{Problem 3}: \begin{cases} f_{1,opt} = \max f_1(x) \\ f_{2,opt} = \min f_2(x) \end{cases}, \tag{10}$$

$$\text{Problem 4}: \begin{cases} f_{1,opt} = \max f_1(x) \\ f_{2,opt} = \max f_2(x) \end{cases}. \tag{11}$$

Each of the four problems gives rise to a different shape of the POF and they are shown in Figure 5.

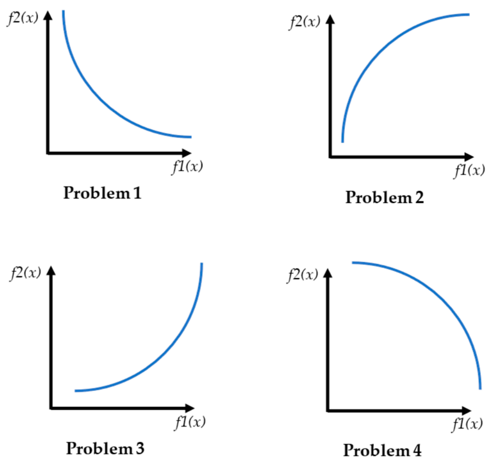

Figure 5. Representation of POF for problems with 2 objective functions.

The objective function space for a problem with three functions can be represented in a three-dimensional space, but for a problem with a number of objective functions greater than three, the space cannot be represented.

To find non-dominated solutions, specific algorithms are used such as the continuously updated method reported in [29].

2.5.2. Scalarization Method

The scalarization method includes each function of the MO problem into a single scalar fitness function, practically transforming the MO problem into a single objective problem. Different types of scalarization can be defined and some basic approaches are the Weighted Global Criterion Method (WGCM), Weighted Sum Method (WSM), Exponential Weighted Criterion (EWC), or the Weighted Product Method (WPM) [30].

From a conceptual point of view, the WSM is the easiest way to implement the scalarization method. Each function of the MO problem is summed by adopting weights; it gives rise to a new total objective function $F(x)$ which is given as follows:

$$F(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_i f_i(x) + \ldots + w_n f_n(x) \tag{12}$$

where i is the i-th function of the original MO problem, w_i represents the weight of the i-th function, and f_i is the i-th function of the original MO problem composed of n objective functions [29]. The values of the weights are chosen in agreement with the performance priority: a large weight is given to functions with higher priority and vice versa for functions with lower priority. The following are three examples of how weights can be set:

- Equal weights: $w_i = 1/n$.
- Rank Order Centroid (ROC) weights: $w_i = \frac{1}{n}\sum_{k=i}^{n} \frac{1}{k}$.
- Rank Sum (RS) weights: $w_i = \frac{2(n+1-i)}{n(n+1)}$.

To provide a balance among functions, normalization is needed, and it can be obtained by dividing each objective function by its own RMS value f_i^{RMS} [31]. The final expression is given as follows:

$$F(x) = +\frac{w_1 f_1(x)}{f_1^{RMS}} + \frac{w_2 f_2(x)}{f_2^{RMS}} + \ldots + \frac{w_n f_n(x)}{f_n^{RMS}} \tag{13}$$

In Equation (13), the sign of each objective function must agree with the request of minimization (minus) or maximization (plus).

The WSM can be considered as a particular case of the WGCM approaches, in which the scalarization can be obtained by adopting one of the following definitions [30]:

$$F(x) = \sum_{i=1}^{n} \omega_i [f_i(x)]^p \tag{14}$$

$$F(x) = \sum_{i=1}^{n} [\omega_i f_i(x)]^p \tag{15}$$

where p is a parameter which can enhance the minimization of the function, and weights must satisfy $\sum \omega_i = 1$.

The EWC method is defined as follows [32]:

$$F(x) = \sum_{i=1}^{n} (e^{p\omega_i} - 1)e^{pf_i(x)}, \tag{16}$$

whilst the WPM is given as follows [33]:

$$F(x) = \prod_{i}^{n} [f_i(x)]^{\omega_i}. \tag{17}$$

2.5.3. Summary of the Optimization Problems

In summary, from the point of view of the number of objectives, optimization problems can be divided into two types: SO and MO problems. The main difference in the formulation of the problem is that in the former, only one objective is considered, whereas in the latter, there are at least two objectives. The MO case can be considered an extension of the SO case, and allows the optimization of more functions at the same time that belong to the same problem and have the same design variables.

Finally, the MO problem can be solved by using either the Pareto method or one of the scalarization approaches. The first one deals with the optimization problem by solving each fitness function separately, whereas the second ones provide a function that is a single figure of merit that englobes all the objectives of the problem in one function. The main advantage of the Pareto approach is that each fitness function is solved independently without any regard for the other objectives, and the final solution is extracted by considering the solution that has the shortest distance from the utopia point. On the other hand, scalarization transforms the MO problem into the SO type, but has the drawback of setting the weights of each objective properly in order to correctly set their priority.

3. Optimization Algorithms Used in the Field of Localization

In this section, all optimization methods in the field of localization are reported. Generally, a description of the algorithm, its concept, and the mathematical formulation are reported; then, a real and meaningful application for localization purposes is described.

3.1. Newton's Method, Gradient Descent, and Gauss–Newton Method

Newton's Method and the Gradient Descent (GD) algorithm are the first methods which were studied in the field of optimization to deal with local problems, and they have a similar structure but with the difference that the GD algorithm is a first-order method and Newton's method is a second-order type. The latter has the disadvantage of a higher computational cost because it requires the calculation of the Hessian matrix. At the end of the paragraph, an extension of Newton's method called the Gauss–Newton method is presented.

3.1.1. Newton's Method

Newton's algorithm is a classical method and is an unconstrained algorithm that is derived from a second-order Taylor series expansion of the objective function starting from an initial design point x_0. The function can be expressed as follows [15]:

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T H(x_0)(x - x_0) \tag{18}$$

where x is the variable of the problem, and $H(x_0)$ is the Hessian matrix that contains the second-order gradient information of the objective function. By differentiating Equation (18) with respect to x and setting the result equal to zero according to the KKT conditions, it gives rise to the following iterative formula for the current design point [34]:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \tag{19}$$

where f' and f'' are the first and the second derivative, respectively, which can be expressed as $f'(x_n) = \nabla f(x_n)$ and $f''(x_n) = H(x_n)^{-1}$. The step size cannot be changed, and no one-dimensional search is required because it is provided by $-H(x_n)^{-1}\nabla f(x_n)$.

The method has a quadratic rate of convergence, requiring only a single step to obtain the optimum; however, the estimation of the second-order derivative requires a high computational cost and often the method can result in being unpractical. To overcome this drawback, some variation of gradient-based methods can be adopted, which makes use only of the first-order gradient information.

3.1.2. Gradient Descent Algorithm

The GD algorithm is a first-order optimization algorithm that finds a local minimum (or maximum) by using an iterative process. The function to be optimized must be differentiable and convex. However, it is also possible to apply the algorithm to quasi-convex functions, but with the risk of being stuck at saddle points.

The GD algorithm is based on this iterative equation [35]:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{20}$$

where f is the function to be optimized, x_k is the input variable at the k-th step, and α is a parameter which controls the step size. It is highlighted that the value of α controls the behavior of the method because, for example, the smaller the learning rate, the longer the GD convergence time, and a too-small α could make the algorithm reach the maximum number of iterations too early without finding the right solution; on the other hand, a too-large α could make the process diverge or just jump around to the minimum (or maximum) of the problem. The value $\alpha\nabla f(x_k)$ in (20) is subtracted when the minimum is looked for, whereas it is summed when the maximum is the desired output.

The GD algorithm deals with unconstrained problems, and it can be extended to also handle constraints by adopting a specific approach called the penalty function method, which is able to transform an unconstrained problem into a constrained one [20]. The method adds a penalty function to the original objective function that is composed of a penalty parameter multiplied by a measure of violation of the constraints. The latter is zero when there is no violation and greater than zero when violations are obtained. Considering a generic problem min $= f(x)$ subjected to $c_i(x) \leq 0$, it is transformed into a series of unconstrained minimization problems, given as follows [36]:

$$\min \Phi_k(x) = f(x) + \sigma_k \sum_i g(c_i(x)) \tag{21}$$

where $g(c_i(x)) = \max\,(0, c_i(x))^2$ is the exterior penalty function, σ_k is the penalty coefficient, and k represents the iteration. In each step of the algorithm, the penalty coefficient is increased, the unconstrained problem is solved, and the provided result is used in the next iteration as the initial guess. In this manner, the solution of the original constrained problem is obtained by the convergence of the unconstrained one.

In general, this method has been used in many different scientific fields, and, in particular, one of them where it has been widely adopted is Machine Learning (ML), in which the GD algorithm has three variations: the Batch GD, Stochastic GD, and Mini-Batch GD algorithms. These three different approaches manage the training process of a neural network in different ways: in the first one, the algorithm calculates the gradient to correct the weights by considering the complete training dataset in each iteration, in the second one, the gradient is calculated by considering only one sample of the dataset in each iteration, and in the last case, which is an intermediate approach between the two aforementioned approaches, the dataset is split into small subsets (batches) and the gradients are computed for each sub-batch in each iteration [37].

3.1.3. Gauss–Newton Method

The Gauss–Newton method is adopted to solve convex non-linear least-squares problems, i.e., a class of problems where it requested to minimize a sum of squared function values. Given an objective function which is twice differentiable, it has a form given as follows:

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2 \tag{22}$$

and its gradient is defined as follows:

$$\nabla f(x) = \sum_{i=1}^{m} f_i(x)\nabla f_i(x). \tag{23}$$

The search direction of the Gauss–Newton method is as follows [14]:

$$\Delta x_{gn} = -\left(\sum_{i=1}^{m} \nabla f_i(x)\nabla f_i(x)^T\right)^{-1}\left(\sum_{i=1}^{m} f_i(x)\nabla f_i(x)\right). \tag{24}$$

where it is assumed that the inverse exists. The difference with the classic Newton's method is that in this case, the direction is obtained by neglecting the second term from the Hessian. This result can be obtained by considering the first order of Taylor's series $f_i(x+v) = f_i(x) + \nabla f_i(x)^T v$, which gives rise to the following:

$$f(x+v) = \frac{1}{2}\sum_{i=1}^{m}\left(f_i(x) + \nabla f_i(x)^T v\right)^2 \tag{25}$$

where $v = \Delta x_{gn}$ is the needed exact value to minimize this approximation; furthermore, it is highlighted that the Gauss–Newton step direction can be estimated by solving a linear least-squares problem.

3.1.4. The Integer Least-Squares Problem

A particular case among least-squares problems is when the design variables are integers and these types of problems are named integer-least-squares (ILS) problems; they arise, for example, to solve the integer frequency ambiguity in Global Navigation Satellite Systems (GNSSs) for real-time applications, or in other fields such as communications, cryptography, and lattice design [38].

The pseudorange and carrier phase are extracted from GNSS measurements, and only the latter allows pose estimation with high precision in the order of the millimeter and centimeter level. However, the unknown integer part of the carrier phase, i.e., the frequency integer ambiguity, needs to be solved, and it can be performed by adopting the geometry-based model, which relies on the resolution of an ILS problem [39]. Such a problem can be defined as follows [38]:

$$\min_{a \in \mathbb{Z}^n} \left((a - \hat{a})^T Q_{\hat{a}}^{-1} (a - \hat{a}) \right) \tag{26}$$

where a is the double-differenced integer ambiguity vector, $\hat{a}$ is the integer estimation of a, and $Q_{\hat{a}}$ is the symmetric positive $n \times n$ variance–covariance matrix.

To solve the problem represented by Equation (22), specific algorithms, such as the least-squares AMBiguity Decorrelation Adjustment (LAMBDA) algorithm or its variations, have been developed to reduce the computational resolution complexity of the problem [38]. They are composed of two parts: the reduction process, and the discrete search process. In this review, they are not investigated further because these algorithms are considered numerical techniques adopted to simplify the resolution of an optimization problem.

3.1.5. Integer Frequency Ambiguities Resolution

The authors in reference [39] propose a single-epoch triple-frequency real-time kinematic positioning method that exploits the information provided by the GNSS to obtain high-precision positioning in urban environments.

In their article, they propose the Random Sample Consensus (RANSAC) that is adopted after the ILS is solved to detect and exclude incorrectly fixed ambiguities of the Extra-Wide-Lane (EWL), Wide-Lane (WL), and original frequencies. The method deals with a triple-frequency integer ambiguity, and the double-differenced pseudorange P_i and carrier phase L_i observations for the i-th frequency are given as follows:

$$\begin{aligned} P_i &= \rho + M_i + \varepsilon_i \\ L_i &= \rho + \lambda_i N_i + m_i + \epsilon_i \end{aligned} \tag{27}$$

where ρ is the double-differenced geometric range; M_i and m_i are the multipath errors in the pseudorange and carrier phase, respectively, λ_1 is the wavelength of the carrier; N_i is the integer ambiguity, and ε_i and ϵ_i are random noises of the pseudorange and carrier phase, respectively. By considering three frequencies, i.e., $i = \{1, 2, 3\}$, as presented in reference [39], it is possible to define two robust double-differenced geometric ranges, $\widetilde{\rho}_1$ and $\widetilde{\rho}_2$, which allows for the computation of the EWL and WL ambiguities that are given, respectively, as follows:

$$N_{(0,-1,1)} = \text{round}\left(\left(\frac{L_3}{\lambda_3} - \frac{\lambda_2}{\lambda_2} \right) - \left(\frac{\widetilde{\rho}_1}{\lambda_3} - \frac{\widetilde{\rho}_1}{\lambda_2} \right) \right) \tag{28}$$

$$N_{(1,-1,0)} = \text{round}\left(\left(\frac{L_1}{\lambda_1} - \frac{\lambda_2}{\lambda_2} \right) - \left(\frac{\widetilde{\rho}_2}{\lambda_1} - \frac{\widetilde{\rho}_2}{\lambda_2} \right) \right). \tag{29}$$

Equations (28) and (29) allow for the definition of the following system of equations:

$$\begin{cases} N_{(1,-1,0)} = N_1 - N_2 \\ N_{(0,-1,1)} = N_3 - N_2 \\ L_1 = \rho + \lambda_1 N_1 + m_1 + \epsilon_1 \\ L_2 = \rho + \lambda_2 N_2 + m_2 + \epsilon_2 \\ L_3 = \rho + \lambda_3 N_3 + m_3 + \epsilon_3 \end{cases} \tag{30}$$

and by adopting an ILS method, it is possible to estimate ρ, N_1, N_2, and N_3 of the problems.

3.2. Levenberg–Marquardt

The Levenberg–Marquardt algorithm (LMA), or the Damped least-squares (DLS) method, is used to solve non-linear optimization problems. The LMA exploits a parameter λ such that it can switch between the GD algorithm and the Gauss–Newton algorithm;

practically, the LMA interpolates between the Gauss–Newton algorithm and the method of GD according to the current prediction accuracy, but at the expense of a certain convergence rate [40]. The algorithm needs the initial guess and, if it is too far, then the error might be large, and the algorithm might not be able to give a correct global solution due to the presence of many local minima, and it might provide a local minimum which is not necessarily the global minimum. To obtain a better correct initial guess for the LMA, it is possible to use EAs in the initial phase because they do not suffer from the initial value problem.

Considering a non-linear least-squares problem, it can be expressed mathematically as follows:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) = \frac{1}{2}\|r(x)\|^2 \tag{31}$$

where x is the vector input, f is the objective function, r_i is the residual and it is defined as a function such that $\mathbb{R}^n \to \mathbb{R}$ with the assumption of $m \geq n$, and $r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^T$ is the residual vector and is defined such that $\mathbb{R}^m \to \mathbb{R}^n$.

The Levenberg–Marquardt algorithm is a modification of the Gauss–Newton method, and it is based on a local linearization of the residuals as follows [40–42]:

$$r_m(x + \delta x) \approx r_m(x) + J_{m\mu}\delta x \tag{32}$$

where $J_{m\mu} = J(x) = \frac{\partial r_m}{\partial x_\mu}$ is the Jacobian Matrix, and δx is an infinitesimal of x. Next, the Gauss–Newton technique recursively updates in accordance with the following:

$$\delta x = -\left(J^T J\right)^{-1} \nabla f = -\left(J^T J\right)^{-1} J^T r. \tag{33}$$

If the initial guess of the Gauss–Newton method is near the optimum of f, it will usually converge quickly; however, this is not always the case, and the method can take very large and uncontrolled steps, failing to converge. To deal with this issue and control the steps in a useful way, Levenberg [41] and Marquardt [42] each proposed the damping of the $J^T J$ matrix by a diagonal cut-off, modifying (33) as follows:

$$\delta x = -\left(J^T J + \lambda D^T D\right)^{-1} \nabla f \tag{34}$$

where $D^T D$ is a positive–definite, diagonal matrix that is the relative scale of the parameters and λ is a damping parameter adjusted by the algorithm. If λ is large, the steps are small along the gradient direction, and vice versa when λ is small. As the method is reaching a solution, λ is chosen to be small and the method converges quickly via the Gauss–Newton method.

The implementation of the Levenberg–Marquardt algorithm follows these iterative steps [40]:

1. Based on the current parameter values, update the function and Jacobian values (if necessary);
2. Update the scaling matrix $D^T D$ and damping parameter λ;
3. Propose a parameter step δx and evaluate the function at the new parameter values $x + \delta x$;
4. Accept or reject the parameter step depending on whether the cost has decreased at the new parameters;
5. Stop if any of the desired convergence criteria of the algorithm are met or if the limit of function evaluations or iterations has been exceeded.

