computers

MDPI

Article

Development of a Test-Bench for Evaluating the Embedded Implementation of the Improved Elephant Herding Optimization Algorithm Applied to Energy-Based Acoustic Localization

Sérgio D. Correia [1,2,*], João Fé [1,2], Slavisa Tomic [1] and Marko Beko [1,3]

1 COPELABS, Universidade Lusófona de Humanidades e Tecnologias, Campo Grande 376, 1749-024 Lisboa, Portugal; 19389@ipportalegre.pt (J.F.); slavisa.tomic@ulusofona.pt (S.T.); beko.marko@ulusofona.pt (M.B.)
2 VALORIZA–Research Centre for Endogenous Resource Valorization, Instituto Politécnico de Portalegre, Campus Politécnico n.10, 7300-555 Portalegre, Portugal
3 Instituto de Telecomunicações, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisboa, Portugal
* Correspondence: scorreia@ipportalegre.pt

Received: 29 September 2020; Accepted: 30 October 2020; Published: 3 November 2020

Abstract: The present work addresses the development of a test-bench for the embedded implementation, validity, and testing of the recently proposed Improved Elephant Herding Optimization (*iEHO*) algorithm, applied to the acoustic localization problem. The implemented methodology aims to corroborate the feasibility of applying *iEHO* in real-time applications on low complexity and low power devices, where three different electronic modules are used and tested. Swarm-based metaheuristic methods are usually examined by employing high-level languages on centralized computers, demonstrating their capability in finding global or good local solutions. This work considers *iEHO* implementation in C-language running on an embedded processor. Several random scenarios are generated, uploaded, and processed by the embedded processor to demonstrate the algorithm's effectiveness and the test-bench usability, low complexity, and high reliability. On the one hand, the results obtained in our test-bench are concordant with the high-level implementations using MatLab® in terms of accuracy. On the other hand, concerning the processing time and as a breakthrough, the results obtained over the test-bench allow to demonstrate a high suitability of the embedded *iEHO* implementation for real-time applications due to its low latency.

Keywords: arduino programming; ESP32 platform; embedded programming; energy-based acoustic localization; Internet of Things; software/hardware testing; swarm optimization; wireless sensor networks

1. Introduction

Embedded systems are widely spread in our daily life. They are a vital component of larger structures such as wireless sensors networks (WSNs) [1], Internet of Things (IoT) [2], automotive electronics [3], home automation [4], energy management [5–7], noise monitoring [8,9], autonomous vehicles [10,11], among several others. Elecia White defines an embedded system as " *a computerized system that is purpose-built for its application*" [12]. Extending this concept, today's modern systems are based on microcontrollers with integrated memory (volatile and nonvolatile), digital inputs/outputs, and application-specific peripherals. Features such as analog-to-digital or digital-to-analog conversion, connectivity (802.15.4, 802.11b/g/n and LORA), or power management make them well suited for data harvesting, sensing, and actuating the physical environment as edge devices [13]. Bearing in mind that these types of equipment produce knowledge in the context of sensor networks on IoT platforms, the integration of new sources of information (based on more efficient algorithms) should be considered an essential direction of research. In this context, the role of power management, coupled with the efficient implementation of integrated algorithms, plays an important role in achieving significantly longer battery life of embedded systems [14,15]. The embedded system design has several constraints, namely its cost, energy consumption, performance, processing time, flexibility, time, or sustainability [12]. When it comes to WSNs, features such as energy consumption and performance are of crucial importance. Wireless sensor nodes (or simply sensors) typically aim to retain their energy level for an extended time, monitor required data, and send it to a base station located in a remote place. When the sensors are distributed in geographically isolated areas, they are deployed with batteries that cannot be replaced, or their replacement comes with high-cost [16].

Keeping the same line of reasoning, knowing the location of sensors plays an important role in several fields and applications, namely acoustic localization. Some examples of application can be found in shooter localization in urban terrain [17], smart surveillance [18], wildlife [19] or robotics [20]. The state-of-the-art comprises some platforms that have been proposed in the context of an acoustic acquisition. Nonetheless, localization processing is mostly done offline, on a central computing unit, or when considered locally, on complex and expensive devices such as field-programmable gate arrays (FPGAs) [9,10]. Some proposals of hardware platforms that have been presented in the context of acoustic acquisition can be found in [21] but with the need for complex mathematical processing. The platform described in [22] considers acoustic processing stage (detection, classification, and analysis) on a sensor itself, creating the need for complex sensors architectures. In [23], an application to support Ambient Assisted Living based on acoustic events was proposed for indoor environments. However, it is confined to a single place of application, again with the need for complex hardware architectures.

One the other hand, swarm-based algorithms have demonstrated their potential in solving complex engineering problems [24,25]. The methodology was originally proposed to mimic the social behavior of a bird flock [26], but nowadays, it comprises a broad group of computational methods known as swarm intelligence. In the particular case of acoustic energy-based source localization, an approach based on Elephant Herding Optimization [27] was initially proposed in [28] and improved in [29]. Earlier results demonstrated the applicability of the methodology, as well as a significantly reduced complexity of implementation in comparison with other approaches available in the scientific literature. As such, the present work focuses on developing a test-bench to evaluate the performance, validate, and test an embedded implementation of *iEHO* algorithm [29] for performing localization of acoustic events. In addition to the challenge of putting into operation the mentioned algorithms on processors with low computational resources that use low level programming languages, the present works aim to develop a reliable and simple test-bench to perform a wide range of test conditions. Instead of using well established protocols such as JTAG or ARM Serial Wire Debug [30], that would imply specific tools; the setup and observation data are supplied to the processor through a serial link, and the same goes for the obtained results, storing only a minimal memory overhead.

To the best of the authors' knowledge, no localization algorithms with the use of swarm-based optimization have been incorporated directly on the sensors themselves, where maximizing performance and minimizing computational resource are the two main objectives. Therefore, standard inline debug strategies do not suffice to validate the overall performance of this type of implementation. The proposed work assumes the following claims: (1) to design and implement an embedded swarm-based methodology for energy-based acoustic localization; (2) to design a simple integrated test-bench to validate the implementation as a proof of concept; (3) to consider a sufficient number of hardware platforms to generalize their usability; (4) to consider a wide set of simulation conditions to extrapolate the effectiveness of the embedded implementation; (5) to validate the embedded results with high level languages such as MatLab®.

The current work falls in the field of computation, software testing, and applications of sensor networks signal processing, far from computer simulations with well established software platforms, integrating complex libraries and graphical interfaces. The obtained results will thus serve as a proof of concept given that they are achieved through real-life platforms for product development. All procedures will thus be embedded, namely the data sources and performance metrics calculation.

The remaining paper is organized as follows. Section 2 provides a contextualization with related embedded testing frameworks and protocols. Section 3 introduces the problem of interest, based on presenting its theoretical and technical foundations. Section 4 presents the test-bench developed, the methodology used to address the problem and the experimental setup employed in this work. Section 5 presents and discusses the obtained results, and Section 6 concludes the paper and presents future extensions of the present work.

2. Related Work

Embedded software testing lifecycle has the goal of finding defects and evaluating the performance of implemented algorithms. To achieve these goals, a test procedure containing activities and test data must be planned carefully in order to specify what should be evaluated and how. Therefore, some means of communication with the embedded processor should be designed. Alternatively, already existing communication channel could be used [31]. To provide standardized approaches, the "IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture", also known as JTAG, specifies a debug interface that can be included in any Integrated Circuit (IC) [32]. The standard is widely adopted by IC manufacturers, and it specifies a four-pin test access port and an optional test reset pin as a debug interface for a chip. Although single wire interfaces were proposed in the literature [33,34], they are only applicable on System on Chip ICs [35,36] or field-programmable gate array designs [37]. When available, a serial communication port is one of the most flexible solutions, deploying only a two-pin full-duplex interface.

When considering the application level, debugging frameworks are usually used for debugging, verifying, and classifying some performance or behaviors. Implementation examples can be found regarding embedded wireless sensors [38], web-based sensors [39], Internet of Things devices [40], power quality sensor nodes [41], or cyber-physical automation systems [42]. Application-specific debugging is known for the decrease in development cost and time of faulty stage detection [43]. The present work relies on embedded acoustic sensor nodes to validate and evaluate a swarm-based algorithm to perform acoustic localization.

3. Theoretical Background

This section is divided into two parts. The first part introduces the acoustic measurement model and formalizes the localization problem, while the second part summarizes the swarm-based algorithm of interest here, i.e., the *iEHO* method. As such, the present section intends to provide the necessary theoretical fundamentals to specify the test-bench environment and its components.

3.1. Acoustic Problem Formulation

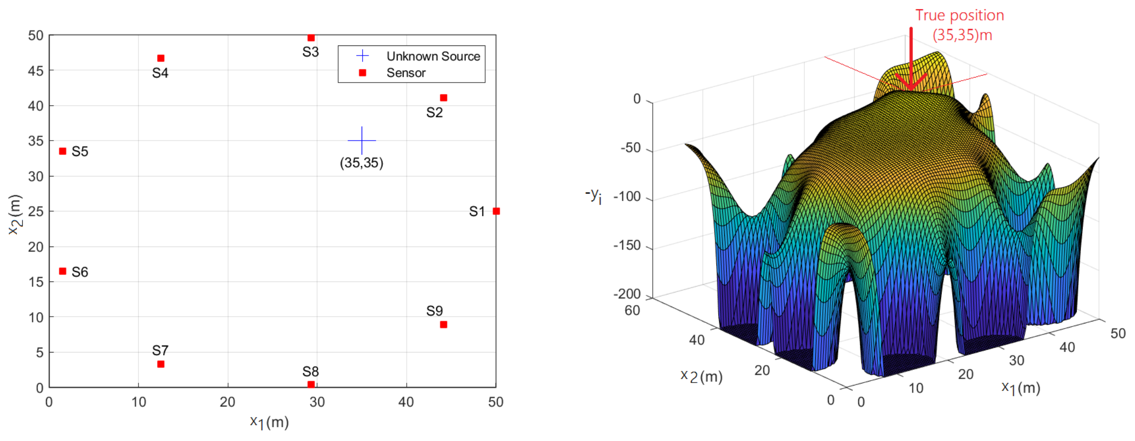
The location of an acoustic source consists of analyzing a q-dimensional (where q = 2 or 3) sensor network, composed of N sensors and one acoustic source. The true (but unknown) location of the source is denoted by $\mathbf{x}$ and the known location of the i_{th} sensor by $\mathbf{s}_i$, where $i = 1, \ldots, N$. The location of the source is determined by employing the acoustic energy measurements acquired by sensors. The relationship between the acoustic energy and distance is modeled through a decay model [44,45] where it is considered that the sound propagates in free space, the sound is omnidirectional, the propagation medium (air) is homogeneous, and there is no sound reverberation [44]. Each sensor performs M noisy measurements, within a time frame $T = M/f_s$, where f_s is the sampling frequency. Thus, when considering the average of the energy signatures over the time window, according to [44,45], the received signal at the i_{th} sensor can be modeled as:

$$y_i = \frac{g_i P}{||\mathbf{x} - \mathbf{s}_i||^{\beta_L}} + \nu_i, \qquad \text{for } i = 1, \ldots, N. \tag{1}$$

where g_i is the sensor gain, P is the transmitted power, $\nu_i \sim \mathcal{N}(0, \sigma^2_{\nu_i})$ represents the measurement noise, assumed to follow a Gaussian distribution with zero mean and variance $\sigma^2_{\nu_i}$, and β_L is the path loss exponent. For the sake of simplicity and without loss of generality, the measurement noise is considered equal for all links and thus, $\sigma^2_{\nu_1} = \sigma^2_{\nu_2} = \ldots = \sigma^2_{\nu_N} = \sigma^2_{\nu}$. The value of β_L is typically considered within the interval $[2, 4]$ [46]. In this work, we consider $\beta_L = 2$, since we consider signal propagation in free space, without reflections or reverberations [46]. Incorporating all observations from the multiple sensors (1), the maximum likelihood (ML) estimator of $\mathbf{x}$ can be formulated as [44,45]:

$$\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}} \sum_{i=1}^{N} \left(y_i - \frac{g_i P}{||\mathbf{x} - \mathbf{s}_i||^2} \right)^2. \tag{2}$$

The mathematical expression (2) is highly nonconvex, has singularities in each sensor coordinate, several suboptimal solutions and saddle regions. In Figure 1, a fixed setup was considered with the purpose of highlighting the mentioned characteristics. The setup includes 9 sensors uniformly distributed around the center of the search space and an unknown source at $(35, 35)$ m (Figure 1a). The observations were generated considering $\sigma^2_{\nu} = -50$ dB, a transmitted power of $P = 5$, and $g_i = 1$, for $i = 1, \ldots, N$. Figure 1b depicts the symmetric of problem (2) with the goal of improving the visualization and interpretation of the cost function. Considering that the singularities at the coordinates of the sensors would imply a value of infinity, a limit value of -200 was represented for the surface plot as it can be seen from Figure 1b, where the above-mentioned features are well emphasized.



(**a**) (**b**)

Figure 1. Graphic Representation of ML Model. (**a**) Sensors and Source Setup (**b**) Surface representation.

Several solutions for tackling (2) have been proposed in the literature, mostly using deterministic approaches. The work in [47], in which a closed-form solution was proposed, exhibits good performance for low noise power but suffers considerable degradation for higher levels of noise power. A weighted least squares method was presented in [48,49]. Although its computational load is low, its performance degrades in noisy environments, due to the approximations necessary to achieve a linear estimator. Semidefinite relaxation was proposed in [50,51], where good performance was obtained in terms of localization accuracy, even in noisy environments, but with a high computational cost. The problem of high computational cost was partially by applying Second-Order Cone Programming [52], but still, the reduced computational complexity is not suitable for the implementation in real-time scenarios, at least for the time being. Authors in [28,53] adapted the classical Elephant Herding Optimization (*EHO*) method [27] to the problem of energy-based source localization. This implementation of a swarm-based metaheuristic algorithm allowed the possibility to use significantly less-complex algorithms in terms of computation, which achieve a location accuracy similar to its deterministic counterparts.

Apart from the energy model discussed, time difference of arrival has also been employed to solve the source localization problem. The approach consist of estimating the time delay between a pair of microphones. For that purpose, the work presented in [54] used a triangular array consisting of 3-microphones to determine the source location from observations of a mobile robot. Additionally, in [55], an evaluation of real-time sound localization approaches are compared using an 8-microphone array. Similar architectures were presented in [56,57], where the common need of an array of microphones can be considered as a drawback for practical applications [58,59]. As such, the present work considers the acoustic energy-based model of [44].

3.2. Swarm Optimization

Swarm optimization falls within the set of algorithms for global optimization inspired by nature behavior. A group of initial candidate solutions is generated and updated based on a particular swarm behavior in an iterative process. Each new generation is produced by removing less desired solutions and introducing small random changes based on the behavior in question [60]. Regarding *EHO*, the algorithm emulates the herding behavior of elephants in a group. In nature, elephants belonging to different clans live together under the leadership of a matriarch, and the male elephants leave their family group when they reach adulthood [27]. Thanks to the low number of control parameters and its simple implementation, the *EHO* algorithm has been successfully applied to several scientific fields. Applications range from drone placement [61,62], power flow management [63], image encryption [64], proportional-integral-derivative control [65,66], or localization in WSNs [67,68].