The methods for the choice strategy of the damping parameter λ and the scaling matrix $D^T D$ require specific methods (as described in [40]) which are not further investigated in this article.

Magnetic Localization Technique Based on the Levenberg–Marquardt Algorithm

The authors in reference [10] propose a method for the localization of a permanent magnet based on the LMA. The permanent magnet is modelled as a dipole, and the magnetic field is measured by an array of 4×4 magnetometers (16 in total). The general optimization problem can be expressed as follows:

$$f(a, b, c, \theta, \varphi) = \sum_{i=1}^{N} \left|\hat{B}_i - B_i\right|^2 \tag{35}$$

where f is the objective function, a, b, and c are the coordinates of the permanent magnet position, θ and φ identify the orientation of the permanent magnet, N is the number of the i-th magnetometer, $\hat{B}_i$ is the measured magnetic field, and B_i is a function representing the model of the system based on the magnetic dipole theory, and it includes the coordinates and the orientation angles.

The problem can be solved by using the LMA, but the issue is the need for a good initial guess to obtain a solution because when a too-far initial point is provided, divergence can occur. To deal with this problem, a solution could be to combine the PSO and LMA methods: the former is used to calculate an initial guess, and the second is used to calculate the local minima because it results in a faster method. The drawback of this approach is that it is not suitable for real-time applications, and the authors of paper [10] deal with it by proposing to use only the LMA and a properly modified model of Equation (35). In the first step, the new model neglects the orientation to find only the coordinates of the permanent magnet; then, in the second step, the orientation is also obtained. The modified model is expressed as follows:

$$f(a, b, c) = \sum_{i=1}^{N} \left|\hat{B}_i - \frac{B_T}{|R_i|^3} A_i \hat{m}_{avg}\right|^2 \tag{36}$$

where B_T is a scalar quantity that represents the magnetic strength, R_i is the vector from the center of the permanent magnet to the i-th sensor, A_i is a matrix that is a function of the magnet position, and $\hat{m}_{avg}$ is the averaged orientation vector, expressed as follows:

$$\hat{m}_{avg} = \frac{\sum_{i=1}^{N} \frac{\tilde{m}_i}{\left|\tilde{m}_i\right|}}{\left|\sum_{i=1}^{N} \frac{\tilde{m}_i}{\left|\tilde{m}_i\right|}\right|} \tag{37}$$

where $\tilde{m}_i = \frac{|R_i|^3}{2B_T}(A_i - I)\hat{B}_i$, and I is the one-diagonal matrix. The orientation vector is obtained by applying the following equation:

$$\hat{m} = \frac{|R_i|^3}{2B_T}(A_i - I)B_i \tag{38}$$

The authors claim that in such a way, a more suitable optimization problem for the LMA is obtained, and for this application, the method always provides an acceptable solution independently from the distance of the initial guess; moreover, the speed of computing is improved.

3.3. Lagrange Multipliers Method and Lagrange Duality

3.3.1. Lagrange Multipliers Method

The Lagrange multipliers method is an analytic method for finding the local maxima and minima of a function subject to equality constraints. The concept is to reformulate the original problem, obtaining a new function called the Lagrangian function, which allows for solving a new problem without constraints. Given a problem with an objective function

$f(x)$ subjected to the equality constraint $h(x) = 0$, the Lagrangian function is expressed as follows [43]:

$$L(x, \lambda) \equiv f(x) + \lambda h(x) \tag{39}$$

where λ is the Lagrangian multiplier. The solution is obtained by finding the stationary point of L by setting $\frac{\partial L(x,\lambda)}{\partial x} = 0$ and $\frac{\partial L(x,\lambda)}{\partial \lambda} = 0$, which gives rise to the following system:

$$\begin{cases} \frac{\partial f(x)}{\partial x} + \lambda \frac{\partial h(x)}{\partial x} = 0 \\ h(x) = 0 \end{cases} \tag{40}$$

The solution corresponding to the original problem is always a saddle point of the Lagrangian function [44]. Then, a possible solution of the system (40) (x^*, λ^*) allows for the finding of an optimal point of the original problem given by $f(x^*)$.

The method of Lagrange multipliers is generalized by the KKT condition, also allowing for the consideration of inequality constraints; given a minimization non-linear problem (in which the x variable is a vector) with an objective function $f(x)$ subject to equality constraints $h_i(x) = 0$ (with $i = 1, \ldots, p$), and inequality constraints $g_j(x) \leq 0$ (with $j = 1, \ldots, m$), the Lagrangian function is set as follows:

$$L(x, \lambda, \mu) \equiv f(x) + \mu^T g(x) + \lambda^T h(x) \tag{41}$$

where μ and λ are the Lagrange multipliers related to inequalities and equalities, respectively, and they are vectors. If (x^*, λ^*, μ^*) is a saddle point of $L(x, \lambda, \mu)$ in the x domain, and the KKT conditions are satisfied, then x^* identifies an optimal solution. The KKT conditions can be summarized as follows [45]:

$$\begin{aligned} & g_j(x^*) \leq 0 \\ & h_i(x^*) = 0 \\ & \mu_j^* \geq 0 \\ & \mu_j^* g_j(x^*) = 0 \\ & \nabla f(x^*) + \sum_j \mu_j^* \nabla g_j(x^*) + \sum_i \lambda_i^* \nabla h_i(x^*) = 0 \end{aligned} \tag{42}$$

In general, once the gradient has been obtained, to determine if a constrained local optimum has been found, the KKT conditions are used to verify the necessary conditions for a local optimum; for unconstrained problems, the KKT conditions only require the gradient of the objective function to vanish at the optimum design point.

It is highlighted that the KKT conditions cannot indicate whether a global optimum has been found, and they are useful only for identifying a local optimum.

3.3.2. Lagrange Duality

The duality principle claims that an initial optimization problem called the primal problem can be transformed into a dual problem by applying a specific procedure. In such a way, it allows for the transformation of a minimization problem into a maximization one and vice versa. In some cases, the method might make the solution of the transformed problem easier and more manageable than the primal problem [14].

Mathematically, the Lagrange duality is defined as follows [46]. A not necessarily convex optimization problem is considered and is represented as:

$$\begin{aligned} \text{minimize}: \quad & f_0(x) \\ \text{subject to}: \quad & f_i(x) \leq 0 \quad i = 1, \ldots, m \\ & h_i(x) = 0 \quad i = 1, \ldots, p \end{aligned} \tag{43}$$

where $x \in \mathbb{R}^n$, with a domain D and optimal value p^*. Then, by using (41), the Lagrangian function of the problem is obtained and given as:

$$L(x, \lambda, \nu) \equiv f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{44}$$

where $L(x, \lambda, \nu)$ is such that $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ with a domain $L = D \times \mathbb{R}^m \times \mathbb{R}^p$, and λ_i and ν_i are the Lagrange multipliers related to inequalities and equalities, respectively. Now, it is possible to define the Lagrangian dual function $g(\lambda, \nu)$ of the problem as the infimum of the Lagrangian function over x and it is given as:

$$g(\lambda, \nu) = \inf_{x \epsilon D} L(x, \lambda, \nu) = \inf_{x \epsilon D} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \tag{45}$$

where $g(\lambda, \nu)$ is such that $\mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$. It is observed that the infimum is unconstrained (unlike the beginning problem) and $g(\lambda, \nu)$ is concave; moreover, the function g could be $-\infty$ for some λ, ν.

At this point, by adopting the lower bound property, if $\lambda \geq 0$, then $g(\lambda, \nu) \leq p^*$; this allows us to define the dual problem by maximizing the obtained Lagrangian dual function in (45). It is given as follows:

$$\begin{aligned} &\underset{\lambda, \nu}{\text{maximize}}\ g(\lambda, \nu) \\ &\text{subject to } \lambda \geq 0 \end{aligned} \tag{46}$$

The solution of (46), denoted with λ^*, ν^*, provides for all convex and non-convex problems a lower bound for the optimum function such that $g\left(\lambda^*, \nu^*\right) \leq f(x^*)$, like it is represented in Figure 6.

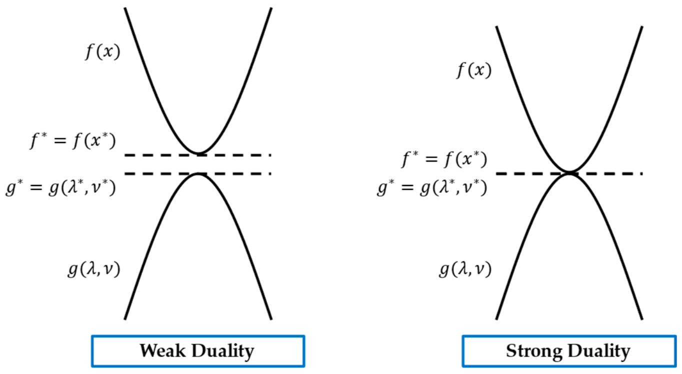

Figure 6. Representation of weak and strong duality.

Furthermore, Figure 6 clearly represents the weak and strong duality principles. The former provides a bound for the optimal values of the transformed problem, whereas the latter claims that the optimal values of the two problems are equal. The strong duality is usually true for convex optimization problems.

3.3.3. A Localization Non-Convex Problem

The authors in reference [47] propose a method to obtain localization by exploiting only the range measurement data provided by wireless sensors. The robot localization is modelled as a non-convex optimization problem, and to obtain the solution, as a first step, it is reformulated as a convex problem, and then duality theory is applied to find, in an easier way, the optimum.

The localization problem is formulated as follows. The measured range data from anchors are given as follows:

$$\hat{d}_k = \|x - y_k\|_2 + \eta_k \tag{47}$$

where x and y_k are two column vectors to determine the positions of the target and the anchors, respectively, η_k is the corresponding measurement noise, $k = 1, \ldots, m$ identifies the k-th anchor, which are m in total. The optimal position can be obtained by minimizing the measurement noise between the measured distance $\hat{d}_k$ and the real distance $d_k = \|x - y_k\|_2$; then, the objective function can be expressed as follows:

$$f_0(x) = \sum_{k=1}^{m} \left(\hat{d}_k - \|x - y_k\|_2\right)^2 \tag{48}$$

The function in (48) is non-convex, and to solve the problem, the authors proceeded in this way. In the first step, they reformulated the problem into a convex-constrained problem by exploiting a change in the variable; the new problem is expressed as follows:

$$\begin{aligned} &\min f(z) = \|Az - b\|_2^2 \\ &\text{subject to } (z^T C z + 2f^T z) = 0 \end{aligned} \tag{49}$$

where $z = \begin{bmatrix} x^T & x^T x \end{bmatrix} = \begin{bmatrix} x^T & t \end{bmatrix}^T$ and coefficients are defined as $A = \begin{bmatrix} -2y_1^T & 1 \\ -2y_2^T & 1 \\ \vdots & \vdots \\ -2y_k^T & 1 \end{bmatrix}, b = \begin{bmatrix} d_1^2 - \|y_1\|_2^2 \\ d_2^2 - \|y_2\|_2^2 \\ \vdots \\ d_k^2 - \|y_k\|_2^2 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, and $f = \begin{bmatrix} 0 \\ 0 \\ -1/2 \end{bmatrix}$.
Then, duality theory is adopted, and the obtained Lagrangian function is given as follows:

$$L(z, v) = \|Az - b\|_2^2 + v\left(z^T C z + 2f^T z\right) = z^T\left(A^T A + vC\right)z - 2\left(A^T b - vf\right)^T z + b^T b \tag{50}$$

where v is the Lagrange multiplier. Now, by computing the minima of $L(z, v)$ over z, it is possible to obtain the Lagrangian dual function, and the final Lagrange dual problem is given as follows:

$$\begin{aligned} \text{maximize} \quad & G(v) = -\left(A^T b - vf\right)^T \left(A^T A + vC\right)\left(A^T b - vf\right) + \|b\|_2^2 \\ \text{subject to} \quad & (A^T A + vC) \geq 0 \\ & (A^T b - vf) \epsilon R\left(A^T A + vC\right) \end{aligned} \tag{51}$$

where R identifies the domain. The problem in (51) is a convex problem and the solution can be computed.

3.4. Genetic Algorithm

The GA was inspired by Darwin's principle of survival of the fittest solutions and can solve both constrained and unconstrained optimization problems; it is based on natural selection, the process that drives biological evolution [48]. The algorithm is implemented with the following steps:

1. Creation of the initial population.
2. Calculation of fitness of each individual.
3. New individual generation by adopting the following operators:
    - 3.1. Selection;
    - 3.2. Crossover;
    - 3.3. Mutation;
4. Calculation of fitness of each individual.
5. Test:
    - 5.1. If the convergence criteria are satisfied, stop.
    - 5.2. Otherwise, repeat from step 3.

It is iterative, and after the first cycle, it is repeated from step 3 until the desired criteria are satisfied. In step 1, the process begins with a set of individuals, which is called the population; each individual is a solution to the problem to be solved. An individual is characterized by a set of parameters (variables) known as genes, which are joined into a string to form a chromosome, i.e., a solution. In the second step, the fitness function of the problem determines how an individual (or solution) fits, and it measures the ability of an individual to compete with other individuals. The output is a fitness score for each individual, and the probability that an individual will be selected for reproduction is based on its score. In step 3, the new individual generation process begins, and it is composed of four operators, i.e., (i) selection, (ii) crossover, and (iii) mutation. In the selection, some individuals are chosen to generate new ones and selected individuals pass some of their genes into the next generation; in other words, some individuals are selected as parents to generate children. In general, individuals are selected based on their fitness scores, and the higher the fitness value, the higher the probability of being selected for reproduction. In the literature, there are many proposals for selection operators, and some of them are the following: roulette wheel selection, sequential selection, tournament selection, dominant selection, hybrid selection, kin selection, back controlled selection, and so on [49]. It is important to highlight that, at every cycle, an elitist population is extracted from the population of the current iteration to directly pass some of the best individuals to the next generation. In some cases, such a procedure might give rise to problems of fast convergence, and to tackle such an issue, an adaptive elitist population strategy which changes the number of the elitist population can be implemented, as proposed in reference [50]. After selection, the crossover operator creates new children by mixing the relative genes. For example, Figure 7 shows the concept of the 1-crossover point operator, where two parents generate two offspring by exchanging portions of their genes. In the literature, there are many proposals for crossover operators and the authors in reference [51] classify them in three categories: standard (such as 1-crossover point, k-crossover point, shuffle-crossover, reduced surrogate-crossover, and so on), binary (random respectful crossover, masked crossover, 1-bit adaption crossover, and so on), and application dependent (or real/tree).

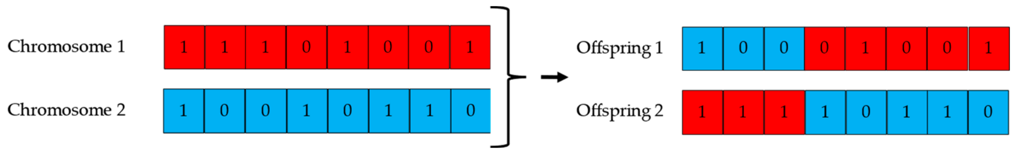

Figure 7. Representation of the standard 1-crossover point operator concept.

Finally, the mutation operators can introduce random variation of the genes, and for this purpose, the mutation probability value is adopted, which is a random value between 0 and 100%; a value equal to zero means no-mutations are adopted, whereas a 100% value modifies all genes of the child. In general, mutations should not occur so often, but they are important because they allow for the avoidance of being stuck in local minima [52]. For example, some mutation operators in the literature are mirror mutation and binary bit-flipping mutation, mutation based on directed variation techniques, directed mutation, and so on [53].

Once a new population is obtained, step 4 computes the fitness values of the offspring and, finally, step 5 verifies whether the convergence criteria are satisfied or not. If not, the cycle is repeated from step 3 until a satisfactory solution is found or the maximum number of iterations is reached; over successive generations, the population evolves toward an optimal solution [15,48,54].

The GA can solve a variety of optimization problems that are not well-suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly non-linear. Furthermore, GAs are also suitable for problems of mixed-integer programming where solutions have some components that are defined as integers.

Evolutionary Optimization Strategy for Indoor Position Estimation Using Smartphones

The authors in reference [9] propose a smartphone-based indoor localization system adopting a sensor fusion algorithm where localization information from an Inertial Measurement Unit (IMU), Wireless Local Area Network (WLAN), Bluetooth, and magnetic field anomalies is fused to provide the estimated localization.

In the fusion step, the probability distributions of each localization information source are superimposed; in the summation, each distribution is weighted with a coefficient in the interval from 0 to 1.

For determining the weights, optimization methods can be applied; they exploit the result of some tests that were run in buildings to obtain the ground truth points (GTPs). In such a way, it is possible to estimate the weights with a Root-Mean-Square Error (RMSE)-based fitness metric, defined as follows:

$$\text{Fitness} = \prod_{\text{test run}=1}^{N} \frac{10}{\text{RMSE}_{\text{test run}}} \tag{52}$$

Due to the high ambiguity and the unknown (mixed) stochastic distributions of the input variables, using classical search methods (e.g., gradient-based methods), the optimization quickly converges to a local minimum. Then, a global optimization procedure is needed to maximize the quality metric and avoid local extrema. Because of the building model's non-linearities, no gradients can be easily derived analytically, and gradient-based optimization methods are not applicable. Therefore, the authors proposed the use of a GA for finding the optimal weights, and in their article, they compared the results with four other local and global optimization strategies: Hill Climbing, the Nelder–Mead method, SA, and PSO. The outcome is that the GA outperforms the other algorithms; however, the drawback is that it needs the most computing time, but it is not a particular issue because the optimization process of the parameters must only be calculated once the end of the offline phase is reached.