The mathematical models used are based on simple algebraic expressions, allowing an initial population to approach the real solution. While the initial population is usually randomly generated, authors in [29] developed a new strategy, that by exploiting the particularities of the problem at hand, allowed to start the search procedure around the region in which it is most likely to find an suboptimal solution. The main finding reported in [29] was the significant reduction of the number of population generations. As such, it is expected that the implementation of such methodology significantly reduces the processing needs for finding a feasible solution, i.e., the desired location with high accuracy, simultaneously guaranteeing low latency to a more generic architecture. Figure 2 shows the algorithm's flowchart, where usual swarm-based method features such as sorting, updating, performing elitism, and evaluating can be found. In the case of *iEHO* [29], the population is initialized based on distance estimation, the stopping criteria monitors the cost function progress, and the clan operator performs a local search based on a discrete gradient descent method.

The present work considers an embedded implementation of *iEHO* [29], using low cost processors, available as development boards, compatible with the Arduino® Integrated Development Environment (IDE). The implementation of a problem-specific test-bench will allow collecting information for generating performance metrics and evaluating the obtained results. The *iEHO* algorithm is considered here at the expense of other swarm-based ones, due to its reduced number of tuning parameters and its proven effectiveness in the specific problem under study [28,29,53]. In addition, its simplicity contrasts the need for matrix calculations and other essential complex mathematical calculation in the deterministic context [69,70]. In order to get a better understanding and appreciation of the obtained results, the standard *EHO* is also implemented and evaluated for comparison. Thus, it will be possible to evaluate the modifications on *iEHO*, such as the obtained error and the computational time with regard to the original *EHO*.

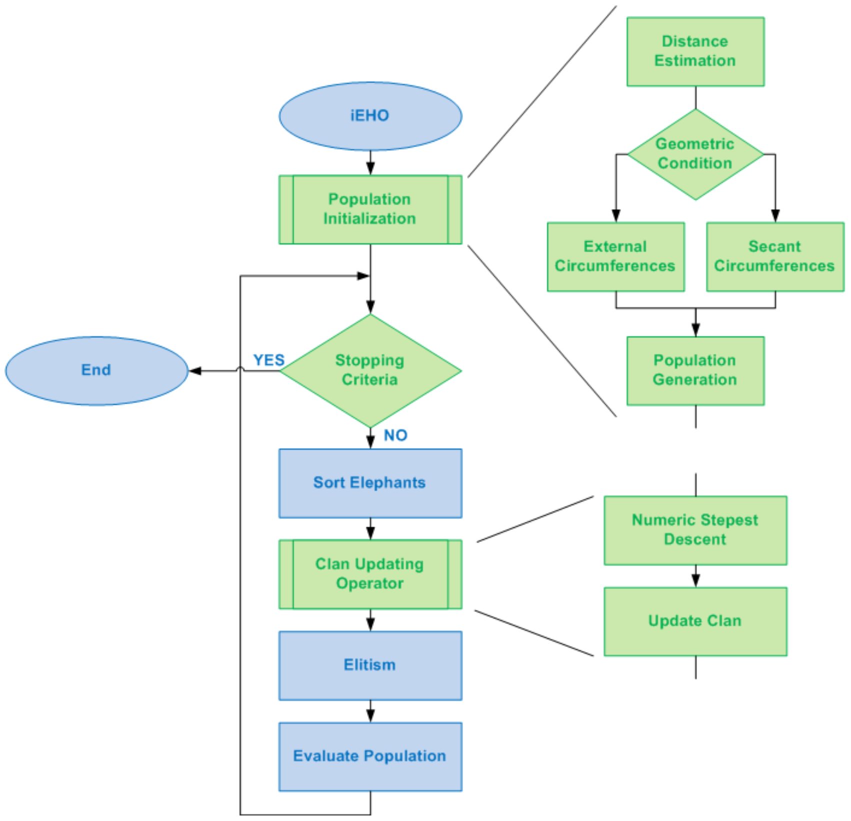


Figure 2. *iEHO* flowchart.

4. Methodology and Experimental Setup

Embedded software test design techniques can be classified between *White-Box* or *Black-Box*, whenever the explicit knowledge of the implementation details is under scrutiny or not [13]. In a *Black-Box* test, the system is fed with some inputs, and the resulting outputs are analyzed as to whether it complies with the expected behavior. On the other hand, a *White-Box* technique would be based on the knowledge of the firmware internal structure [13]. Given our primary focus and the fact that, since the major concern of our platform is to determine acoustic events positions, a *Black-Box* approach is considered here (Figure 3).

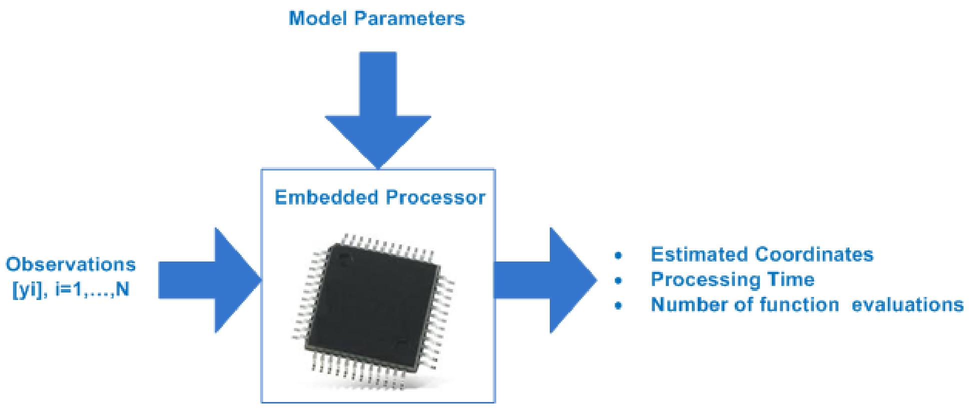


Figure 3. *Black Box* test-bench.

The adopted procedure consists of generating Monte Carlo, M_c, random observations, corrupted by white Gaussian noise, ν, of variance σ_ν^2, i.e., $\nu \sim \mathcal{N}(0, \sigma_\nu^2)$. Those observations are the only input provided to the embedded processor in the test-bench, which will estimate the unknown position by applying *iEHO* algorithm for solving the problem given by Equation (2). Besides the estimated position, to evaluate the implementation performance, additional metrics will be gathered, namely the processing time and the number of evaluations of the cost function. It should be noted that *iEHO* uses a stopping criteria based on the cost function value between successive iterations (Δf), in order to possibly stop the method before it attains a predefined maximum number of evaluations [29]. As such, some correlation between the number of function evaluations and the processing time is expected.

The tests are individualized and run on three different platforms. The first one comprises a NodeMCU module, constituted by a L106 32-bit RISC core processor from *Tensilica Xtensa* (ip.cadence.com), running at 80 MHz, with 4 MBytes of RAM and 128 KBytes of flash memory, embedded on an *Espressif Systems* (www.espressif.com) ESP8266 microchip. The second, an Arduino Due, is based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It runs at 84 MHz clock, with 96 KBytes of RAM and 512 KBytes of flash memory. Finally, the third module consists of a LX6 32-bit RISC core processor from *Tensilica Xtensa* (also known as ESP32), featuring 512 KBytes of RAM and 4 MBytes of flash memory. Both NodeMCU and ESP32 platforms incorporates a native 2.4 Gh WIFI module implementing norms 802.11 b/g/n with an internal TCP/IP (IPv4) stack. All platforms features several GPIO (general-purpose input/output) pins, analog-to-digital converters, and multiple communication interfaces. Present on all modules are different low power modes with small energy consumptions, making them very suitable and affordable modules for IoT projects. The modules present themselves with an internal power regulator and an USB chipset, providing a flexible connection to a personal computer for supplying and retrieving the test data information.

Table 1 resumes all individual features of the modules used in the test-bench. The described hardware features two similar architectures, but with different clock speeds, with the purpose of highlighting the expected reduction on the processing time. Since Tensilica Xtensa employs a 32-bit proprietary RISC CPU, we choose a third processor incorporating the well known ARM Cortex-M3 architecture, allowing the extrapolation of the results since it is present on several semiconductor manufactures microcontrollers such as STMicroelectronics, NXP Semiconductors, Microchip (ATMEL), Broadcom, Texas Instrumrnts or Silicon Labs.

Table 1. Comparison of modules' specifications.

Specifications	**NodeMCU (ESP8266)**	**Arduino DUE**	**ESP32**
Microcontroller	ESP8266	AT91SAM3X8E	ESP32-WROOM-32
CPU Core	Tensilica Xtensa LX106	ARM Cortex-M3	Tensilica Xtensa LX6
Clock Speed	80 MHz	84 MHz	160 MHz
Flash Memory	128 KB	512 KB	4 MB
SRAM	4 MB	96 KB	520 KB
Digital I/O Pins	16	54	32
Analog Input Pins	$1\times$ 10-bit ADC	$12\times$ 12-bit ADC	$18\times$ 12-bit ADC
Analog Output Pins	-	$2\times$ 12-bit DAC	$2\times$ 8-bit DAC
Connectivity	IEEE 802.11 b/g/n FTDI USB UART	Native USB FTDI USB UART	802.11 b/g/n Bluetooth v4.2 BR/EDR and BLE FTDI USB UART

The test-bench developed for validating the performance of the *iEHO* algorithm consists of two main components. The first one relies on a software script running in MatLab®R2019, on an Intel® Core™ I7-4700HQ CPU, at 2.4 GHz, with 16 GB of RAM, on Windows® 8 (64 Bits) operating system. Firstly, the script is responsible for loading the generated observations according to (1), and supplying them to the embedded processor, complying with the entry format of the *iEHO* function call. It controls the loop, providing the M_c samples, depending on the vector size of the generated observations. Secondly, the script waits for an incoming message in the serial port. This corresponds to the processing of the algorithm in the embedded processor. When the data is available, in the third stage, the script receives the data, processes any occurred error, and stores the estimated position, along with the corresponding metrics (time and number of function evaluations). Figure 4 resumes the three procedures, each one being highlighted with different colors. It is worth mentioning that the elapsed time is measured on the embedded processor and not by the test-bench script. The second component relies on the embedded processor application software (firmware) that is running on the module, which was written in C language and performs several tasks (Figure 5). The complete compiled code (*iEHO* and test-bench) takes only 280 Kb of the flash memory and 28 Kb of RAM.

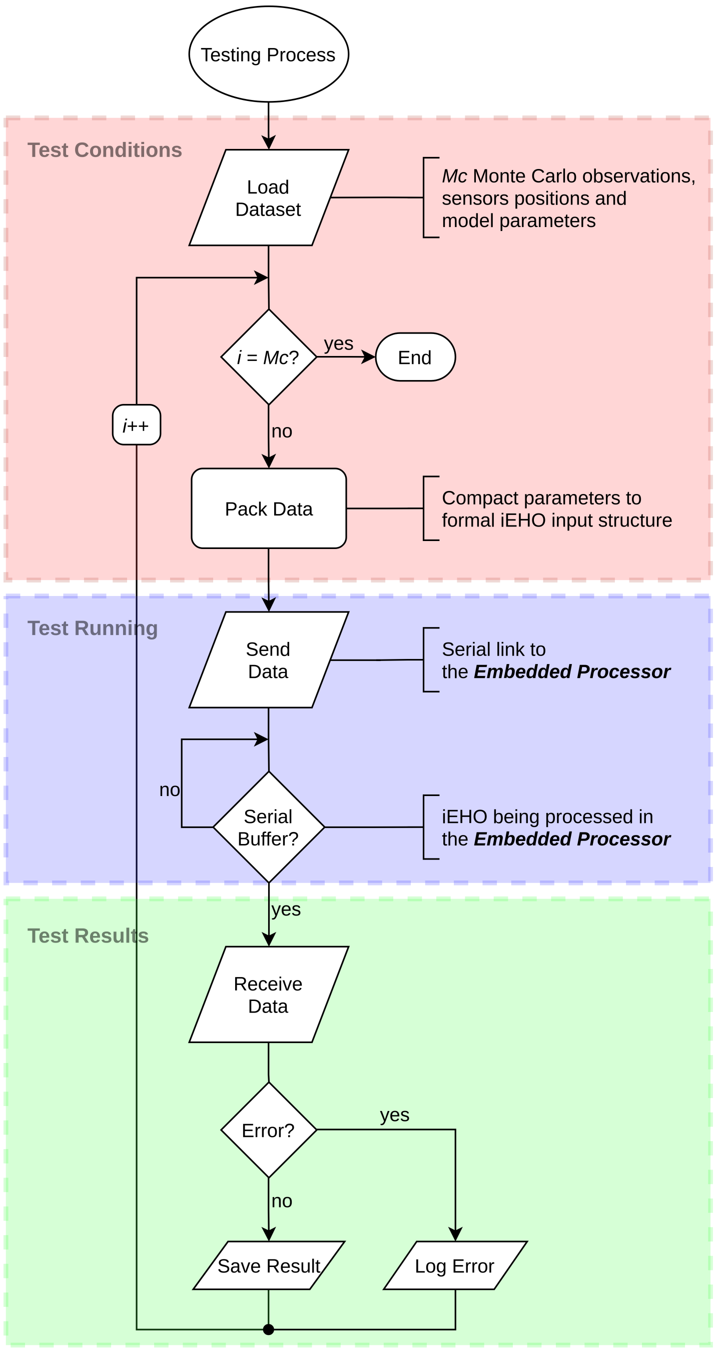

Testing Process
Test Conditions
Load Dataset
Mc Monte Carlo observations, sensors positions and model parameters
i = Mc?
yes
End
i++
no
Pack Data
Compact parameters to formal iEHO input structure
Test Running
Send Data
Serial link to the Embedded Processor
no
Serial Buffer?
iEHO being processed in the Embedded Processor
yes
Test Results
Receive Data
Error?
yes
no
Save Result
Log Error

Figure 4. Test-bench procedures flowchart.

As it can be seen from Figure 5, the developed firmware has some specific features for supporting the test-bench, more specifically the main cycle that receives the serial communication data and call the *iEHO* function, providing the observations under test. The figure shows a "Parameters Setup", a functional block that processes the model parameters and the positions of the sensors in an offline configuration, apart from the test running *iEHO*. Regarding the possibility of performing a *White-Box Test*, there are only two function calls that are flow-dependent (green blocks), which would be scorned in the test. These functions correspond to two distinct situations regarding the geometric characteristics of the distance estimations. Since several different scenarios will be randomly generated, for several values of the observations noise, both functions will be called, when considering the M_c observation data. With the goal of establishing the considered test conditions, Table 2 summarizes the values for the model parameters, test conditions, the sets of the sensor numbers and, the variance noise intervals.