3.5. Quantum Genetic Algorithm

The Quantum Genetic Algorithm (QGA) exploits the concept of qubits and the superposition of states in quantum mechanics. The algorithm is explained as follows [11,55]. The information is represented by a quantum bit or qubit which may be in the '0' state, in the '1' state, or any superposition of the two states [55]. A representation of the qubit state is given as follows:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{53}$$

where α and β are complex numbers that specify the probability amplitudes of the corresponding states, and $|\alpha|^2$ and $|\beta|^2$ are the probabilities of the states to be either 0 or 1, respectively, and the condition $|\alpha|^2 + |\beta|^2 = 1$ must be satisfied.

A system with m qubits represents 2^m states at the same time, which collapse into a single state during the observing process of a quantum state.

In general, the traditional GA can use binary, numeric, or symbolic representation for a solution encoded with a chromosome. The QGA uses the qubits, where one qubit is represented as $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$, whilst m qubits are represented as follows:

$$\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{array}\right] \tag{54}$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1$ for $i = 1, 2, \ldots, m$. For example, considering m = 3 and the following chosen amplitudes of probability, a superimposition of states is represented as follows:

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & 1 & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & \frac{\sqrt{3}}{2} \end{bmatrix}, \tag{55}$$

and the system state is expressed as follows:

$$\frac{1}{2\sqrt{2}}|000\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|001\rangle + \frac{1}{2\sqrt{2}}|100\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|101\rangle. \tag{56}$$

It has four pieces of information at the same time, i.e., the probabilities of $|000\rangle$, $|001\rangle$, $|100\rangle$, and $|101\rangle$ are $\frac{1}{8}$, $\frac{3}{8}$, $\frac{1}{8}$, and $\frac{3}{8}$, respectively. For a better comprehension of how coefficients are obtained, the coefficient $\frac{1}{2\sqrt{2}}$ of the binary number $|000\rangle$ is obtained by multiplying each α_i in the matrix (55), i.e., $\alpha_1\alpha_2\alpha_3 = \frac{1}{\sqrt{2}}*1*\frac{1}{2}$, whereas the coefficient $\frac{\sqrt{3}}{2\sqrt{2}}$ of the number $|101\rangle$ is obtained by $\beta_1\alpha_2\beta_3 = \frac{1}{\sqrt{2}}*1*\frac{\sqrt{3}}{2}$. The square power of a coefficient gives the probability of the related binary number. Moreover, the second digit is always zero, because $\alpha_2 = 1$.

Because QGA computing can represent a superposition of states, it provides a better characteristic of diversity than classical approaches; Equation (56) is one qubit chromosome that represents four states at the same time, whereas, in a classical representation, at least four chromosomes (000, 001, 100, and 101) would be needed.

During the convergence process, the $|\alpha|^2$ or $|\beta|^2$ of the qubit chromosomes can approach 1 or 0, giving rise to a single state.

In the QGA, a population of qubit chromosomes $Q(t) = \{q_1^t, q_2^t, \ldots, q_n^t\}$ at generation t is defined, where n is the size of the population, and q_j^t is a qubit chromosome with m qubits defined as follows:

$$q_j^t = \left[\begin{array}{c|c|c|c} \alpha_1^t & \alpha_2^t & \cdots & \alpha_m^t \\ \beta_1^t & \beta_2^t & \cdots & \beta_m^t \end{array}\right]. \tag{57}$$

Considering the main step to implement the QGA, in the initialization step of $Q(t)$, and all α_i^t and β_i^t of each q_j^t are set to $1/\sqrt{2}$ and it means that the qubit chromosome at $t = 0$ represents the linear superposition of all possible states with the same probability $\left(\frac{1}{\sqrt{2^m}}\right)$, and it can be expressed as follows:

$$\left|\Psi_{q_j^0}\right\rangle = \sum_{k=1}^{2^m} \frac{1}{\sqrt{2^m}}|S_k\rangle \tag{58}$$

where S_k is the k-th state represented by the binary string $x_1x_2\ldots x_m$, where x_i can be equal to 0 or 1.

The next step is to generate a set of solutions from $Q(t)$, and this is called the observing process, which provides $P(t) = \{p_1^t, p_2^t, \ldots, p_n^t\}$ at the time t.

One binary solution, p_j^t, for j from 1 to n, is a binary string of length m, and is formed by selecting each bit using the probability of qubit $|\alpha_i|^2$ (or $|\beta_i|^2$), which is compared with an r_i value that is a randomly generated number between 0 and 1; if $r_i \geq |\alpha_i|^2$, then the i-th element x_i of the j-th binary string p_j^t is 1, or, if $r_i < |\alpha_i|^2$, it is 0. Each solution p_j^t is evaluated to give a measure of its fitness. The initial best solution is then selected and stored among the solutions $P(t)$.

After initialization, the iterative part of the algorithm begins, and a set of binary solutions $P(t)$ is formed by observing the $Q(t-1)$ population; each binary solution in $P(t)$ is evaluated to give the fitness value. In the next step, called "update $Q(t)$", a set of qubit chromosomes q_j^t is updated by applying an appropriate quantum gate $U(t)$. A quantum gate is a reversible gate represented by a unitary operator U acting on the qubit basis states, and it satisfies the relation $U^+U = UU^+$, where U^+ is the Hermitian adjoint of U. The quantum gate is selected or designed in agreement with the considered practical problem to be solved; for example, the rotation gates can be used for knapsack problems, and it is given as follows:

$$U(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{59}$$

where θ is the rotation angle. For instance, a qubit amplitude probability $\begin{bmatrix} \alpha_i & \beta_i \end{bmatrix}^T$ in a chromosome can be updated by a quantum rotation that is given as follows:

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = U(\theta_i) \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \tag{60}$$

where $\theta_i = s(\alpha_i, \beta_i)\Delta\theta_i$, and $s(\alpha_i, \beta_i)$ is a function to determine the sign of θ_i, and $\Delta\theta_i$ is the variation of θ_{i-1}. The functions $s(\alpha_i, \beta_i)$ and $\Delta\theta_i$ are chosen in agreement with the lookup table (as indicated in reference [55]), which is based on experience and experiments; $\Delta\theta_i$ is a key parameter because it influences the computation quality and efficiency of the method; if it is too small, the number of iterations might be too big, whereas when it is too large, the solution might be trapped into divergent locations or give rise to a premature convergence to a local optimum [11].

Thanks to this step, the qubit chromosomes move and can converge to a better solution. In the next step, among the $P(t)$ solutions, the best one is selected and compared with the previous best one; if the new solution is better than the previous one, the older solution is discarded; otherwise, the old one is retained. By the application of some genetic operators, it is also possible to obtain mutations and crossovers of chromosomes; the mutation gives rise to new individuals characterized by a small change, whereas the crossover allows the generation of new individuals by combining parts of two or more different chromosomes. Genetic operators make the probability of the linear superposition of states change; however, mutation and crossover are not needed in the QGA because the quantum representation provides a diversity in chromosomes and the application of genetic gates with a high probability of variation can result in worse performance.

The diversity of the QGA can provide better convergence than conventional GAs, which can deal only with fixed 0 and 1 information. Moreover, the QGA has a speed of calculation $\sqrt{N}$ times faster than the speed of the traditional genetic method. The main steps of the QGA are summarised as follows:

1. Initialize of $Q(t = 0)$.
2. t = t + 1.
3. Observe $Q(t - 1)$ and create $P(t)$.
4. Evaluation of $P(t)$.
5. Update $Q(t)$ using quantum gates $U(t)$
6. Store the best solution among $P(t)$.
7. If max generation is not reached or convergence criteria are not satisfied, repeat from step 2.

An Adaptive Quantum-Inspired Evolutionary Algorithm for Localization of Vehicles

The authors in reference [11] introduce a method based on an Adaptive Quantum-Inspired Evolutionary Algorithm (AQIEA) to estimate the calibration parameters for a real-size intelligent vehicle that can be driven along a non-predefined road path. They use the Differential Global Positioning System (DGPS) as a ground truth and propose an optimization-based calibration method to reduce the systematic errors of wheeled odometers (the differences between the actual wheel diameters and the nominal diameters) and the gyroscope (initial installation error due to misalignment of the sensor with the longitudinal axis of the vehicle). The diameter of the left and right rear wheels and the heading of the gyroscope are considered as the calibration parameters to be optimized to reduce the localization errors in the dead reckoning.

The dead reckoning model can be expressed as follows:

$$\begin{aligned} x_{t+\Delta t} &= x_t + \Delta d \cdot \cos\ (\vartheta_t + \Delta\vartheta/2) \\ y_{t+\Delta t} &= y_t + \Delta d \cdot \sin\ (\vartheta_t + \Delta\vartheta/2) \\ \vartheta_{t+\Delta t} &= \vartheta_t + \Delta\vartheta \end{aligned} \tag{61}$$

where x, y, and ϑ are, respectively, the coordinates of the vehicle along the x and y-direction in the Cartesian coordinate system and the heading of the vehicle at time t; Δt is the sampling period; Δd is the incremental travel displacement of the vehicle, and it can be expressed as $\Delta d = \frac{\Delta d_r + \Delta d_l}{2}$, where $\Delta d_{r(l)}$ is the displacement of the right (left) rear wheel of the vehicle, and $\Delta\vartheta$ is the heading change during the sampling period.

The wheel displacement can be expressed as follows:

$$\Delta d_r = \frac{\Delta N_{r(l)}}{R_r(l)} \pi D_{r(l)} \tag{62}$$

where $\Delta N_{r(l)}$ is the increment of the encoder pulses of the right (left) during the sampling period, $R_{r(l)}$ is the encoder resolution of the right (left) wheel in the unit of pulses per revolution, and $D_{r(l)}$ is the nominal diameter of the right (left) wheel.

In the real application, this model is influenced by systematic errors and random errors; for example, considering the odometers, the former are due to differences between wheel diameters, different nominal diameters, and the misalignment of wheels; the latter is due to the limited resolution of encoders, the limited sampling rate, the condition of the road (rough surface), the wheel slippage, and so on.

The systematic errors can be compensated and by including them in the model, the displacements and heading variation are expressed as follows:

$$\begin{aligned} \Delta\widetilde{d}_r &= \Delta d_r + c_1 \\ \Delta\widetilde{d}_l &= \Delta d_l + c_2 \\ \widetilde{\vartheta}_t &= \vartheta_t + c_3 \end{aligned} \tag{63}$$

where the symbol $\sim$ denotes the new variables that include systematic errors, and c_1, c_2, and c_3 are the parameters to compensate for the right wheel diameter, the left wheel diameter, and the misalignment of the gyroscope, respectively. By considering these errors, the model represented by Equation (61) is rewritten as follows:

$$\begin{aligned} \widetilde{x}_{t+\Delta t} &= \widetilde{x}_t + \tfrac{\Delta\widetilde{d}_r + \Delta\widetilde{d}_l}{2} \cdot \cos\ (\widetilde{\vartheta}_t + \Delta\widetilde{\vartheta}/2) \\ \widetilde{y}_{t+\Delta t} &= \widetilde{y}_t + \tfrac{\Delta\widetilde{d}_r + \Delta\widetilde{d}_l}{2} \cdot \sin\ (\widetilde{\vartheta}_t + \Delta\widetilde{\vartheta}/2) \\ \widetilde{\vartheta}_{t+\Delta t} &= \widetilde{\vartheta}_t + \Delta t. \end{aligned} \tag{64}$$

The positioning error E_i at the i-th sample is defined as the difference between the data provided by DGPS and the dead reckoning method, and it is computed by the following:

$$E_i\left(c_1, c_2, c_3, p_i^{GPS}, p_i^{DR}\right) = \sqrt{\left(x_i^{GPS} - \widetilde{x}_i\right)^2 + \left(y_i^{GPS} - \widetilde{y}_i\right)^2} \tag{65}$$

for $i = 1, 2, \ldots, N$ where N is the total number of samples.

Then, it is possible to formulate the optimization problem, which is expressed as follows:

$$\begin{aligned} \text{minimize}: &\quad M(c_1, c_2, c_3) \\ \text{subject to}: &\quad \begin{cases} c_1^L \le c_1 \le c_1^U \\ c_2^L \le c_2 \le c_2^U \\ c_3^L \le c_3 \le c_3^U \end{cases} \end{aligned} \tag{66}$$

where M is the objective function to be minimized with respect to c_1, c_2, and c_3, and the superscript L and U denote the low and upper limits, respectively, for the considered variables. Two different optimization objective functions are considered to determine the optimal calibration parameters; the first one is the maximum positioning error, which is defined as follows:

$$M = \max E_i, \tag{67}$$

and the second one is the mean positioning error, defined as follows:

$$M = \frac{1}{N}\sum_{i=1}^{N} E_i \tag{68}$$

To solve the optimization problem, the authors use the AQIEA, which is an evolution of the QGA. In the adaptive method, the lookup table is not used because it is substituted by a different approach based on the idea of choosing as input for the quantum gate either a large value for $\Delta\theta_i$ or a small one. The former is used when an individual is away from the optimal solution, and it needs to bring individuals quickly towards the optimal solution; the latter is used to avoid an individual moving away from the optimal solution when it is going to be reached. Then the magnitude of the rotation angle is obtained by the following:

$$\Delta\theta_i = \begin{cases} \theta_{max} - (\theta_{max} - \theta_{min})\frac{f - f_{ave}}{f_{max} - f_{ave}} & f > f_{ave} \\ \theta_{max} & f \leq f_{ave} \end{cases} \tag{69}$$

where θ_{max} and θ_{min} are two positive real numbers ($0.001\pi \leq \theta_{min} < \theta_{max} < 0.5\pi$), f_{max} is the maximum fitness in the current population, f_{ave} is the average fitness of the individuals in the current population, and f is the fitness of the selected individual. In such a way, an adaptive change in the rotation angle magnitude of the quantum rotation gate is obtained.

The sign of the rotation angle is determined by the following:

$$s(\alpha_i, \beta_i) = \begin{cases} \text{sgn}\left(\left(o_i^* - c\right)\alpha_i\beta_i\right) & \alpha_i\beta_i \neq 0 \\ 0 & \left(\alpha_i = 0 \text{ and } o_i^* = 1\right) \text{ or } \left(\beta_i = 0 \text{ and } o_i^* = 0\right) \\ \pm 1 & \text{otherwise} \end{cases} \tag{70}$$

where sgn is a function that provide the sign, o_i^* is the observed state (0 or 1) of the i-th qubit of the best individual, and c is a positive real number between 0 and 1. The proof of (70) is given in reference [11].

Moreover, to improve global searching and to avoid being trapped in local optimal minima, an adaptive quantum mutation operation is introduced (unlike the traditional QGA, where mutations are not used). Given a selected chromosome in the population, the mutation operation is conducted based on a mutation probability p_m:

$$p_m = \begin{cases} p_{max} - (p_{max} - p_{min})\frac{f - f_{ave}}{f_{max} - f_{ave}} & f > f_{ave} \\ p_{max} & f \leq f_{ave} \end{cases} \tag{71}$$

where p_{max} and p_{min} are two small positive real values between 0 and 1. It is observed, for example, that for an individual with $f = f_{max}$, p_m is 0 and no mutations are applied. The mutation is applied by choosing two random positions in the selected individual, and two qubits are mutated by a quantum-not gate that exchanges the values of the amplitude probabilities; the gate is defined as follows:

$$U(\theta) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{72}$$

3.6. Differential Evolution Algorithm

The DE algorithm is a power population-based and stochastic evolutionary algorithm to solve continuous global optimization problems that can be constrained or unconstrained;

it maintains a population of candidate solutions and creates new candidate solutions by combining existing ones according to its formula and then keeping whichever candidate solution has the best score or fitness on the considered optimization problem. In this way, the optimization problem is treated as a black box that provides a measure of quality given by a candidate solution.

The main steps of the DE algorithm are as follows: (i) generate the initial population of the specified size; the population evolves by using (ii) mutation, (iii) crossover, and (iv) selection operators during each generation; and (v) step (ii) is repeated until a termination criterion is obtained [56].

In the initialization step, a number N of populations is set and each of them has a number M of D-dimensional vectors (D is a number related to the nature of the problem), which are potential solutions to the problem. The i-th individual, i.e., the i-th D-dimensional vector, is given as follows:

$$x_{i,G} = \left\{x_{i,G}^{1}, x_{i,G}^{2}, \ldots, x_{i,G}^{D}\right\} \tag{73}$$

where G identifies the number of the generation. Initial individuals are set with the following formula:

$$x_{i,0} = x_{min} + rand(0,1) * (x_{max} - x_{min}) \tag{74}$$

where x_{min} and x_{max} are vectors which identify the limits of the solution, and $rand(0,1)$ is a function to generate random individuals with a normal distribution.