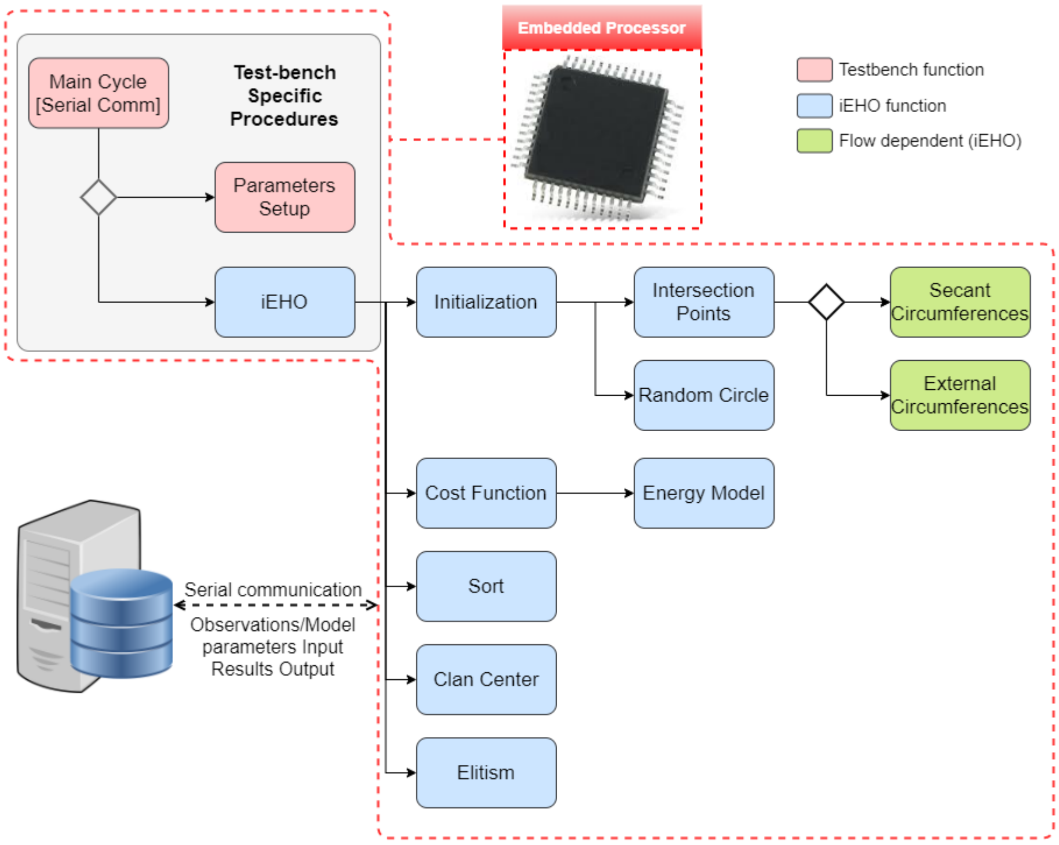


Figure 5. Embedded firmware call stack.

Table 2. Test-bench parameters set.

Model and Test Parameters		**Algorithm Parameters**	
Search Space	50×50	α (from [28])	0.7
P	5	β (from [28])	0.1
g_i	$1\ (i = 1, ..., N)$	L (from [29])	3
β_L	2	Δf	$5\dot{1}0^{-5}$
M_c	1000	Number of Clans	4
Variance Noise	$-80\ \text{dB} < \sigma_v^2 < -50\ \text{dB}$	Population Size	120
Sensor Number	$S_N \in (6, 9, 12, 15)$	Maximum Evaluations	3000

Regarding the variance noise, increments of 5 dB are considered for the proposed range. Overall, considering the 3 hardware modules, 4 different sets of sensors (6, 9, 12, 15), 7 sets of noise variance added to the observations, the 1000 Monte Carlos runs, running *iEHO* and *EHO* as the reference method, a total of 168,000 simulation are arranged and discussed on the test-bench. With regard to the evaluation metrics, the obtained results are processed in order to obtain: the root mean squared error (RMSE) of the M_c positions errors (the true source position used to generate the observations is applied); the mean value of the number of cost function evaluations; and the mean of the M_c processing times. Since four sets of sensors ($N = 6$, $N = 9$, $N = 12$ and $N = 15$) and seven sets of variance are under scrutiny, twenty eight values of each considered metric are available for validating the newly designed implementation.

5. Results and Discussion

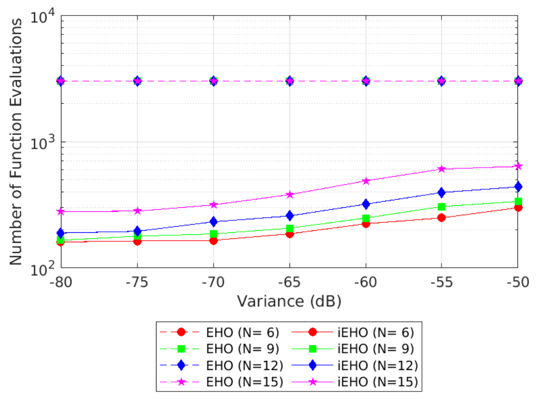
This section presents a set of numerical results based on the test-bench developed, where the three processed metrics of performance, over the M_c runs on the processor, are analyzed. For demonstration purposes, it is considered here that the sensors are uniformly distributed on a circle, centered at the middle point of the search space, deployed over a 50×50 square region. Sources are randomly generated over the search space.

Firstly, the number of the function cost evaluations is analyzed, where *iEHO* algorithm is compared with the standard *EHO*. As it can be seen from Figure 6, for the three modules under test (Figure 6a–c), the mean number of functions evaluations, over the *Mc*, runs is substantially decreased with regard of *EHO*, where the number of function evaluations is a constant initial parameter. In addition, when a lower number of sensors are considered, the number of evaluations is likewise lower. This situation occurs essentially due to the population initialization. Since the initial population is closer to the expected solution, less population generations will be carried out, accelerating the convergence rate of the algorithm and assessing the initialization of the population methodology. Concretely, although the maximum number of 3000 function evaluations was allowed, most of the processor runs were far below this number. This result indicates that the population was, in fact, initialized in the region of its (suboptimal) solution. Regarding the noise superimposed on the measurements, it can be seen that for higher values of its variance, the number of function evaluations also increases. This behavior is expected, since noisy measurements will widen the intersection region, resulting in a greater search space, i.e., more function evaluations.

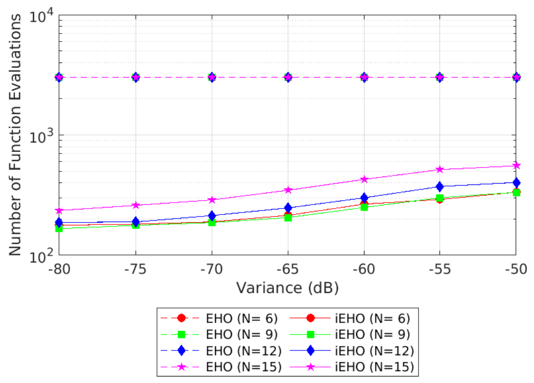
The significance of the mean number of function evaluations can only be validated if the localization error (RMSE) falls into acceptable values. Besides the gain obtained with regard to the number of function evaluations, it is mandatory to obtain a lower value of the RMSE, or leastwise, an equal value when applying *iEHO*. As such, Figure 7 plots the RMSE for both standard *EHO* and *iEHO* over the sets of the number of sensors and the range of the considered noise variance.

As it can be seen from Figure 7, the lowest error value occurs when $N = 15$ and $\sigma_v^2 = -80$ dB, that is, for the highest value of the number of sensors and the lowest value of the noise variance. When the number of sensors is decreased, higher values of the RMSE arises. With regard to the measurement noise, the *iEHO* algorithm always presents lower values than *EHO*, at the most, the *iEHO* RMSE is similar with its *EHO* counterpart for the highest values of the considered noise variance, following its growth. Even so, the use of *iEHO* is justified from the number of function evaluation perspective (please see Figure 6). When the noise is varied between its minimum and maximum value, as the noise increases, the error also reaches higher values. Nonetheless, when considering the mentioned search space, the sources far from the sensors will be subject to a very low signal-to-noise-ratio (SNR) value when disturbed with a noise of magnitude -50 dB. This situation is evident when, for high values of the noise power, increasing the number of sensors does not have the expected effect on the error reduction. This comportment is quite perceptible for small values of noise, where the stochastic behavior of the algorithm has scope effect on the results. When comparing the embedded implementation results with MatLab® (Figure 7d), one can see the same order of magnitude and evolution of the curves, although with slightly lower values. This difference is mostly due to the fact that MatLab® implements double-precision data type according to IEEE® Standard 754 (64 bits). On the other hand, the embedded firmware was implemented using single precision (32 bits). The analysis of both metrics, the number of function evaluations and the

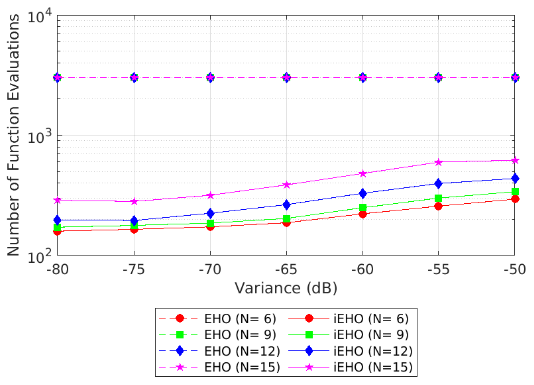
RMSE, shows a consistent behavior between all three modules considered for applying the developed test-bench. This situation allows to consider the proposed algorithm as robust with regards to real-life implementations, taking into account the used hardware platforms used.



(**a**) NodeMCU



(**b**) Arduino DUE



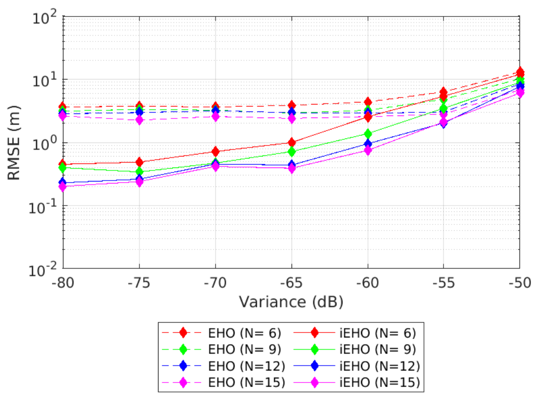
(**c**) ESP32

Figure 6. Mean value of the number of functions evaluations comparing standard *EHO* and *iEHO* for $N = 6$, $N = 9$, $N = 12$ and $N = 15$.

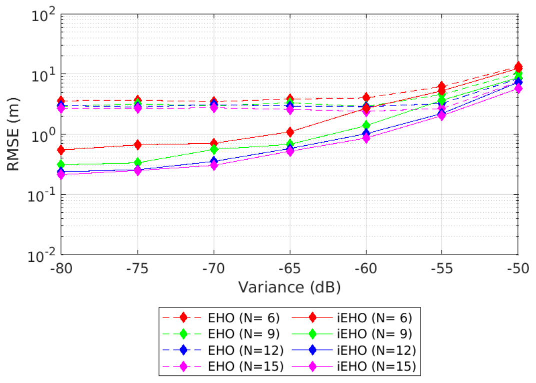
Having corroborated the behavior of the algorithm and verified the obtained positioning error values falls acceptable ranges, it remains to be analyzed whether this situation fits in execution times that allow running the method in real-time. For that purpose, the third metric, the processing time measured by the embedded processor, is analyzed. This metric is presented in relation to the number of sensors and the measurement noise, superimposed on the number of evaluations of the cost function. This overlay intends to demonstrate the correlation between the number of evaluations and the processing time aiming to illustrate the fitness of the population initialization methodology.

Observing Figure 8a–c, it can be seen that the mean computing time follows the mean number of the function evaluations over the M_c test runs. A clear distinction exists between using $N = 6$ or $N = 15$ sensors, where the lower number of sensors has a lower number of function evaluations. The same observation can be applied with the variance of the perturbation noise, where the execution time increases with the increase of the noise variance, a situation that is emphasized with a higher number of sensors. From a quantitative point of view, computing times are below 150 ms for reduced noise values. When the noise increases, their average remains around 100 ms when $N = 6$, and about 500 ms when $N = 15$ when using the NodeMCU and Arduino DUE modules. When considering the ESP32 module, the computation time reduces to 15 ms and 100 ms for $N = 6$ and $N = 15$ respectively.

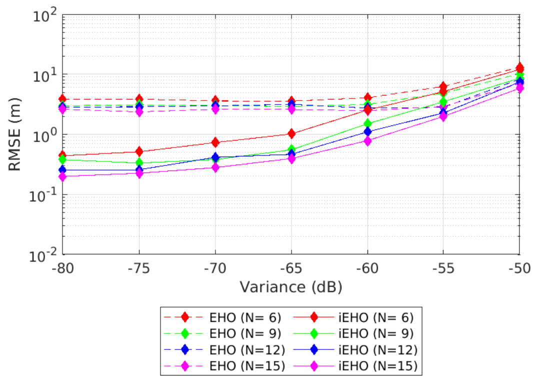
This huge difference for the ESP32 module is due to its clock speed, which duplicates regarding its counterparts.



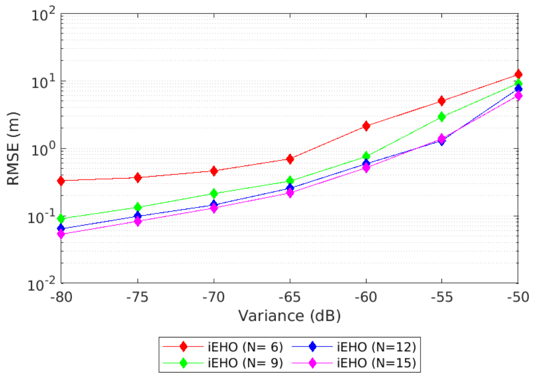
(**a**) NodeMCU



(**b**) Arduino DUE



(**c**) ESP32

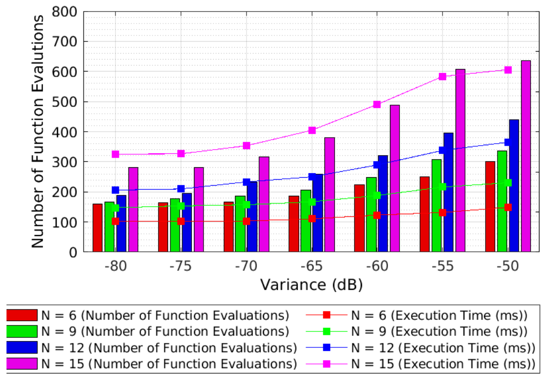


(**d**) Matlab

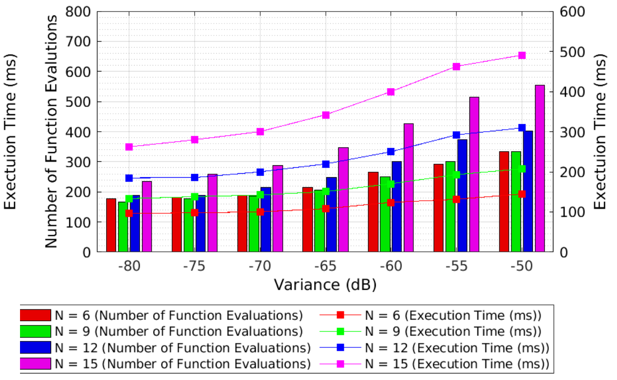
Figure 7. (**a**–**c**) RMSE comparing standard *EHO* and *iEHO* for $N = 6$, $N = 9$, $N = 12$ and $N = 15$. (**d**) RMSE of *iEHO* computed in MatLab for $N = 6$, $N = 9$, $N = 12$ and $N = 15$.