In the mutation step, mutation vectors $v_{i,G}$ are generated for each individual of a population, which is called the target vector. The basic mutation strategies are as follows [57]:

$$(1)\ DE/rand/1:\ v_{i,G} = x_{r1,G} + F(x_{r2,G} - x_{r3,G}) \tag{75}$$

$$(2)\ DE/best/1:\ v_{i,G} = x_{best,G} + F(x_{r1,G} - x_{r2,G}) \tag{76}$$

$$(3)\ DE/current\ to\ best/1:\ v_{i,G} = x_{i,G} + F(x_{best,G} - x_{i,G}) + F(x_{r1,G} - x_{r2,G}) \tag{77}$$

$$(4)\ DE/best/2:\ v_{i,G} = x_{best,G} + F(x_{r1,G} - x_{r2,G}) + F(x_{r3,G} - x_{r4,G}) \tag{78}$$

$$(5)\ DE/rand/2:\ v_{i,G} = x_{r1,G} + F(x_{r2,G} - x_{r3,G}) + F(x_{r4,G} - x_{r5,G}) \tag{79}$$

where r1, r2, r3, r4, and r5 are integers randomly generated in the range between 1 and M, F is a scale factor, and $x_{best,G}$ is the best individual vector in a population during the G-th generation.

As for the crossover step, trial (or test) vectors $u_{i,G}$ are created by crossing each individual $x_{i,G}$ (or target vector) and the corresponding mutation vector $v_{i,G}$; it needs to form a set of trial vectors used in the selection step; there are two types of crossover methods largely used in the literature: binomial and exponential. In the former, the test vector is given as follows:

$$u_{i,G}^{j} = \begin{cases} v_{i,G}^{j} & \text{if } rand_{i,j}(0,1) \leq CR \\ x_{i,G}^{j} & \text{otherwise} \end{cases} \tag{80}$$

where CR is the crossover factor in the range 0 and 1 and controls the probability that a component of a trial vector is obtained from either a mutant or a previous individual, and j is between 1 and D. In this method, the distribution of the number of parameters inherited from the mutant is (almost) binomial. On the other hand, in the latter, a random integer r in the range $[1, D]$ is chosen and it is the component starting point for the crossover process. The components for the trial vector $u_{i,G}^{j}$ are donated from the mutated vector until a Bernoulli experiment linked to a probability CR is true [58].

$$u_{i,G}^{j} = \begin{cases} v_{i,G}^{j} & j = r, r+1, \ldots \text{until Bernulli experiment is true or } j = D \\ x_{i,G}^{j} & \text{otherwise} \end{cases} . \tag{81}$$

In the end, in the selection step, the objective function of the problem is evaluated for all generated trial vectors and the corresponding target vectors; the best one is selected, and it is a new individual of the next generation. The selection is given as follows:

$$x_{i,G+1} = \begin{cases} u_{i,G} & \text{if } f(u_{i,G}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{82}$$

There are two main differences between the GA and DE algorithm: (i) the GA is conducted by changing some genes in the chromosomes, while in the DE algorithm, the mutant individuals are obtained by adding the difference between two individuals to a third individual, which has a stronger global search capability; (ii) in the traditional GA, offspring individuals replace their parent individuals with a certain probability, while in the DE algorithm, such a condition only happens when the fitness of the offspring is better than the fitness of their parents; in such a way, a greatly increased convergence speed is provided [59].

Visible Light Communication Using a Modified Differential Evolution Algorithm

Authors in reference [59] have developed a Visible Light Communication (VLC) system where the VLC positioning model is transformed into a global optimization problem and a DE algorithm is used to calculate the z-coordinate of the receiver by adopting a modified DE algorithm that is called the self-adaptive DE algorithm; in this work, the 3D problem is reduced to a 1D problem, improving the speed of localization. Because the individual dimension is one (only the z-coordinate), the crossover process is ignored. The 3D positioning strategy exploits some specific equations to calculate the overlapped area of three circles determined by the position of the sources and the target, and is obtained by the projection on the x-y plan; the DE algorithm is used to solve the specific equation and calculate the z-coordinate by finding the optimal fitness value; the x and y coordinates are calculated by exploiting three equations for trilateration in the projection plan.

The proposed algorithm has three steps: (i) the generation of the population, (ii) calculation of the fitness, and (iii) mutation operator. The fitness function evaluates the quality of each of the candidate solutions in the population. In the VLC-based 3D positioning system, the individual with the smallest overlapped area can be regarded as the estimated position. The fitness equation can be treated as the overlapped area S and is represented as follows:

$$\text{fitness}(z) = S \tag{83}$$

If no overlapped area is formed between the three circles, the fitness is zero; the presumed height z is increased by 0.2 m for each iteration until the overlapped area appears. If the fitness $\leq 10^{-7}$, then the currently estimated height jumps out of the iteration loop and it is considered as the optimal solution z_e; otherwise, the iteration process goes on (step (iii)). In the case that the number of iterations reaches the maximum, and the fitness is still higher than 10^{-7}, the algorithm goes back to step (i).

The authors considered three types of DE algorithms: (i) DE/rand/1/binomial; (ii) DE/current-to-best/1/binomial; and (iii) the proposed self-adaptive DE algorithm. The first and the second are two traditional DE algorithms which adopt the strategy given in (75) and in (77), respectively; furthermore, the DE/current-to-best/1/binomial benefits from its fast convergence because the best solutions in the search space are used and its performance is better than the performance of the DE/rand/1/binomial algorithm. As for the third algorithm, the proposed self-adaptive DE algorithm is derived on the following observation: as the iteration number increases, the difference among individuals is reduced, and therefore a smaller amount of adjustment is needed because an individual has a better fitness and most of its information can be retained; on the contrary, when the number of iterations are low, the solution is still far from the optimal solution and a larger scaling factor might help the exploration process. Then, the authors proposed the following variation:

$$v_{i,g+1} = x_{best,g} + F(x_{r2,g} - x_{r3,g}) \tag{84}$$

where F is defined as follows:

$$F = \left(\frac{|\mathrm{fit}(x_i) - \mathrm{fit}(x_{best})|}{\sigma + \mathrm{fit}(x_{worst}) - \mathrm{fit}(x_{best})} \right) \times (F_{max} - F_{min}) + F_{min} \tag{85}$$

where x_{best} and x_{worst} are the best and worst individuals in the population, namely the vectors with the lowest and highest fitness values in the entire search space, and $\sigma = 10^{-13}$ is a constant added to avoid the zero division. In such a way, the scaling factor varies between the maximum F_{max} and minima F_{min} values according to its difference with the best individual at each generation.

3.7. Mind Evolutionary Algorithm

The Mind Evolutionary Algorithm (MEA) is a method which emulates the learning modes and activities of the human mind. Unlike genetic-based algorithms, which use mutation, crossing, and selection operators, in the MEA, an evolutionary process is also exploited but with the difference that two operations named similartaxis and dissimilation are adopted to overcome the drawback of the bad solutions generated during the mutation process; furthermore, every generation's evolutionary information is memorized, and it is used to guide the evolution toward the global optimum [60].

The MEA steps are defined as follows:

1. Initialization.
2. Similartaxis and local competition.
3. Dissimilation and global competition.

In the initialization, two main spaces are considered: the superior group space, which is composed of S superior groups, and the temporary group space with T temporary groups. Every group is initialized with m individuals, which are generated by considering an individual as the center and adopting a uniform distribution to generate the other individuals around the center. Furthermore, every individual is scored in agreement with their adaptability to the search space [60,61].

The similartaxis and local competition phase aims to make the subgroups mature by performing a local competition. In every group, individuals compete with each other to find the local optimum. To create a new generation, a winner who has the best score is selected, and a new generation is created around it. Then, individuals are scored again to identify if there is a better new solution or to keep the previous one; in such a way, a new generation can be created, competition starts again, and so on. The information of every new winner is recorded on the local billboard of the group. The generation process in a group continues until the maximal and minimal scores within one group satisfy the mature condition; then, the information related to the mature group is stored in the global billboard. The process to make the group mature is named similartaxis and is performed for both superior and temporary space [60,61].

In the dissimilation and global competition phase, all groups compete against each other by observing the information reported on the global billboard. If, in the temporary space, there is a group with a higher score than any group in the superior space, then this temporary group will replace the bad groups in the superior space. Moreover, if the score of any mature group in the temporary space is lower than the score of any group in the superior one, this temporary group will be discarded, and new ones will be generated in the temporary space with random values. The generation process is called dissimilation, and in such a way, the similartaxis can continue in every group [60,61].

The aforementioned operations are repeated until the scores of the groups in the superior space are so high that they cannot be further improved. At this point, the algorithm is considered convergent, and the winning individual of the group with the best score in the superior space is the global optimum [61]. The architecture of the MEA is reported in Figure 8 [60].

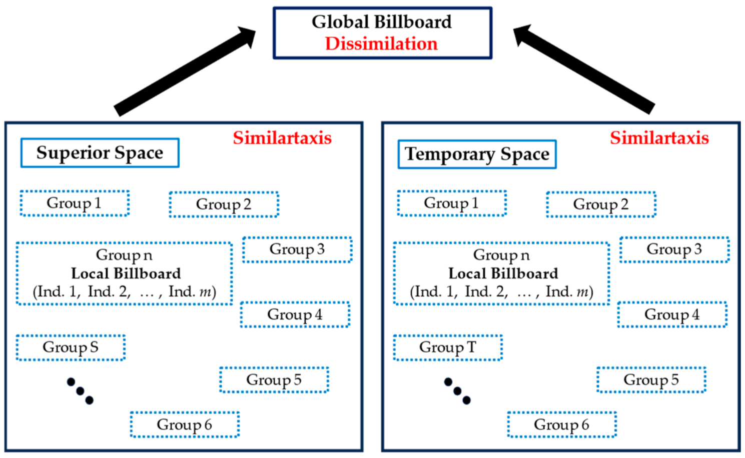

Figure 8. The architecture of MEA.

A Mind Evolutionary Algorithm-Based Geomagnetic Positioning System for Indoor Localization

The authors in reference [60] propose a solution to solve the problem in the pedestrian dead reckoning field where a large error is introduced when a walker goes backwards or laterally but the position is still updated forward. To address this problem, they propose a system for the indoor localization of smartphones based on magnetic positioning and the pedestrian dead reckoning (PDR) method, which are fused using an Extended Kalman Filter (EKF).

To improve the performance of magnetic positioning, the MEA-based heuristic searching strategy is adopted to search for the optimal magnetic position.

The geomagnetic features are extracted using a three-axis magnetometer installed on a smartphone. The magnetic vector measured in the device coordinate reference system is transformed into the equivalent vector in the geographic coordinate reference system in order to obtain a measurement that is independent of the attitude of the device. The transformation can be expressed as follows:

$$B_G = \left(C_n^b\right)^T B_D \tag{86}$$

where B_G is the magnetic vector in the geographic coordinate system, C_n^b is the matrix for the transformation, and B_D is the magnetic vector in the device coordinate system.

Then, five magnetic features for matching are extracted from B_G and they are the three magnetic vector components, the horizontal component of the magnetic field, and the magnitude of the vector.

A fine-grained magnetic database is constructed by linearly interpolating the features and the coordinates. Once the fingerprint has been obtained, the MEA can be performed to estimate the magnetic position. The proposed algorithm exploits the large number of samples that are provided by the measurement process, which can be considered to be at a high frequency; over an interval time of one second, these samples generate many temporary positions that are used to generate a population for the MEA; by considering an epoch every second identified by k, the population M(k) can be represented as follows:

$$M(k) = \{g_k(x_1, y_1), \ldots, g_k(x_i, y_i), \ldots, g_k(x_n, y_n)\} \tag{87}$$

where $g_k(x_i, y_i)$ is the i-th individual (i.e., the estimated position) of $M(k)$, x_i and y_i are the coordinates values, $i = 1, 2, \ldots, n$ is the number of the sample, and n is the sampling frequency. The score is calculated considering the following formula:

$$\text{score}\{k, i\} = \frac{1}{\|G(k-1) - g_k(x_i, y_i)\|_2} \tag{88}$$

where $G(k-1)$ is the previous estimated magnetic position.

As the first step, the MEA initializes the population (using the data generated by measurements), and the temporary individuals (i.e., the geomagnetic positions) are scored using Equation (88). The positions with higher scores are selected as centers of the superior groups, whereas the positions with lower scores are selected as centers of the temporary groups. Around the centers, individuals of the population are generated with a random distribution, and they are scored using Equation (88).

As the second step, the MEA executes the similartaxis where individuals within the same subgroup compete by comparing their scores. The second step is completed when the mature condition $|\mathrm{score}_{\max} - \mathrm{score}_{\min}| \leq \varepsilon$ is obtained.

In the end, as the third step, the MEA executes dissimilation for the global competition. The MEA executes similartaxis and dissimilation until convergent conditions are satisfied and the individual with the best score in the mature superior groups is selected as the estimation of the final magnetic position.

3.8. Particle Swarm Optimization

PSO algorithms are based on a simplified social model that tries to emulate the behavior of a swarm of animals such as bees or birds that are searching for a food source. It can be explained as follows [15,62].

PSO finds a solution by exploiting information from each individual of the population (or swarm) and also information from the swarm as a whole. At the beginning of the process, the number of particles in the swarm and the number of iterations are set, and an initial population distributed throughout the design space is randomly generated. In the next step, the position of each particle is updated using the following formula:

$$x_i^{q+1} = x_i^q + v_i^q \Delta t \tag{89}$$

where i is the i-th individual in the swarm, q is the q-th iteration, v_i^q is the velocity vector of the i-th individual at the q-th iteration, and Δt is the time increment. At the beginning, the velocity vector of each particle is initialized randomly and is updated at each iteration using the following formula:

$$v_i^{q+1} = \omega v_i^q + c_1 r_1 \frac{(p_i - x_i^q)}{\Delta t} + c_2 r_2 \frac{(p^g - x_i^q)}{\Delta t} \tag{90}$$

where ω is the inertial parameter, r_1 and r_2 are random numbers between 0 and 1, c_1 and c_2 are the trust parameters, p_i is the best point found so far by the i-th particle, and p^g is the best point found by the swarm. The search behavior of the algorithm is controlled by the inertial parameter ω; the larger the value, the larger the focus in global searching, whereas the smaller the value, the larger the focus in local searching; reference values for these two extremes are 1.4 and 0.5, for either a more focused global search or for more focused local search, respectively [15]. The c_1 and the c_2 are two trust parameters; the former, also referred to as cognitive memory, indicates how much the particle trusts itself, whereas the latter, also referred to as social memory, indicates how much the particle trusts the group. The p^g can be defined as either considering some best points among a subset of particles or considering the best point among all particles of the swarm. The values of ω, c_1, and c_2 parameters must be tuned in agreement with the managed problem.

The main drawbacks of the method are the tuning parameters and the fact that it is an inherently unconstrained optimization algorithm. The last one is a common characteristic of most of the evolutionary algorithms; researchers have tried to deal with this problem by proposing many constraint-handling techniques for evolution, and among these, the penalty function approach is the most widely used because it has the advantage that it is easy to implement [15].

3.8.1. Particle Swarm Optimization-Based Artificial Neural Network (PSO-ANN)

The authors in reference [63] use an Artificial Neural Network (ANN) to solve the fingerprint localization problem by exploiting the Received Signal Strengths Indicator (RSSI) of the Wi-Fi signal in an indoor environment. The ANN provides the localization of the target, and it is trained by a traditional PSO algorithm; this approach provides a strong learning skill which allows one to better deal with complicated indoor environments. Traditionally ANNs are trained by exploiting a backpropagation algorithm that is characterized by a feedforward propagation of signals and a backpropagation of errors; this approach requires significant time for training and can be easily trapped in a local minimum. To solve these issues, authors adopt the PSO algorithm to find the parameters of the trained ANN; they define the following:

(i) Each fingerprint is composed by $F = [\varphi_i(t), P]$, where $\varphi_i(t)$ is the RSSI value at time t of the i-th Wi-Fi source, and P is the position of the collected fingerprint.
(ii) The particle of the PSO is given as $\text{particle}_j = [W_1, \ldots, W_n, B_1, \ldots, B_n]$, where W_j and B_j represent the weight and bias of the j-th neuron.
(iii) The updating equations for the particle position and velocity are defined by the following:

$$\begin{aligned} p_{k+1} &= p_k + v_{k+1} \\ v_{k+1} &= \omega v_k + r_1 c_1 \times (\text{pbest}_k - p_k) + r_2 c_2 \times (\text{gbest}_k - p_k) \end{aligned} \tag{91}$$

where v_k is the velocity in the k-th iteration, p_k is the particle in the k-th iteration, c_1 and c_2 are two acceleration coefficients, and pbest_k and gbest_k are the personal best position and global best position in the k-th iteration, respectively, r_1 and r_2 are two random constriction coefficients in the range (0, 1) sampled from a uniform distribution to prevent explosion and induce particles to converge on optima, and ω is the inertia factor which helps to control the ability of particles to jump out of the local optimum and find the global optimum.

Considering the above definitions, the proposed algorithm follows these steps:

1. Set the particle group size M, the max iteration number m, and the expected error e. Then, initialize randomly the initial position and velocity of the particles;
2. For all RSSI training sets $[\varphi_i(t), P]$, put $[\varphi_i(t)]$ in the ANN model and compute the ANN output $\widetilde{P}$;
3. Update the *pbest* and *gbest* according to the distance between the predicted position $\widetilde{P}$ and the real position P;
4. Update the particle position and velocity according to Equation (91);
5. Repeat steps 3 and 4 until the distance between $\widetilde{P}$ and the real position P is less than the expected error or the max iteration number n has been reached.