In Figure 9a–c, a comparison in execution time between standard *EHO* and *iEHO* is illustrated. It should be noticed that in the standard *EHO*, there is no difference between the different disturbance noises, since there is no initialization and, for every test, the maximum number of function evaluations is performed. However, there is a significant difference when considering the different sets of sensors. While a mean value around 1000 ms is obtained for $N = 6$, the value increases to about 2500 ms for $N = 15$ (more than 100%) for the Arduino DUE module. The rate is essentially the same for the NodeMCU and ESP32 modules. Once again, the difference is justified with the clock speed of the ESP32 module.

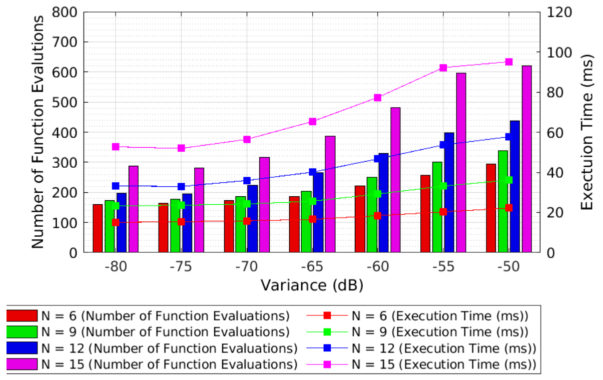
In the case of comparing the standard *EHO* and *iEHO* execution time, Tables 3–5 show the numerical values in relation to the average time of the M_c runs. The last line of each N sensor group is the fraction of *iEHO* with regard to *EHO* time in percentage, that means: $F = \frac{t_{iEHO}}{t_{EHO}} \times 100\%$. This indicator shows that for all modules, *iEHO* computation time is around 10% of *EHO* for $N = 6$. When increasing the number of sensors reaching $N = 15$, still the *iEHO* computation time reduction falls in the interval $[10, 25]\%$ of *EHO*, where higher values match greater values of the measurement noise.



(**a**) NodeMCU



(**b**) Arduino DUE

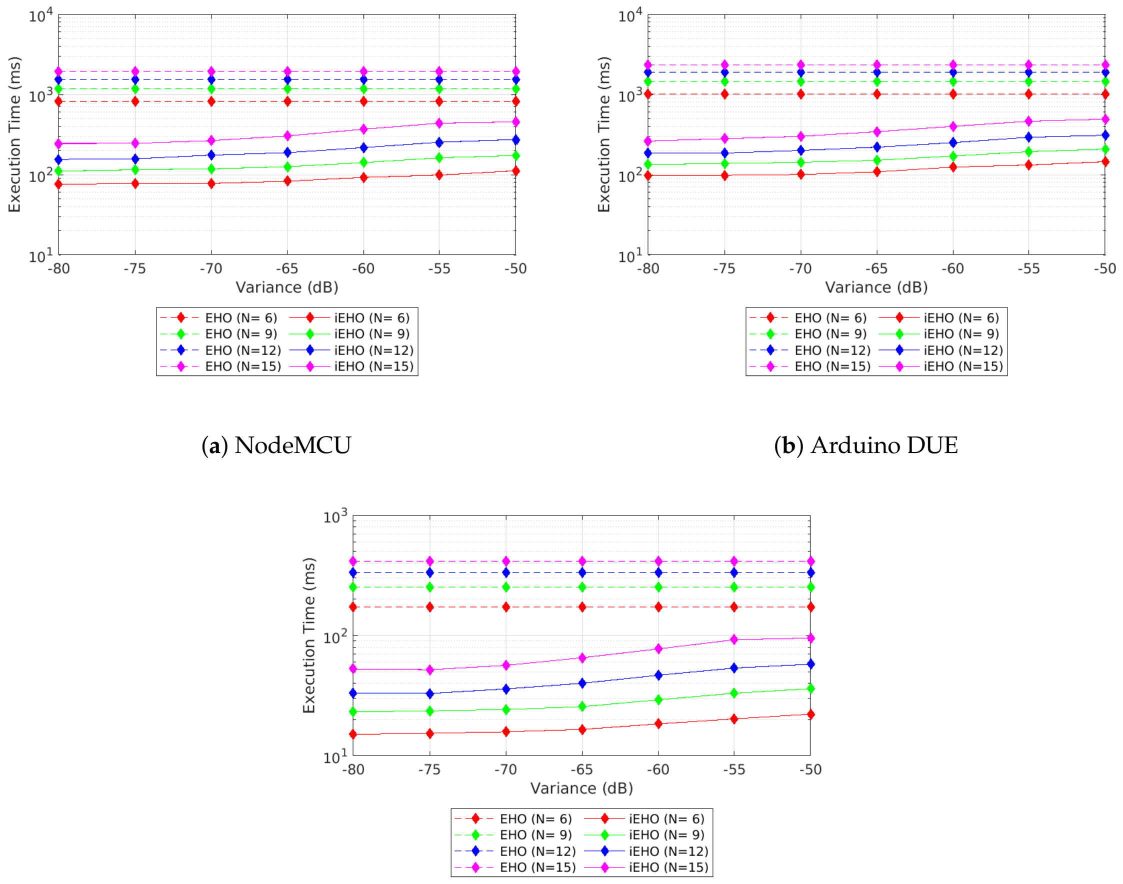


(**c**) ESP32

Figure 8. Execution time considering *iEHO* for *N* = 6, *N* = 9, *N* = 12 and *N* = 15 overlaid with the mean number of function evaluations.

Table 3. Execution time of *iEHO* over *EHO* (*N* = 6, 9, 12, 15) for NodeMCU module.

N	**Execution Time Test-Bench Data**							
6	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	818.34	818.36	818.43	818.32	818.47	818.49	818.25
	iEHO (ms)	76.20	76.93	77.43	82.73	92.23	98.71	111.54
	F(%)	**9.31%**	**9.40%**	**9.46%**	**10.11%**	**11.27%**	**12.06%**	**13.63%**
9	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	1183.2	1183.2	1183.2	1183.2	1183.2	1183.0	1182.7
	iEHO (ms)	110.5	115.1	117.9	125.3	140.9	162.0	172.8
	F(%)	**9.34%**	**9.72%**	**9.96%**	**10.59%**	**11.91%**	**13.70%**	**14.61%**
12	**Var(dB)**	−80	−75	−70	−65	−60	-55	−50
	EHO (ms)	1545.59	1545.65	1545.61	1545.62	1545.56	1545.31	1544.83
	iEHO (ms)	154.44	157.20	174.85	187.47	217.06	253.24	273.80
	F(%)	**9.99%**	**10.17%**	**11.31%**	**12.13%**	**14.04%**	**16.39%**	**17.72%**
15	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	1912.87	1912.80	1912.89	1912.76	1912.74	1912.50	1912.08
	iEHO (ms)	244.31	244.73	264.92	303.40	367.66	43740	454.54
	F(%)	**12.77%**	**12.79%**	**13.85%**	**15.86%**	**19.22%**	**22.87%**	**23.77%**



(**a**) NodeMCU (**b**) Arduino DUE

(**c**) ESP32

Figure 9. Execution time comparing standard *EHO* and *iEHO* for *N* = 6, *N* = 9, *N* = 12 and *N* = 15, using the ESP32 Module.

Table 4. Execution time of *iEHO* over *EHO* (*N* = 6, 9, 12, 15) for Arduino DUE module.