3.8.2. Particle Swarm Optimization-Based Minimum Residual Algorithm

The authors in reference [64] adopt an RSSI range-based localization method for a mobile robot working in an indoor environment where the localization is estimated by a Minimum Residual (MinRe) localization algorithm based on PSO.

According to the principle of MinRE localization, the sum of the target's squared residual between the real and the measured distances is given as follows:

$$f(X) = \sum_{i}^{N} \{\|X - AN_i\| - d_i^{mes}\}^2 \tag{92}$$

where $f(X)$ is the function to be minimized, X is the real vector position to be found on the target, AN_i is the vector position of the i-th anchor node, N is the total number of nodes, and d_i^{mes} is the measured distance from the RSSI, which can be expressed as $d_i^{mes} = d_i + \omega_i$, where d_i is the unknown real distance and ω_i is the noise.

The term $\|X - AN_i\|$ represents the Euclidean distance between X and AN_i and it can be expressed as follows:

$$\|X - AN_i\| = \sqrt{(x - x_i)^2 + (y - y_i)^2 (z - z_i)^2} \tag{93}$$

where x, y, and z are the coordinates of the position of the target X, and x_i, y_i, and z_i are the coordinates of the position of the AN_i. If there is no error, $f(X)$ would be zero.

The PSO algorithm is applied to estimate the position of the target and the final problem can be written as follows:

$$X = \arg\min f(X) = \sum_{i}^{N} \left\{ \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - d_i^{mes} \right\}^2 \tag{94}$$

Setting the number of particles M, the maximum number of iterations W, considering w as the index of the particle updating iteration for this problem, and m as the index of the m-th particle, the solution of the problem follows these steps:

1. Initialization, where the particles and their initial velocity are generated randomly within the target environment: $X_m^1 = \left[x_m^1, y_m^1, z_m^1\right]$, and $V_m^1 = \left[vx_m^1, vy_m^1, vz_m^1\right]$. The fitness $f(X_m^1)$ of the m-th particle is calculated and the initial individual best value is set: $B_m^1 = X_m^1$. The initial global best fitness of the particle swarm is the particle which achieves the minimum of $f(X)$ and it is represented as $B_g^1 = \text{argmin}\, f(X_m^1)$ where $m = 1 : M$.
2. Particle velocities and positions, which are updated in agreement with the individual best and global best values at ω-th iteration for $2 \leq w \leq W$.

$$\begin{aligned} V_m^w &= \omega V_m^{w-1} + c_1 \xi \left(B_m^{w-1} - X_m^{w-1}\right) + c_2 \eta \left(B_g^{w-1} - X_m^{w-1}\right) \\ X_m^w &= X_m^{w-1} + V_m^{w-1} \end{aligned} \tag{95}$$

   where ω is the inertia weight that controls the exploration search region, c_1 and c_2 are the acceleration constants between 0 and 2, and ξ and η are random numbers between 0 and 1.
3. The updating of the optimal particle and optimal swarm is initiated:

$$\begin{aligned} B_m^w &= \begin{cases} B_m^{w-1}, & f(X_m^w) > f(B_m^{w-1}) \\ X_m^w, & f(X_m^w) \leq f(B_m^{w-1}) \end{cases} \\ B_g^w &= \begin{cases} B_g^{w-1}, & \min_{m=1:M} f(X_m^w) > f(B_g^{w-1}) \\ \arg\left(\min_{m=1:M} f(X_m^w)\right), & \min_{m=1:M} f(X_m^w) \leq f(B_g^{w-1}). \end{cases} \end{aligned} \tag{96}$$

4. The stopping criterion is checked: if $w < W$, then set $w = w + 1$ and go to step 2; otherwise, the estimated position is $\tilde{X} = \overset{w}{B_g}$.

The estimated $\tilde{X}$ is the position at the time t where the localization is estimated.

3.8.3. Mobile Robot Localization Based on PSO and Simultaneous Perturbation

The authors in reference [65] propose a mobile robot localization based on the PSO estimator where the system exploits the dynamic optimization to find the best robot pose estimate, recursively. The proposed solution is an alternative to using EKF or PF. PSO searches recursively and stochastically along the state space for the best robot pose estimation. Unlike the localization based on PF, in the proposed method, the resampling step (where samples with negligible weights are discarded and substituted by resampling) is not required, and no noise distribution is required either. The robot localization x_t at the t-th epoch is modelled like a Markov process (a random process in which the future is independent of the past given the present), where the initial distribution is $p(x_t)$.

Considering a maximum, a posteriori point of view, the localization estimation can be considered as an optimization problem, where the process seeks to estimate the pose that maximizes the a posteriori probability density, defined as follows:

$$\arg\max_{x} \, p\left(x^t \middle| z^t\right) \tag{97}$$

where $x^t = \{x_0, \ldots, x_t\}$ is the history of the estimated positions, and $z^t = \{z_0, \ldots, z_t\}$ is the history of the observations. By applying Bayes, the posterior over the robot path is given by the following:

$$p\left(x^t \middle| z^t\right) = \eta p\left(z_t \middle| x^t, z^{t-1}\right) p\left(x_t \middle| x^{t-1}, z^{t-1}\right) p\left(x^{t-1} \middle| z^{t-1}\right) \tag{98}$$

where η is a normalizing constant. The problem can be reformulated as follows:

$$\arg\max_{x} \left(\prod_{i=1}^{t} p\left(z_i \middle| x^i, z^{i-1}\right) \prod_{i=1}^{t} p\left(x_i \middle| x^{i-1}, z^{i-1}\right) p(x_0) \right) \tag{99}$$

where $p(x_0) = p\left(x^{t-1} \middle| z^{t-1}\right)$ and i is an index to count the epochs. After some mathematics passages, the objective function can be expressed as follows:

$$f_0(x_t) = \log\, p(z_t|x_t) + \log\, p(x_t|x_{t-1}) + f_0(x_{t-1}) \tag{100}$$

where $f_0(x_{t-1})$ considers the previous history for all the $t-1$ epochs; it is suitable for an iterative application.

Now, the following is considered: (i) the motion model that gives the robot pose at time t. Specifically, this is considered as follows:

$$x_t = f(x_{t-1}, u_t) + \omega_{t-1} \sim p(x_t|x_{t-1}, u_t) \tag{101}$$

where f is a non-linear function, ω_t is the Gaussian noise distribution with covariance matrix Q_t, and u_t is the control; (ii) the measurement model for the observed Q_t feature is defined as follows:

$$z_t = h(x_t, \theta) + v_t \sim p_{\theta t}(z_t|x_t) \tag{102}$$

where h is the function of the model, v_t is the measurement Gaussian noise distribution with covariance matrix R_t; (iii) the state transition probability density $p_{\omega t}\left(x_t \middle| x_{t-1}\right)$ derived from the robot motion model and the observation probability density $p_{\upsilon t}(z_t|x_t)$ derived from the observation model the problem is rewritten in the following form:

$$\min_{x_t} \left(\log p_{\upsilon t}(z_t|x_t) + \log\, p_{\omega t}(x_t|x_{t-1})\right) \tag{103}$$

where $p_{\upsilon t}(z_t|x_t) \propto \exp\left(\frac{1}{2}(z_t - \hat{z}_t)R_t^{-1}(z_t - \hat{z}_t)^T\right)$ and $p_{\omega t}\left(x_t \middle| x_{t-1}\right) \propto \exp\left(\frac{1}{2}(x_t - \hat{x}_t)Q_t^{-1}(x_t - \hat{x}_t)^T\right)$, where $\hat{z}_t$ and $\hat{x}_t$ are the estimated measurement vector and the estimated state vector, respectively; then, the fitness function can be written as follows:

$$\text{Fitness} = (z_t - \hat{z}_t)R_t^{-1}(z_t - \hat{z}_t)^T + (x_t - \hat{x}_t)Q_t^{-1}(x_t - \hat{x}_t)^T. \tag{104}$$

It is a quadratic and strictly convex function; the problem is solved using PSO, which can be applied to complex problems such as non-linear and non-differentiable problems.

It is now assumed that the current particles of PSO are given as follows:

$$C_t^G = \left\{x_{t,1}^G, \ldots, x_{t,N_p}^G\right\} \tag{105}$$

where C_t^G is an N_p dimensional vector, and $x_{t,i}^G$ represents each candidate solution i to the optimization problem at iteration G at time t. In standard PSO, the local information of an objective function obtained, for example, from the gradient, is not exploited. This means that even if a search point of PSO around the optimal value exists, the particle could go past the optimum point, since local information is not considered; nevertheless, the searching ability of PSO can be improved when gradient information is considered, but with the risk of being trapped in a local minimum and missing the global solution of the problem. In order to overcome this problem, authors combine PSO with the Simultaneous Perturbation (SP) method and gradient. The SP is a method that is given as follows:

$$\begin{aligned} x_{t+1} &= x_t + a\Delta g_t \\ \Delta g_t^j &= \frac{f(x_t + c_t) - f(x_t)}{c_t^j} \end{aligned} \tag{106}$$

where a is a positive constant, c_t and c_t^j are a perturbation vector and its j-th element, which is determined randomly, and Δg_t^j represents the j-th element of Δg_t, which estimates the gradient.

As proposed in reference [66], the combination of PSO and SP gives rise to the following equations:

$$\begin{aligned} x_{t+1} &= x_t + \Delta x_t \\ \Delta x_{t+1} &= \chi(-a\Delta g_t + \phi_1(p_t - x_t) + \phi_2(n_t - x_t)) \end{aligned} \tag{107}$$

where x_t is the particle, p_t is the best value of the particle, n_t is the best value of the swarm, ϕ_1 and ϕ_2 are two positive numbers in a predefined range, and χ is a gain coefficient. To provide global and local search capabilities at the same time, and keeping in the meantime population diversity, the authors define a combination of PSO and SP, given as follows:

$$x_t^i = x_{t-1}^i + v_t^i \tag{108}$$

$$v_t^i = \begin{cases} v_{max} & v_t^i > v_{max} \\ w(-\gamma\nabla\psi(x_{t-1}^i)) + c_1 r_1 (P_{pest} - x_{t-1}^i) + c_2 r_2 (P_{gest} - x_{t-1}^i) & -v_{max} < v_t^i < v_{max} \\ -v_{max} & v_t^i < -v_{max} \end{cases} \tag{109}$$

where $c_1 = 2$, $c_2 = 2$ are positive coefficients, r_1 and r_2 are random numbers in the interval (0, 1), v_{max} is the maximum velocity limited to 10% of the dynamic range of the variables on each dimension, w is the inertial weight, γ is a positive constant, and $\nabla\psi$ is the gradient of the objective function.

3.9. Differential Evolution and Particle Swarm Optimization: DEPSO

The authors in reference [67] propose a hybrid algorithm that combines the DE algorithm and PSO and is composed of two main steps. In the former, mutation and selection operators are used to produce a new population, while in the latter, PSO is adopted to explore local optimums, followed by crossover and selection operations. In this way, it is possible to increase and decrease the extent of searching regions for the population.

The advantages are that the algorithm overcomes degradation defects, enhances the effectiveness of the particles, and improves accuracy and robustness.

Hybrid Algorithm DEPSO

In reference [67], authors use DEPSO in a Particle Filter (PF) algorithm to reduce the particle number and the computational time complexity. The DE algorithm aims to create a new candidate, and to select the best ones among the current individuals and the candidate for the next generation. In this work, the DE steps are developed as follows:

1. Mutation. Three individuals from the population are randomly selected and the i-the mutant vector v_k^i is generated with this equation:

$$v_k^i(g+1) = F(x_k^{r_2}(g) - x_k^{r_3}(g)) + x_k^{r_1}(g) \tag{110}$$

where F is the mutation parameter, r_1, r_2, and r_3 identify the three selected individuals among N, k identifies the k-th step of the PF, and g identifies the generation of the DE.

2. Crossover. This step performs the crossover operator to generate a candidate for the trial vector and the current individual. The j-th dimension of the candidate is calculated by the following:

$$u_k^{i,j}(g+1) = \begin{cases} v_k^{i,j}(g+1) & \text{if}((\text{rand}(0,1) \leq CR)\text{or}(j = j_{\text{rand}})) \\ x_k^{i,j}(g) & \text{otherwise} \end{cases} \tag{111}$$

where j_{rand} is a random integer in the range of $[1, n]$, where n is the number of dimensions of the decision variables and CR is the crossover parameter.

3. Selection. In this step, the fitness function of $u_k^i(g+1)$ and x_k^i are compared and the best current individual is selected.

$$x_k^i(g+1) = \begin{cases} u_k^i(g+1) & \text{if}(f(u_k^i(g+1)) \geq f(x_k^i(g))) \\ x_k^{i,j}(g) & \text{otherwise} \end{cases} \tag{112}$$

4. End. The end condition is reached when a maximum number of iterations are executed or when the population's average fitness meets requirements. Otherwise, the algorithm is repeated from step 1.

Considering the PSO algorithm, each particle of the swarm has a position X and a velocity V; it is recalled that their evolution is classically based on the following formulas:

$$\begin{aligned} V_i &= w \cdot V_i + c_1 \cdot \text{rand} \cdot (\text{pbest}_i - X_i) + c_2 \cdot \text{rand} \cdot (\text{gbest}_i - X_i) \\ X_i &= X_i + V_i \end{aligned} \tag{113}$$

where w is the inertia weight, pbest_i is the best position of particle X_i so far; gbest_i is the best position of the swarm so far; rand is a random number between (0,1); and c_1 and c_2 are acceleration constants.

The DE algorithm and PSO have both advantages and disadvantages; for example, the differential algorithms have (i) a good global search capability due to the mutation step, which increases the diversity of the population, (ii) good local search capability, which is improved by the crossover step, and (iii) a memory function due to the selection process; however, the convergence speed is slow. On the other hand, PSO is characterized by a faster convergence rate, but it has the drawback that it might be stuck in a local optimum without converging to the global one because the diversity of the population can be lost very easily. Then, to exploit the characteristics of PSO, i.e., the high rate of convergence and the possibility to explore a local minimum quickly, Equation (113) is simplified as follows:

$$\begin{aligned} V_i &= w \cdot V_i + c_1 \cdot \text{rand} \cdot (\text{gbest}_i - X_i) \\ X_i &= X_i + V_i \end{aligned} \tag{114}$$

After the aforementioned considerations, the hybrid algorithm DEPSO can be represented with the following steps:

1. Set $t = 0$, and randomly initialize all individual positions X_i and velocities V_i.
2. $t = t + 1$.
3. Calculate individual fitness function $f(X_i)$.
4. Update the position of particles using the equation for mutation (110) and selection (112).
5. Select the individual in the best position of the swarm so far.
6. Update the position and velocity of particles according to (114).
7. Perform crossover and selection using (111) and (112), respectively.
8. If the counter is larger than a threshold, then exit; otherwise, return to step 2.

Authors claim that DEPSO avoids particle degradation, improves positioning accuracy, reduces the number of particles needed for positioning, and enhances the diversity of particles and robustness of the system.

3.10. Bat Algorithm

The bat algorithm (BA) idea is inspired by the echolocation behavior of bats to find food; in nature, they usually emit short pulses, and when they encounter food, they change the frequency to tune the echolocation time and increase the location accuracy of the prey [68]. The parameters of loudness and frequency determine the global search capability, and if a bat is near its scope, the impulse rate increases, whereas loudness decreases. In the standard bat algorithm, each individual i has a defined position $x_i(t)$ and velocity $v_i(t)$ in the search space, which are updated iteration by iteration with the following set of equations:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{115}$$

$$v_i(t+1) = v_i(t) + (x_i(t) - p(t))\cdot f_i \tag{116}$$

$$f_i = f_{min} + (f_{max} - f_{min})\cdot \beta \tag{117}$$

where t is the t-th iteration, $p(t)$ is the current global optimal solution, f_i is the frequency of the i-th individual, $f_{min} = 0$, $f_{max} = 1$, and β is a random vector in the range [0, 1] with uniform distribution.

To implement the local search strategy, the following equation is considered:

$$x_i(t+1) = x_i(t) + \varepsilon \overline{A}(t) \tag{118}$$

where ε is a random number in the range $[-1, 1]$ and $\overline{A}(t)$ is the average loudness of the population.

The global search is controlled by loudness $A_i(t+1)$ and pulse rate $r_i(t+1)$, which are updated with the following equations:

$$A_i(t+1) = \alpha A_i(t) \tag{119}$$

$$r_i(t+1) = r_i(t) + (1 - \exp\ (-\gamma t)) \tag{120}$$

where $0 < \alpha < 1$ is the amplitude decrease rate and is a constant term, and $\gamma > 0$ is a constant. The initial values of loudness $A_i(0)$ and pulse rate $r_i(0)$ are set in the initialization.

The steps of the standard algorithm are as follows:

1. Initialize, for each bat, the position and velocity parameters and randomly generate the frequency with Equation (117).
2. Update the position and velocity of each bat with (115) and (116).
3. For each bat, generate a random number rand1 in the range [0, 1];
   if rand1 $< r_i(t)$, then update the temp position with (118) and calculate the fitness value for the corresponding i-th bat.
4. For each bat, generate a random number rand2 in the range [0, 1];
   if rand2 $< A_i(t)$ and if $F(x_i(t)) < F(p_i(t))$, where F is the fitness function, then update the loudness with (119) and the pulse rate with (120).
5. Sort each individual based on fitness values and save the best position.
6. If the final condition is met, then exit; otherwise, go to step 2.

Mobile Robot Localization Based on the Leader-Based Bat Algorithm

The authors in reference [69] propose a modified BA to solve the mobile robot global localization problem, which is often characterized by complex multimodal beliefs. In some problems, the standard BA might provide a premature convergence solution. Authors propose the leader-based bat algorithm (LBBA), introducing leaders who can influence the behavior of other individuals. This gives rise to more stable particle movements at multiple local optimum points, providing a convergence to the global optimum in more complex environments.

The LBBA is a nonparametric filter with N individuals, and each of them approximately corresponds to one region in the search space. The a priori states of individuals are represented by a set of random samples, and this enables the simultaneous representation of multiple optimal locations.

Unlike the standard BA, the LBBA uses both the best global g^* and the best individual x^*. The last one enables a higher diversity of solutions in the population, and, in addition, the performance and efficiency in the search for the global optimum are improved thanks to the adoption of leaders, who promote the creation of different colonies, decreasing the randomness of flight.

To determine the target position, the following object function is defined:

$$O(x_i) = \frac{(m_i - m)}{m} \tag{121}$$

where m_i is the measurement performed by the simulated sensor of the i-th individual, and m is the measurement performed by the real robot sensor.

The main steps of a general LBBA are as follows:

- Step 1. Initialization of x_i, v_i where $i = 1, \ldots, N$. Define the number of leaders $L = 1, \ldots, L_{max}$.
  Define the number of leaders $L = 1, \ldots, L_{max}$.
  Initialization of frequency f_i, impulse rate r_i, and amplitude A_i.
  Assess the fitness of each bat using the fitness function $fit(x_i)$.
  Determine leaders x_L^* with the fitness function.
  Define the best global bat and the relative best fitness: x^* and $fit(x^*)$.
- Step 2. Compute the following:
  - $f_i = f_{min} + (f_{max} - f_{min}) \cdot \beta$;
  - $v_i(t+1) = v_i(t) + (x_i(t) - x_L^*) \cdot f_i$;
  - $\hat{x}_i = x_i(t) + v_i(t+1)$, where $\hat{x}_i$ is a temporary variable.
- Step 3. If rand1 $> r_i$ then $\hat{x}_i = x^* + \delta$, where x^* is the best global variable, and δ is a random displacement variable.
- Step 4. Estimate the fitness of $\hat{x}_i$ using $fit(\hat{x}_i)$.
- Step 5. Define new leaders $\hat{x}_L^*$.
- Step 6. If $fit(\hat{x}_i) \leq fit(x_i)$ and rand2 $< A_i$, then compute the following:
  - $x_i = \hat{x}_i$;
  - $fit(x_i) = fit(\hat{x}_i)$;
  - $r_i = r(0)(1 - e^{-\gamma t})$;
  - $A_i(t+1) = \alpha A_i(t)$.
- Step 7. If $fit(\hat{x}_i) \leq fit\left(x^*\right)$, then compute $x^* = \hat{x}_i$, and $fit(x^*) = fit(\hat{x}_i)$.
- Step 8. If the condition is met, return x^*; otherwise, repeat from step 2.

Authors claim that the LBBA has better performances when compared to other algorithms for robot localization such as PF and PSO, showing the ability to escape from local minima in highly complex environments. Moreover, it also succeeds in solving the robot's kidnapping problem without premature convergences.

3.11. Artificial Bee Colony Optimization Algorithm

The Artificial Bee Colony (ABC) algorithm is a metaheuristic search algorithm inspired by the intelligent foraging behavior of honeybees in nature [70]. The algorithm is explained as follows [71]. The colony of artificial bees is composed of three groups: employed bees, onlookers, and scouts. The number of employed bees and onlooker bees is equal to the number of food sources. Every bee can only exploit one food source at a time. The scout bee was an employed bee that became a scout bee when it left the food source looking for a new one. A possible solution to the problem corresponds to a food source and the fitness of a solution corresponds to the nectar amount of a source. Each employed bee first explores the associated solution (or food source) according to their memories and explores possible solutions in corresponding neighborhoods. Then, they fly back to the hive and share the information about the food sources with the onlooker bees. The onlooker bees select the food sources, and a source is chosen using a probability criterion based on the value of fitness or amount of nectar: the food sources with more nectar have a higher probability of being selected. If a food source does not provide any better solution after several times that it is explored, the employed bee of that source becomes a scout bee and randomly selects a new source as it continues the work as an employed bee. The main steps of the algorithm are as follows [71]:

1. Initialization. A solution is considered a multi-dimensional vector x_i and SN solutions are randomly generated at the beginning; the components of the vector are generated by the following equation:
$$x_{ij} = x_j^{min} + rand(0,1) \times (x_j^{max} - x_j^{min}) \tag{122}$$
where i is one of the SN solutions, j is one of the D dimensions of the vector, the x_j^{min} and x_j^{max} represent the lower and upper bound of the j-th dimension, respectively, and $rand(0,1)$ provides a random number in the range [0, 1].
2. Employed bee phase. Each employed bee is associated with a solution x_i and then new solutions, v_i, are explored (or new food sources are generated) using the following equation:
$$v_{ij} = x_{ij} + rand(-1,1) \times (x_{ij} - x_{kj}) \tag{123}$$
where v_{ij} is the j-th component of a new solution, $rand(-1,1)$ provides a random number in the range [−1,1], x_{kj} is the j-th component of another food source x_k that is selected randomly from the remaining population, and $k \neq i$. The old solution, x_{ij}, and the new one, v_{ij}, are compared and the employed bee brings back the information of the better one, i.e., the solution with a smaller objective value or the best fitness. It is called the greedy selection strategy. The fitness values, fit_i, are calculated with the following equation:
$$fit_i = \begin{cases} \frac{1}{1+f_i} & f_i \geq 0 \\ 1 + abs(f_i) & f_i < 0 \end{cases} \tag{124}$$
where f_i is the output of the objective function calculated using the x_i solution.
3. Onlooker bee phase. The employed bees share the fitness information with the onlooker bees. Each onlooker bee chooses a solution for itself using a probability criterion: the higher the probability of a solution, the higher the possibility of being chosen. The selection probability of the i-th solution is called the roulette wheel selection and the probability p_i is calculated as follows:
$$p_i = \frac{fit_i}{\sum\limits_{m=1}^{SN} fit_m}. \tag{125}$$
After the selection, each onlooker bee generates a new solution in the surroundings of the chosen solution by using (123). Each generated solution is compared with the corresponding previous food source via the greedy selection strategy, and if the new one is better, it is set as $x_{ij} = v_{ij}$. The difference between the updating process of the employed bees and the onlooker bees is that the employed bees update all solutions, while the onlooker bees only update the selected one.
4. Scout bee phase. A solution may be explored several times without providing a better solution in the surroundings. After a pre-set number of explorations in the surroundings of the same solution without better results, the solution is abandoned,

and then the corresponding employed bee transforms into a scout bee to randomly generate a new solution by using (122). Then, it continues as an employed bee.

Multistage Localization in Wireless Sensor Networks Using the Artificial Bee Colony Algorithm

The authors in reference [72] propose a method based on the ABC algorithm to solve the sensor localization problem in a wireless sensor network, providing some simulation results. In the considered scenario, N nodes are deployed in a square plan, and each of them has a communication range of r units. A few nodes of them are considered as anchor nodes and they are in a known position; the other nodes are deployed in random locations. A node in an unknown position with at least three non-collinear anchors in its range can estimate the distances from anchors by parameters such as RSSI, Time of Arrival (ToA), and so on; if at least three anchor nodes are not available, the sensor cannot be localized. Each localizable node executes the ABC algorithm to estimate the coordinates of its location; once a sensor in an unknown position has been localized, it is used as a new anchor node to identify the other sensors. The aim is to determine the locations of all or as many nodes as possible, and in such a way, the distributed localization problem is modelled as a two-dimensional optimization problem.

The ABC algorithm computes the estimated location $(\hat{u}_{jx}, \hat{u}_{jy})$, finding the minimum of the mean localization error E_j, which is expressed as follows:

$$E_j = \frac{1}{3}\sum_{i=1}^{3}\left(\sqrt{\left(\hat{u}_{jx} - a_{ix}\right)^2 + \left(\hat{u}_{jy} - a_{iy}\right)^2} - \hat{d}_i\right) \tag{126}$$

where j is the j-th node in an unknown position, i is the i-th anchor node with known coordinates (a_{ix}, a_{iy}), $\hat{d}_i$ is the distance measured (that includes noise and errors), and $(\hat{u}_{jx}, \hat{u}_{jy})$ are the unknown coordinates.

The authors compared the performances of the ABC and PSO algorithms in finding the solution to the specific problem. The ABC algorithm can result in more accurate localization than PSO, but requires a higher computational time.

3.12. Simulated Annealing

The SA algorithm is a random-search technique that is based on the analogy between the process of metal crystalline formation from the liquid phase, and the process of searching for an optimal minimum in a problem. If the cooling process is too rapid, then the metal will solidify into a sub-optimal configuration. If the cooling process takes the proper time, the crystals within the material will solidify optimally into a state of minimum energy, which is the ground state associated with the optimal global solution to the problem. This is a heuristic method that does not need any particular assumptions and can be used for non-linear and non-convex problems; in particular, it results in a suitable technique for solving combinatorial optimization problems.

The main steps of the SA algorithm are as follows [73]:

1. Generate a random initial solution vector $x = x_o$ for the system and then evaluate it using the objective function $f(x)$.
2. Initialize the temperature $T = T_0$.
3. Perturbate the previous solution to obtain a neighboring $x_i + \Delta x$. Evaluate the new solution.
4. A new solution $(x_i + \Delta x)$ is accepted as a new solution $x_{i+1} = x_i + \Delta x$ if $f(x_i + \Delta x)$ is better than $f(x_i)$.
5. Otherwise, accept the new solution $x_i + \Delta x$ with a probability $e^{-\left(\frac{\Delta f}{T_i}\right)}$, where $\Delta f = f(x_i + \Delta x) - f(x_i)$.
6. Reduce the system temperature according to the cooling schedule.
7. Repeat from step 3 until a stopping criterion is met (number of cycles or $T_i = 0$).

In the SA problem, temperature is a parameter that controls the probability in step 5. During the iterations, the lower the temperature, the lower the probability of accepting a bad solution. This mechanism helps the procedure to avoid being stuck in a global minimum at the beginning of the iterations, because it allows the algorithm to jump out of the minima looking for better solutions. To obtain a good result, it is important to properly set the initial value of temperature T. If the value is too high, then it takes more cycles to converge, and vice versa; if the value is too small, the algorithm may provide a local optimum.

In step 6, the choice of an appropriate cooling scheme is important for the algorithm's success. In the literature, there are several types of schemes, and they can be classified as either monotonic or adaptive. The former consists of a static decrease in the temperature at every iteration independently of the goodness of the solutions and independently of the current explored neighborhood structure, whereas the last one considers the quality of transitions, adopting mechanisms for the temperature decrease that can adopt also reannealing techniques [73].

Simulated Annealing-Based Localization in Wireless Sensor Network

The authors in reference [74] propose a SA localization (SAL) method for wireless sensor networks. It aims to estimate the position of sensors by determining the position of a set of anchors and the measured distance between them and the sensors. However, several factors can influence distance measurements, such as synchronization, multipath, fading, Non-Line of Sight (NLoS) conditions, and other sources of interference. To estimate the real distance of the sensor with a minimum error, the authors propose to solve the optimization problem with the SAL method. The objective function is defined as follows:

$$\mathrm{CF} = \sum_{i=1}^{N} \sum_{j \in N_i} \left(\hat{d}_{ij} - d_{ij} \right)^2 \tag{127}$$

where N is number of non-anchor nodes, N_i is the set of neighbors of the i-th node, $\hat{d}_{ij}$ and d_{ij} are the estimated distance and the measured distance of node i with its neighbor j, respectively.

The adopted cooling scheme for temperature is $T_{i+1} = \alpha^* T_i$, where $\alpha < 1$ is the cooling rate that is determined empirically. The solutions are perturbated as $\Delta d_{i+1} = \beta^* \Delta d_i$, where Δd is chosen to span the whole set of allowable moves and $\beta < 1$ is used to bias the generation of random moves at a lower T.

3.13. Ant Colony Optimization

Ant Colony Optimization (ACO) is a probabilistic method, and it was initially proposed to solve the Travelling Salesman Problem (TSP); in general, ACO is suitable for solving problems in which the searched solution is the best path through a graph. It is inspired by the behavior of an ant seeking a path between its colony and food. The ants live in a colony, roam around it looking for food, and communicate with each other by using pheromones which can be released on the ground when insects are moving. Once an ant finds some food, the insect brings it as much as possible back to the colony and, during the travel, it releases pheromones in agreement with the quantity and quality of the found food. Other ants can smell it, and when they come out of the colony looking for food, they choose the path with higher pheromones. The higher the pheromone along a route, the higher the probability that an ant chooses that path. In such a way, more ants follow the path, more pheromones are deposited, and the path has a higher probability of being chosen. After some iterations, the ants will choose the shortest path, which is expected to be the optimum of the problem.

Mathematically, the algorithm is explained as follows [75]. ACO can be understood considering a simple graph, $G = (V, E)$, with two nodes, where V represents the nodes of the graph (the nest v_s and the food source v_d), E represents two links, e_1 and e_2, between nodes with length l_1 and l_2, respectively, such that $l_1 < l_2$. The artificial pheromone trails are modelled for each path by τ_1 and τ_2 and they indicate the strength of the pheromone trail on the corresponding path. Every ant chooses with a probability given as follows:

$$p_i = \frac{\tau_i}{\tau_1 + \tau_2} \tag{128}$$

where i identifies the i-th link; about the lengths, it is $\tau_1 > \tau_2$, which means that link 1 has a higher probability. As for returning, every ant uses the same path chosen in the outward journey, and it updates the artificial pheromone value associated with the used link with the following equation:

$$\tau_i = \tau_i \frac{Q}{l_i} \tag{129}$$

where $Q > 0$ is a parameter of the model. It can be inferred that the shorter the path, the higher the amount of added pheromones. At each step (or iteration), all the ants are initially placed in node v_s and each of them moves from the nest to the food source. The evaporation of the pheromone is simulated by the following:

$$\tau_i = (1 - \rho) \tag{130}$$

where ρ is in the range $(0, 1]$ and regulates the pheromone evaporation. Finally, all ants return to the nest and reinforce their chosen path, as reported above. Iteration by iteration, the shortest path will be the one with the highest pheromones, and then, with the highest probability.

For real combinatory optimization problems with more than two nodes, the solution is constructed as follows [76]. Each of the m ants is randomly placed on a chosen node and then at each node, a state transition rule is iteratively applied. The unvisited nodes, the pheromone trail strength, and the length between two nodes bias the ant's choice. The partial tour is stored in a memory called the tabu list which needs to determine at each construction step the nodes to be visited, to guarantee that a feasible solution is built, and to retrace the trail of the related ant once it has been completed. When all ants end up constructing a path, the pheromones are updated using evaporation and new deposition; short paths are updated with a higher quantity of pheromones. The algorithm is summarized as follows [76]:

1. Set parameters and initialize pheromone trails.
2. Construct solutions.
3. Apply local search.
4. Update trails.
5. If termination conditions are not met, repeat from step 2.

There are different versions of ACO and historically, the first presented version was called the Ant System (AS); some variants of the ACO algorithms (here not investigated) are the Elitist AS (EAS), Rank-based AS (RAS), MAX–MIN Ant System (MMAS), Ant Colony System (ACS), and Hyper-Cube Framework (HCF).

ANN Topology Optimized by ACO

The author in reference [77] deals with the problem of finding a continuous path for a robot from an initial position to a goal position where collisions against static objects must be avoided. The proposed system exploits an ANN that provides the optimum path, and it is trained by adopting the swarm intelligence-based reinforcement learning (SWIRL) method based on PSO and ACO. Because the training process of Multi-Layer Perceptron (MLP) for pattern classification problems can be considered as graph problems where the neurons are vertices and the connections are directed edges (links), PSO is exploited to provide the best ANN connection weights, whilst the ACO is used to optimize the topology of the ANN. In such a way, the SWIRL provides both an appropriate architecture for the problem and the trained connection weights of the network.

At each step of the iteration process, the PSO is applied to adjust ANN connection weights within a given topology structure. Focusing only on the ACO algorithm, it is used to allocate training iterations among a set of candidate network topologies. The desirability $d(i)$ for the i-th ANN is given as follows:

$$d(i) = \frac{1}{h+1} \tag{131}$$

where h is the number of hidden nodes; each i-th ANN represents a topology of a set of ANNs with different numbers of hidden nodes. The pheromone concentration τ_i is initialized to 0.1, and it is expressed as follows:

$$\tau(i, t+1) = \rho\tau(i, t) + n_a(i)\frac{g(i)}{g_{sum}} \tag{132}$$

where ρ is the rate of evaporation, n_a is the number of ants returning from the i-th neural network, $g(i)$ is the global best for i, g_{sum} is the sum of all the current global bests, and t is the index iteration of the ACO. Each ant represents one training process for the PSO. During each ACO step, the ants go out into the topology space of ANNs, and they choose an ANN using a probability criteria, given as follows:

$$p(i) = \frac{[d(i)]^b[\tau(i, t)]^a}{\sum\left\{[d(i)]^b[\tau(i, t)]^a\right\}} \tag{133}$$

where a and b are constant factors which control the relative influence of pheromones and desirability, respectively [77].

3.14. Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) is a metaheuristic method inspired by the hunting behavior of humpback whales, and it has been proposed by authors in reference [78]. The algorithm mimics three behaviors: (i) encircling prey or siege predation, (ii) spiral bubble-net feeding maneuver, and (iii) searching for prey. In the first one, a whale is in a location looking for prey to encircle; at the beginning, the locations (or solutions) are random, and the current best solution, defined as the best search agent, is found by exploiting the fitness function; the other search agents or candidate solutions move iteration by iteration towards the current best one. Mathematically, the behavior of each whale can be modelled by the following:

$$D = \left|C{\cdot}X^*(t) - X(t)\right| \tag{134}$$

$$X(t+1) = X^*(t) - A{\cdot}D \tag{135}$$

where t is the t-th iteration, A and C are coefficient vectors, X^* is the position vector of the best solution obtained so far, X is a candidate position vector, and the dot $(\cdot)$ represents the element-by-element multiplication. Furthermore, it is possible to express $A = 2a{\cdot}r - a$ and $C = 2{\cdot}r$, where a is linearly decreased from 2 to 0 iteration by iteration, and r is a random vector in the range [0, 1] that allows it to reach any position of the design space. The used vectors in this explanation have the same dimension as the input vector of the problem; however, the algorithm can be applied to a problem with a dimension greater than three even if the algorithm is inspired by the analogy of a moving whale in a three-dimensional space.

In the second behavior (spiral bubble-net feeding manoeuvre), two approaches are adopted: the shrinking encircling mechanism and the spiral updating position. The former needs to push the X vectors towards X^* and it consists in decreasing the value of a iteration by iteration; in such a way, considering that r is random, the result is that A is composed by random values in the range $[-a, a]$. As for the second approach, to mimic an encircling movement, a spiral equation is created between the position of a whale and the best position; it is expressed as follows:

$$X(t+1) = D_i \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t) \tag{136}$$

where $D_i = \left| X^*(t) - X(t) \right|$ indicates the distance of the i-th candidate solution (or whale) to the best solution obtained so far, b is a constant for defining the shape of the exponential spiral, and l is a random number in the range $[-1, 1]$. Considering the analogy with whales, they are swimming around their candidate location prey within a shrinking circle and along a spiral-shaped path simultaneously. To update the position of whales and to mimic the simultaneous behavior of the shrinking encircling mechanism and the spiral model, in the algorithm they are chosen randomly with a probability of 50% in each iteration. The two approaches are expressed as follows:

$$X(t+1) = \begin{cases} X^*(t) - A \cdot D & \text{if } p < 0.5 \\ D' \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t) & \text{if } p \geq 0.5 \end{cases} \tag{137}$$

where p is a random number in the range [0, 1].

In the search for prey behavior, whales look for a prey location, randomly giving the algorithm the global searching capability. It is expressed as follows:

$$D = |C \cdot X_{rand} - X(t)| \tag{138}$$

$$X(t+1) = X_{rand} - A \cdot D \tag{139}$$

where X_{rand} is a random position vector chosen from the current whale population.

The WOA pseudo code is composed of four main parts, and they are given as follows [78]:

1. Initialize the whale population X_i with $i = 1, 2, \ldots, n$.
   Calculate the fitness of each search agent.
   Define the best search agent X^*.
2. While $t <$ max number of iterations:
   - For each search agent:
     Update a, A, C, l, and p.
     If $p < 0.5$:
     - If $|A| < 0.5$, update the position of the current search agent by (134).
     - Elseif $|A| \geq 0.5$, select a random search agent X_{rand} and update the position of the current search agent by (139).
       }

   Elseif $p \geq 0.5$, update the position of the current search agent by (136).

- End of for loop.
- Check if any search agent goes beyond the search space and amend it.
- Calculate the fitness of each search agent.
- Update X^* whether a better solution has been found.
- $t = t + 1$.

3. End.
4. Return X^*.

In this basic version of the proposed WOA, other evolutionary operations are not included, but they might have been included by allowing hybridization with other evolutionary search schemes.

Three Magnetic Targets Using Individual Memory-Based WOA and LM Algorithm

The authors in reference [79] deal with multi-permanent magnet localization, exploiting an array of magnetometers. The aim is to estimate the location of the permanent magnets inside the human body by measuring the magnetic field. To obtain the best estimation of the measurements, the authors propose an improved WOA combined with the LM algorithm.

Each magnetometer in the array measures the magnetic flux that is given by the linear superposition of the flux densities generated by each permanent magnet. The objective function E of the total error is obtained by considering the sum of the three errors between the measured values by the magnetometers and the theoretical values; it is given as follows:

$$E = \sum_{l=1}^{N} \left[\left(B'_{lx} - B_{lx}\right)^2 + \left(B'_{ly} - B_{ly}\right)^2 + \left(B'_{lz} - B_{lz}\right)^2 \right] \tag{140}$$

where each B'_l represents one of the triaxial components of the magnetic flux density measured by the l-th magnetometer, B_l is the theoretical value of one of the triaxial components of the magnetic flux, and N is the total number of magnetometers. A generic component B_l is given by the superimposition of the magnetic fields generated by all permanent magnets, and it includes the distance variables between each permanent magnet and each magnetometer.

The name of the proposed algorithm is the Individual Memory-based Whale Optimization and Levenberg–Marquardt (IMWO-LM) algorithm; the IMWO algorithm is used to provide an approximate solution of each magnet pose, which is given as a guess input to the LM algorithm, which is suitable for finding the local minima.

In the proposed IMWO, S is defined as the number of whales and each of them searches for a solution of dimension d; the vector position for the i-th whale is defined as $X^i = \left[x^i_1, x^i_2, \ldots, x^i_d\right]$. The three behaviors presented in the WOA are here modified to give whale the ability of self-memory and to always remember the previous best position. The IMWO behaviors are defined as follows:

1. Search for prey. The current search agent position is updated considering both the position of the randomly selected whale and the historical optimal position. It is given as follows:

$$\begin{cases} D^i_{rand} = \left|C \cdot X^i_{rand} - X^{t,i}\right| \\ D^i_{pbest} = CX^{(t-1),i}_{pbest} - X^{t,i} \\ X\prime^{t,i} = X^i_{rand} - AD^i_{rand} + AD^i_{pbest} \end{cases} \tag{141}$$

   where t identifies the t-th iteration, i identifies the search agent, $X^{t,i}$ is the current position, $X^{(t-1),i}_{pbest}$ is the historical optimal position, D^i_{pbest} is the random distance between the current position and its historical optimal position, X^i_{rand} is a randomly selected search agent, D^i_{rand} is the random distance between a randomly selected search agent and current search agent, C and A are weight vectors defined as in the WOA, and $X\prime^{t,i}$ is the newly updated location of the i-th search agent.

2. Encircling prey or siege predation. The best candidate search agent X^* of the current population is considered the target prey at each iteration; search agents have memory and can remember the previous best position. It is given as follows:

$$\begin{cases} D^i_{leader} = \left|CX^* - X^{t,i}\right| \\ D^i_{pbest} = CX^{(t-1),i}_{pbest} - X^{t,i} \\ X\prime^{t,i} = X^* - AD^i_{leader} + AD^i_{pbest} \end{cases} \tag{142}$$

   where D^i_{leader} is the random distance between the optimal candidate search agent and the current search agent, and X^* is the optimal candidate search agent in the whole population so far. In such a way, the current search agent is updated to different positions around X^* and $X^{(t-1),i}_{pbest}$.

3. Spiral bubble-net feeding maneuver or bubble predation. The helix-shaped movement of the whale is obtained using the spiral equation, considering the position of the current search agent, its historical optimal position, and the position of the optimal candidate search agent. The current position is given as follows:

$$X'^{ti} = \left(r_1 D^* + r_2 D^i_{pbest}\right) \cdot e^{bl} \cdot \cos(2\pi l) + r_1 X^* + r_2 X^{(t-1),i}_{pbest} \quad (143)$$

where $D^* = \left|X^* - X^{t,i}\right|$ is the distance between the current search agent and the optimal candidate search agent, b is a constant for defining the shape of the exponential spiral, l is linearly decreased from 1 to −1 throughout iterations, and r_1 and r_2 are random numbers in the range [0, 1].

The upper and lower boundary of the pose parameters are set, and at each iteration, the algorithm must check if the search agents violate them.

3.15. Summary of the Investigated Methods

In this section, Table 1 reports a summary of the investigated methods, where the main characteristics, the significative advantages and disadvantages, and the investigated applications are listed to have a comprehensive view of the conducted research.

Table 1. Summary of the investigated methods.

Investigated Methods	Main Characteristics	Significative Advantages (ADV) and/or Disadvantages (DIS)	Applications
Newton's Method	• Iterative method. • Fixed step size. • Suitable for unconstrained problems. • Second-order method. • Quadratic rate of convergence. • Needing of an initial design point.	• High computation cost. (DIS) • Needing a twice differentiable function. (DIS) • Good initial starting point. (DIS)	• No applications investigated for this method.
GD	• Iterative method. • Variable step size. • First-order method. • Dealing with convex functions. • Unconstrained problems. • Convergence time linked to the parameter α.	• Penalty function can be used. (ADV) • The parameter α needs to be set properly to avoid skipping the optimum. (DIS) • Function must at least once differentiable. (DIS) • It could skip the global optimum and converge to a local one. (DIS)	• No applications investigated for this method.
Gauss–Newton	• Iterative method. • Suitable for convex non-linear least-squares problems. • It requires at least once-differentiable function. • Assumption: the inverse of the gradient exists.	• It is suitable only for a specific class of problems. (DIS)	• No applications investigated for this method.
ILS Method/ LAMBDA	• Iterative method. • Suitable for ILS problems.	• It is suitable only for a specific class of problems. (DIS) • Increase the resolution speed of the problem. (ADV)	• To solve frequency ambiguity problems in GNSS applications [38,39].

Table 1. *Cont.*

Investigated Methods	Main Characteristics	Significative Advantages (ADV) and/or Disadvantages (DIS)	Applications
LM	• Iterative method. • Suitable for non-linear optimization problems. • It interpolates between the GN and the GD.	• A good initial guess is needed. (DIS) • If the problem has several local optima, a wrong global optimum could be provided. (DIS)	• No applications investigated for this method.
Lagrange Multipliers Method	• Analytic method. • Suitable for constrained problem (with equality and/or inequality). • Differentiable functions are needed.	• It guarantees only that an optimum has been found but does not guarantee that is the global one. (DIS)	• No applications investigated for this method.
Lagrange Duality	• Analytic method. • It transforms a minimization or maximization problem into its dual one.	• It could provide an easier problem to solve. (ADV)	• To estimate the measured range of radio source anchors [47].
GA	• Iterative method. • Metaheuristic method. • Suitable for constrained and unconstrained problems. • Inspired by natural selection of animals. • It works directly with strings of bits • It deals with discontinuous, nondifferentiable, stochastic, or highly non-linear functions. • It is suitable for mixed-integer type problems.	• It can solve many types of different problems. (ADV) • Large solutions space can be spanned. (DIS) • Slow solving speed. (DIS) • Each considered problem needs a specific tunning. (DIS) • Easy to tune. (ADV)	• Computation of parameters to superimpose information from different sensors and radio signal [9].
QGA	• It is a variation of the GA. • Unlike GA, QGA deals with Q-bits and therefore can exploits superimposition of states that allows to reduce the solution time convergence.	• The solving speed is $\sqrt{N}$ times faster than the speed of classic GA. (ADV) • High complexity. (DIS)	• To calibrate the parameters of a wheeled vehicle to compensate the systematic errors [11]
DE	• Iterative method. • Metaheuristic method. • Suitable for constrained and unconstrained problems. • Not inspired by natural selection of animals. • It works directly with vectors • It deals with continuous problems.	• Good global search capability. (ADV) • Good local search capability. (ADV) • A memory function. (ADV) • A slow convergence speed. (DIS) • Performance affected by the trial vector. (DIS) • Needing a tuning procedure. (DIS)	• To calculate the z-coordinate of the receiver in VLC system for localization purposes [59].

Table 1. *Cont.*

Investigated Methods	Main Characteristics	Significative Advantages (ADV) and/or Disadvantages (DIS)	Applications
MEA	• Iterative method. • Metaheuristic method. • Suitable for constrained and unconstrained problems. • Inspired by learning modes and human mind activities. • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Fast rate of convergence. (ADV) • It can deal with data from sensors that are used as population generator when work at high frequency. (ADV)	• To update the forward or backward movement of a walker in a PDR system [60].
PSO	• Iterative method. • Metaheuristic method. • Suitable for unconstrained and constrained (by using penalization methods) problems. • Inspired by natural swarm behaviors of animals. • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Fast rate of convergence. (ADV) • Needing a tuning procedure to set the parameters. (DIS) • Possibility to be stuck in a local optimum without converging to the global. (DIS)	• To train an ANN (used in a fingerprint localization system) [63]. • To estimate the RSSI range by minimizing the squared residual between the real and the measured distances [64]. • To solve the localization problem modelled as a Markov process in which the position is extracted by maximizing the a posteriori probability density. The approach is an alternative to EKF or PF [65]. • To estimate the best ANN connection weights during the training process for an on moving robot that is able to avoid collisions [77]
DEPSO	• Hybrid algorithm in which the concepts of DE and PSO are fused.	• Better exploration of local optima. (ADV) • Better convergence speed than classic DE. (ADV)	• To reduce the particle number and the computational time complexity in a PF [67].
BA	• Iterative method. • Metaheuristic method. • Suitable for unconstrained and constrained (by using penalization methods) problems. • Inspired by the echolocation behavior • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Fast rate of convergence. (ADV) • Good ability to explore the search space and to quit from local optima. (ADV) • Standard BA might provide a premature convergence, that it is improved by adopting LBBA. (DIS) • Needing a tuning procedure. (DIS)	• To localize the robot position by minimizing the error between the real and the measured position [69].

Table 1. *Cont.*

Investigated Methods	Main Characteristics	Significative Advantages (ADV) and/or Disadvantages (DIS)	Applications
ABC	• Iterative method. • Metaheuristic method. • Suitable for unconstrained and constrained problems. • Inspired by the intelligent foraging behavior of honeybees. • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Good ability to explore the search space. (ADV) • Slow rate of convergence. (DIS) • Needing a tuning procedure. (DIS)	• To solve the sensor localization problem in a wireless sensor network to get new anchors; the location is estimated by minimizing the mean localization error [72].
SA	• Iterative method. • Metaheuristic method. • Suitable for unconstrained and constrained problems. • Inspired by the process of metal crystalline formation from the liquid phase. • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Good ability to explore the search space. (ADV) • Slow rate of convergence. (DIS) • Needing a careful tuning procedure. (DIS)	• To estimate the real position of sensors by exploiting the anchors positions and the measured distances between them and the sensors [74].
ACO	• Iterative method. • Metaheuristic method. • Inspired by the behavior of ants. • Suitable to solve the problems in which the best path needs to be selected.	• Solve a specific class of problems. (DIS)	• To optimize the best ANN topology during the training process for an on moving robot that is able to avoid collisions [77]
WOA	• Iterative method. • Metaheuristic method. • Suitable for unconstrained and constrained problems. • Inspired by the behavior of a whale to capture a prey • It deals with discontinuous, non-differentiable, stochastic, non-linear, non-convex functions.	• Slow convergence speed. (DIS) • In some cases, it is hybridized with other algorithms to look for local optima. (DIS)	• To estimate the location of the permanent magnets inside the human body by measuring the magnetic field; the total error is estimated by minimizing the sum of the three errors between the measured values and the theoretical values of the magnetic field [79].

4. Conclusions

This paper provides an extensive study related to the optimization methods used in the field of localization. In the first part of the article, a light introduction to the optimization problem was reported, providing a general mathematical framework for the discipline. Besides the SO problem, some concepts about the MO problem have been introduced as well, and they aim to provide a clear and complete view of the basis of the subject.

In the second part of the article, extensive research about some interesting algorithms used in localization has been reported, providing for each of them the concept of the algorithm, the related mathematics model, and an example of application. Both types of algorithms for local and global optimization have been described, and it can be noted that metaheuristic algorithms, such as GA, DE, or PSO, are the most used in the field of localization for optimization problems due to their capabilities to manage non-convex and non-linear problems and provide satisfactory results. Moreover, the GA (as well as its modified versions, such as the QGA) is the most versatile optimization tool that allows for different types of problems such as continuous, discrete, or mixed-integer types. However, the standard GA has a slow speed of convergence. In contrast, PSO has a higher speed, and by observing Table 1, it is the most used among the investigated methods. Furthermore, PSO was shown to be the most prone method that allows for hybridization with other types, such as DEPSO or SWIRL; the modified methods improved the performance in solving specific problems.

In general, by comparing metaheuristic algorithms to gradient-based algorithms (that are especially used for local optimum), it is noted that stochastic algorithms do not need any Jacobian (or Hessian) matrix inversion, demonstrating that they are very flexible and suitable optimization tools for many optimization problems. On the other hand, they need to tune the parameters for each specific solved problem.

In the end, it is highlighted that none of the investigated methods is guaranteed to provide a real global optimum.

Author Contributions: Conceptualization, M.S., P.S. and A.O.; funding acquisition, P.S. and A.O.; project administration, P.S. and A.O.; resources, M.S., P.S., J.S. and A.O.; supervision, P.S., J.S. and A.O.; writing—original draft, M.S.; writing—review and editing, P.S., J.S. and A.O. All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported by the National Research, Development, and Innovation Fund of Hungary through project no. 142790 under the FK_22 funding scheme.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Talbi, E.G. *Metaheuristics: From Design to Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
2. Gogna, A.; Tayal, A. Metaheuristics: Review and Application. *J. Exp. Theor. Artif. Intell.* **2013**, *25*, 503–526. [CrossRef]
3. Gilli, M.; Schumann, E. Heuristic Optimisation in Financial Modelling. *Ann. Oper. Res.* **2012**, *193*, 129–158. [CrossRef]
4. Agrawal, N.; Rangaiah, G.P.; Ray, A.K.; Gupta, S.K. Multi-Objective Optimization of the Operation of an Industrial Low-Density Polyethylene Tubular Reactor Using Genetic Algorithm and Its Jumping Gene Adaptations. *Ind. Eng. Chem. Res.* **2006**, *45*, 3182–3199. [CrossRef]
5. Kulkarni, N.K.; Patekar, S.; Bhoskar, T.; Kulkarni, O.; Kakandikar, G.M.; Nandedkar, V.M. Particle Swarm Optimization Applications to Mechanical Engineering—A Review. In *Materials Today: Proceedings*; Elsevier: Amsterdam, The Netherlands, 2015; Volume 2.
6. Jordehi, A.R. Particle Swarm Optimisation (PSO) for Allocation of FACTS Devices in Electric Transmission Systems: A Review. *Renew. Sustain. Energy Rev.* **2015**, *52*, 1260–1267. [CrossRef]
7. Lin, M.H. An Optimal Workload-Based Data Allocation Approach for Multidisk Databases. *Data Knowl. Eng.* **2009**, *68*, 499–508. [CrossRef]
8. Odry, A.; Kecskes, I.; Csik, D.; Hashim, H.A.; Sarcevic, P. Adaptive Gradient-Descent Extended Kalman Filter for Pose Estimation of Mobile Robots with Sparse Reference Signals. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Kyoto, Japan, 23–27 October 2022; Volume 2022.
9. Grottke, J.; Blankenbach, J. Evolutionary Optimization Strategy for Indoor Position Estimation Using Smartphones. *Electronics* **2021**, *10*, 618. [CrossRef]
10. Yousefi, M.; Nejat Pishkenari, H.; Alasty, A. A Fast and Robust Magnetic Localization Technique Based on Elimination of the Orientation Variables from the Optimization. *IEEE Sens. J.* **2021**, *21*, 21885–21892. [CrossRef]
11. Yu, B.; Zhu, H.; Xue, D.; Xu, L.; Zhang, S.; Li, B. A Dead Reckoning Calibration Scheme Based on Optimization with an Adaptive Quantum-Inspired Evolutionary Algorithm for Vehicle Self-Localization. *Entropy* **2022**, *24*, 1128. [CrossRef]
12. Lei, J.; Fang, H.; Zhu, Y.; Chen, Z.; Wang, X.; Xue, B.; Yang, M.; Wang, N. GPR Detection Localization of Underground Structures Based on Deep Learning and Reverse Time Migration. *NDT E Int.* **2024**, *143*, 103043. [CrossRef]
13. Su, Y.; Wang, J.; Li, D.; Wang, X.; Hu, L.; Yao, Y.; Kang, Y. End-to-End Deep Learning Model for Underground Utilities Localization Using GPR. *Autom Constr* **2023**, *149*, 104776. [CrossRef]
14. Boyd, S.P.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
15. Venter, G. Review of Optimization Techniques. In *Encyclopedia of Aerospace Engineering*; John Wiley & Sons, Ltd.: Chichester, UK, 2010.

16. Esmaeilzadeh Azar, F.; Perrier, M.; Srinivasan, B. A Global Optimization Method Based on Multi-Unit Extremum-Seeking for Scalar Nonlinear Systems. *Comput. Chem. Eng.* **2011**, *35*, 456–463. [CrossRef]
17. Tawfik, A.S.; Badr, A.A.; Abdel-Rahman, I.F. Comparative Optimizer Rank and Score: A Modern Approach for Performance Analysis of Optimization Techniques. *Expert Syst. Appl.* **2016**, *45*, 118–130. [CrossRef]
18. Lin, M.H.; Tsai, J.F.; Yu, C.S. A Review of Deterministic Optimization Methods in Engineering and Management. *Math. Probl. Eng.* **2012**, *2012*, 756023. [CrossRef]
19. Engelbrecht, A.P. *Computational Intelligence: An Introduction*, 2nd ed.; John Wiley & Sons Ltd.: Hoboken, NJ, USA, 2007.
20. Arora, J.S. *Introduction to Optimum Design*; Elsevier: Amsterdam, The Netherlands, 2004.
21. Cox, S.E.; Haftka, R.T.; Baker, C.A.; Grossman, B.; Mason, W.H.; Watson, L.T. A Comparison of Global Optimization Methods for the Design of a High-Speed Civil Transport. *J. Glob. Optim.* **2001**, *21*, 415–432. [CrossRef]
22. Kvasov, D.E.; Mukhametzhanov, M.S. Metaheuristic vs. Deterministic Global Optimization Algorithms: The Univariate Case. *Appl. Math. Comput.* **2018**, *318*, 245–259. [CrossRef]
23. Abdel-Basset, M.; Abdel-Fatah, L.; Sangaiah, A.K. Metaheuristic Algorithms: A Comprehensive Review. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*; Academic Press: Cambridge, MA, USA, 2018.
24. Agrawal, P.; Abutarboush, H.F.; Ganesh, T.; Mohamed, A.W. Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009–2019). *IEEE Access* **2021**, *9*, 3056407. [CrossRef]
25. Francisco, M.; Revollar, S.; Vega, P.; Lamanna, R. A Comparative Study of Deterministic and Stochastic Optimization Methods for Integrated Design of Processes. In Proceedings of the IFAC Proceedings Volumes (IFAC-PapersOnline), 16th Triennial World Congress, Prague, Czech Republic, 3–8 July 2005; Volume 38.
26. Jones, D.R.; Perttunen, C.D.; Stuckman, B.E. Lipschitzian Optimization without the Lipschitz Constant. *J. Optim. Theory Appl.* **1993**, *79*, 157–181. [CrossRef]
27. Deb, K. Multi-Objective Optimisation Using Evolutionary Algorithms: An Introduction. In *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*; Springer: London, UK, 2011.
28. De Weck, O. Multiobjective Optimization: History and Promise. In Proceedings of the The Third China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems, Kanazawa, Japan, 30 October–2 November 2004. Invited Keynote Paper, GL2-2.
29. Gunantara, N. A Review of Multi-Objective Optimization: Methods and Its Applications. *Cogent Eng.* **2018**, *5*, 1502242. [CrossRef]
30. Marler, R.T.; Arora, J.S. Survey of Multi-Objective Optimization Methods for Engineering. *Struct. Multidiscip. Optim.* **2004**, *26*, 369–395. [CrossRef]
31. Gunantara, N.; Sastra, N.P.; Hendrantoro, G. Cooperative Diversity Pathsselection Protocol Withmulti-Objective Criterion in Wireless Ad-Hoc Networks. *Int. J. Appl. Eng. Res.* **2014**, *9*, 22395–22407.
32. Athan, T.W.; Papalambros, P.Y. A Note on Weighted Criteria Methods for Compromise Solutions in Multi-Objective Optimization. *Eng. Optim.* **1996**, *27*, 155–176. [CrossRef]
33. Gerasimov, E.N.; Repko, V.N. Multicriterial Optimization. *Sov. Appl. Mech.* **1978**, *14*, 1179–1184. [CrossRef]
34. Meza Juan, C. Newton's Method. *Wiley Interdiscip. Rev. Comput. Stat.* **2011**, *3*, 75–78. [CrossRef]
35. Lee, J.D.; Simchowitz, M.; Jordan, M.I.; Recht, B. Gradient Descent Only Converges to Minimizers. *J. Mach. Learn. Res.* **2016**, *49*, 1246–1257. [CrossRef]
36. Smith, A.E.; Coit, D.W.; Baeck, T.; Fogel, D.; Michalewicz, Z. Penalty Functions. *Handb. Evol. Comput.* **1997**, *97*, C5.
37. Wang, X.; Yan, L.; Zhang, Q. Research on the Application of Gradient Descent Algorithm in Machine Learning. In Proceedings of the 2021 International Conference on Computer Network, Electronic and Automation, ICCNEA 2021, Xi'an, China, 24–26 September 2021.
38. Chang, X.W.; Yang, X.; Zhou, T. MLAMBDA: A Modified LAMBDA Method for Integer Least-Squares Estimation. *J. Geod.* **2005**, *79*, 552–565. [CrossRef]
39. Cheng, Q.; Chen, W.; Sun, R.; Wang, J.; Weng, D. RANSAC-Based Instantaneous Real-Time Kinematic Positioning with GNSS Triple-Frequency Signals in Urban Areas. *J. Geod.* **2024**, *98*, 24. [CrossRef]
40. Transtrum, M.K.; Sethna, J.P. Improvements to the Levenberg-Marquardt Algorithm for Nonlinear Least-Squares Minimization. *arXiv* **2012**, arXiv:1201.5885.
41. Levenberg, K. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Q. Appl. Math.* **1944**, *2*, 164–168. [CrossRef]
42. Marquardt, D.W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Ind. Appl. Math.* **1963**, *11*, 431–441. [CrossRef]
43. Hoffman Laurence, D.; Gerald, L. *Bradley Calculus for Business, Economics, and the Social and Life Sciences*, 10th ed.; McGraw-Hill Inc.: New York, NY, USA, 2010.
44. Kalman, D. Leveling with Lagrange: An Alternate View of Constrained Optimization. *Math. Mag.* **2009**, *82*, 186–196. [CrossRef]
45. Parkinson, A.R.; Balling, R.J.; Hedengren, J.D. *Optimization Methods for Engineering Design Applications and Theory*; Brigham Young University: Provo, UT, USA, 1972; Volume 94.
46. Ghojogh, B.; Ghodsi, A.; Karray, F.; Crowley, M. KKT Conditions, First-Order and Second-Order Optimization, and Distributed Optimization: Tutorial and Survey. *arXiv* **2021**, arXiv:2110.01858.

47. Xue, K.; Li, J.; Xiao, N.; Liu, J.; Ji, X.; Qian, H. Improving the Robot Localization Accuracy Using Range-Only Data: An Optimization Approach. In Proceedings of the 2021 6th IEEE International Conference on Advanced Robotics and Mechatronics, ICARM 2021, Chongqing, China, 3–5 July 2021.
48. Katoch, S.; Chauhan, S.S.; Kumar, V. A Review on Genetic Algorithm: Past, Present, and Future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
49. Kaya, M. The Effects of a New Selection Operator on the Performance of a Genetic Algorithm. *Appl. Math. Comput.* **2011**, *217*, 7669–7678. [CrossRef]
50. Liang, Y.; Leung, K.S. Genetic Algorithm with Adaptive Elitist-Population Strategies for Multimodal Function Optimization. *Appl. Soft Comput. J.* **2011**, *11*, 2017–2034. [CrossRef]
51. Umbarkar, A.J.; Sheth, P.D. Crossover Operators in Genetic Algorithms: A Review. *ICTACT J. Soft Comput.* **2015**, *6*, 1083–1092. [CrossRef]
52. Lambora, A.; Gupta, K.; Chopra, K. Genetic Algorithm—A Literature Review. In Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COMITCon 2019, Faridabad, India, 14–16 February 2019.
53. Lim, S.M.; Sultan, A.B.M.; Sulaiman, M.N.; Mustapha, A.; Leong, K.Y. Crossover and Mutation Operators of Genetic Algorithms. *Int. J. Mach. Learn. Comput.* **2017**, *7*, 9–12. [CrossRef]
54. Frenzel, J.F. Genetic Algorithms. *IEEE Potentials* **1993**, *12*, 21–24. [CrossRef]
55. Han, K.H.; Kim, J.H. Genetic Quantum Algorithm and Its Application to Combinatorial Optimization Problem. In Proceedings of the IEEE Conference on Evolutionary Computation, ICEC, La Jolla, CA, USA, 16–19 July 2000; Volume 2.
56. Deng, W.; Shang, S.; Cai, X.; Zhao, H.; Song, Y.; Xu, J. An Improved Differential Evolution Algorithm and Its Application in Optimization Problem. *Soft Comput.* **2021**, *25*, 5277–5298. [CrossRef]
57. Bhowmik, P.; Das, S.; Konar, A.; Das, S.; Nagar, A.K. A New Differential Evolution with Improved Mutation Strategy. In Proceedings of the 2010 IEEE World Congress on Computational Intelligence, WCCI 2010—2010 IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18–23 July 2010.
58. Lin, C.; Qing, A.; Feng, Q. A Comparative Study of Crossover in Differential Evolution. *J. Heuristics* **2011**, *17*, 675–703. [CrossRef]
59. Wu, Y.; Liu, X.; Guan, W.; Chen, B.; Chen, X.; Xie, C. High-Speed 3D Indoor Localization System Based on Visible Light Communication Using Differential Evolution Algorithm. *Opt. Commun.* **2018**, *424*, 177–189. [CrossRef]
60. Sun, M.; Wang, Y.; Joseph, W.; Plets, D. Indoor Localization Using Mind Evolutionary Algorithm-Based Geomagnetic Positioning and Smartphone IMU Sensors. *IEEE Sens. J.* **2022**, *22*, 3155817. [CrossRef]
61. Jie, J.; Zeng, J.; Ren, Y. Improved Mind Evolutionary Computation for Optimizations. In Proceedings of the World Congress on Intelligent Control and Automation (WCICA), Hangzhou, China, 15–19 June 2004; Volume 3.
62. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; IEEE: Piscataway, NJ, USA, 1995; pp. 1942–1948.
63. Li, N.; Chen, J.; Yuan, Y.; Tian, X.; Han, Y.; Xia, M. A Wi-Fi Indoor Localization Strategy Using Particle Swarm Optimization Based Artificial Neural Networks. *Int. J. Distrib. Sens. Netw.* **2016**, *2016*, 4583147. [CrossRef]
64. Zhang, Y.; Hu, H.; Fu, W.; Jiang, H. Particle Swarm Optimization-Based Minimum Residual Algorithm for Mobile Robot Localization in Indoor Environment. *Int. J. Adv. Robot. Syst.* **2017**, *14*, 1729881417729277. [CrossRef]
65. Havangi, R. Mobile Robot Localization Based on PSO Estimator. *Asian J. Control* **2019**, *21*, 2167–2178. [CrossRef]
66. Maeda, Y.; Matsushita, N. Simultaneous Perturbation Particle Swarm Optimization Using FPGA. In Proceedings of the IEEE International Conference on Neural Networks—Conference Proceedings, Orlando, FL, USA, 12–17 August 2007.
67. Huo, J.; Ma, L.; Yu, Y.; Wang, J. Hybrid Algorithm Based Mobile Robot Localization Using de and PSO. In Proceedings of the Chinese Control Conference, CCC, Xi'an, China, 26–28 July 2013.
68. Wang, Y.; Wang, P.; Zhang, J.; Cui, Z.; Cai, X.; Zhang, W.; Chen, J. A Novel Bat Algorithm with Multiple Strategies Coupling for Numerical Optimization. *Mathematics* **2019**, *7*, 135. [CrossRef]
69. Neto, W.A.; Pinto, M.F.; Marcato, A.L.M.; da Silva, I.C.; Fernandes, D.d.A. Mobile Robot Localization Based on the Novel Leader-Based Bat Algorithm. *J. Control Autom. Electr. Syst.* **2019**, *30*, 337–346. [CrossRef]
70. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*; Technical Report TR06; Erciyes University: Kayseri, Türkiye, 2005.
71. Zhao, Y.; Yan, Q.; Yang, Z.; Yu, X.; Jia, B. A Novel Artificial Bee Colony Algorithm for Structural Damage Detection. *Adv. Civ. Eng.* **2020**, *2020*, 3743089. [CrossRef]
72. Kulkarni, V.R.; Desai, V.; Kulkarni, R.V. Multistage Localization in Wireless Sensor Networks Using Artificial Bee Colony Algorithm. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, 6–9 December 2016.
73. Amine, K. Multiobjective Simulated Annealing: Principles and Algorithm Variants. *Adv. Oper. Res.* **2019**, *2019*, 8134674. [CrossRef]
74. Kannan, A.A.; Mao, G.; Vucetic, B. Simulated Annealing Based Localization in Wireless Sensor Network. In Proceedings of the Proceedings—Conference on Local Computer Networks, LCN, Sydney, NSW, Australia, 17 November 2005; Volume 2005.
75. Blum, C. Ant Colony Optimization: Introduction and Recent Trends. *Phys. Life Rev.* **2005**, *2*, 353–373. [CrossRef]
76. Stützle, T.; Dorigo, M. ACO Algorithms for the Travelling Salesman Problem. *Evol. Algorithms Eng. Comput. Sci.* **1999**, *4*, 163–183.
77. Mali, S. Mobile Robot Localization Using Multi-Objective Optimization. *Int. J. Latest Technol. Eng. Manag. Appl. Sci.* **2015**, *4*, 21–25.

78. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
79. Lv, B.; Qin, Y.; Dai, H.; Su, S. Improving Localization Success Rate of Three Magnetic Targets Using Individual Memory-Based WO-LM Algorithm. *IEEE Sens. J.* **2021**, *21*, 3101299. [CrossRef]