N	**Execution Time Test-Bench Data**							
6	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	1002.25	1002.32	1002.30	1002.20	1002.50	1002.57	1002.44
	iEHO (ms)	96.75	97.82	100.30	108.09	123.93	131.63	144.72
	F(%)	**9.65**	**9.76**	**10.01**	**10.78**	**12.36**	**13.13**	**14.44**
9	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	1444.84	1444.83	1444.76	1444.80	1444.61	1444.55	1444.15
	iEHO (ms)	133.16	138.01	142.31	150.65	170.22	193.09	207.42
	F(%)	**9.22**	**9.55**	**9.85**	**10.43**	**11.78**	**13.37**	**14.36**
12	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	1881.47	1881.58	1881.68	1881.62	1881.58	1881.24	1880.41
	iEHO (ms)	184.88	185.52	199.90	219.50	250.08	292.01	309.36
	F(%)	**9.83**	**9.86**	**10.62**	**11.67**	**13.29**	**15.52**	**16.45**
15	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	2323.59	2323.50	2323.52	2323.29	2323.37	2323.19	2322.74
	iEHO (ms)	262.21	279.68	300.02	342.04	399.72	462.74	490.71
	F(%)	**11.28**	**12.04**	**12.91**	**14.72**	**17.20**	**19.92**	**21.13**

Table 5. Execution time of *iEHO* over *EHO* (*N* = 6, 9, 12, 15) for ESP32 module.

N	Execution Time Test-Bench Data							
6	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	171.71	171.72	171.73	171.70	171.73	171.77	171.77
	iEHO (ms)	15.08	15.39	15.81	16.55	18.40	20.26	22.19
	F(%)	**8.78**	**8.96**	**9.21**	**9.64**	**10.71**	**11.80**	**12.92**
9	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	252.35	252.37	252.37	252.37	252.36	252.35	252.34
	iEHO (ms)	23.15	23.61	24.23	25.58	29.19	33.12	36.13
	F(%)	**9.17**	**9.36**	**9.60**	**10.13**	**11.57**	**13.12**	**14.32**
12	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	332.82	332.85	332.84	332.84	332.83	332.80	332.75
	iEHO (ms)	33.19	32.95	35.93	40.06	46.67	53.63	57.68
	F(%)	**9.97**	**9.90**	**10.79**	**12.03**	**14.02**	**16.12**	**17.33**
15	**Var(dB)**	−80	−75	−70	−65	−60	−55	−50
	EHO (ms)	413.30	413.29	413.29	413.27	413.26	413.24	413.20
	iEHO (ms)	52.70	51.90	56.42	65.28	77.31	92.16	95.10
	F(%)	**12.75**	**12.56**	**13.65**	**15.80**	**18.71**	**22.30**	**23.01**

6. Conclusions and Future Work

In the current work, a test-bench for evaluating and validating a swarm-based algorithm for acoustic localization was presented. The particularity of this implementation was the platform on which the algorithm was executed, where embedded modules were used to perform the calculations using and the simple procedure for supplying and retrieving data. The obtained results, over the proposed test-bench, validated the use of the recently proposed *iEHO* algorithm with low power, low cost, and low form factor microcontrollers on a wide range of performed tests. They indicated good accuracy, compared with the ones obtained with high-level programming languages, running on high-performance processors. Similar conclusions can be made regarding the processing time. In this case, the results obtained over the proposed test-bench showed computation times sufficiently low to allow *iEHO*'s implementation in real-time applications. This is owed to the accelerated convergence rate, which results in reduced latency, where values as low as 100 ms were obtained. Although the proposed test-bench was used for *iEHO*, its generalization to support implementation of other swarm-based algorithms is straightforward and requires very few modifications. Hence, this work stamps a new stage, where the improved convergence rate, associated with a reduced number of operations of *EHO* compared with *iEHO*, indirectly imply a lower energy consumption and, consequently, extension of the network's lifetime. This result is owed to the flexibility and reliability of the proposed test-bench. As such, new computational/software architectures can be deployed, where swarm-based algorithms (with low complexity and sufficient accuracy) are implemented on embedded frameworks to perform event localization, in this particular case, energy-based acoustic localization.

The proposed and evaluated test-bench in this work, and the data acquired by it, are set to be the fundamentals for several future developments. Creating a fully distributed platform or applying sequential localization algorithms can take full advantage of implementing the positioning method on an embedded processor. Even when considering a centralized scheme, processing the localization algorithm on the edge of the network (Edge Computing) assumes many advantages. Regarding the payload that is being transmitted over the network, its reduction has full implications on the transmission times, network congestion, or energy consumption, leading to new user applications and quality of service. Therefore, the *iEHO* algorithm can be useful for implementing sequential and distributed localization schemes or integrating IoT platforms that combine localization purposes.

Author Contributions: Conceptualization, S.D.C.; methodology, S.D.C., S.T. and M.B.; software, J.F.; validation, S.D.C., J.F., S.T. and M.B.; writing—review and editing, S.D.C., J.F., S.T. and M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by Fundação para a Ciência e a Tecnologia under Projects UIDB/04111/2020, foRESTER PCIF/SSI/0102/2017, and IF/00325/2015.

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

References

1. Hac, A. Embedded systems and sensors in wireless networks. In Proceeding of the IEEE 2005 International Conference on Wireless Networks, Communications and Mobile Computing, Maui, HI, USA, 13–16 June 2005; Volume 1, pp. 330–335. [CrossRef]
2. Kopetz, H. Internet of Things. In *Real-Time Systems Design Principles for Distributed Embedded Applications*; Springer Publishing Company, Incorporated: Berlin/Heidelberg, Germany, 2011; pp. 1–28, ISBN 978-1-4419-8236-0.
3. Sangiovanni-Vincentelli, A.; Di Natale, M. Embedded System Design for Automotive Applications. *Computer* **2007**, *40*, 42–51. [CrossRef]
4. Kovatsch, M.; Weiss, M.; Guinard, D. Embedding internet technology for home automation. In Proceedings of the 15th IEEE Conference on Emerging Technologies & Factory Automation, Bilbao, Spain, 13–16 September 2010. [CrossRef]
5. Pinheiro, E.M.; Correia, S.D. Software Model for a Low-Cost, IoT oriented Energy Monitoring Platform. *Int. J. Comput. Sci. Eng.* **2018**, *7*, 1–5. [CrossRef]
6. Pinheiro, E.M.; Correia, S.D. Hardware Architecture of a Low-Cost Scalable Energy Monitor System. *Int. J. Eng. Trends Technol.* **2018**, *61*, 1–5. [CrossRef]
7. Pinheiro, E.M.; Correia, S.D. Analog Input Expansion Board Based on I2C Communication with Plug-and-Play Feature, Applied to Current Measurements. *Int. J. Electron. Commun. Eng.* **2018**, *2*, 1–5. [CrossRef]
8. Alsina-Pagès, R.; Hernandez-Jayo, U.; Alías, F.; Angulo, I. Design of a Mobile Low-Cost Sensor Network Using Urban Buses for Real-Time Ubiquitous Noise Monitoring. *Sensors* **2016**, *17*, 57. [CrossRef]
9. Noriega-Linares, J.; Ruiz, J.N. On the Application of the Raspberry Pi as an Advanced Acoustic Sensor Network for Noise Monitoring. *Electronics* **2016**, *4*, 74. [CrossRef]
10. Dezan, C.; Zermani, S.; Hireche, C. Embedded Bayesian Network Contribution for a Safe Mission Planning of Autonomous Vehicles. *Algorithms* **2020**, *7*, 155. [CrossRef]
11. Baras, N.; Nantzios, G.; Ziouzios, D.; Dasygenis, M. Autonomous Obstacle Avoidance Vehicle Using LIDAR and an Embedded System. In Proceedings of the 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 13–15 May 2019; pp. 1–4. [CrossRef]
12. White, E. *Making Embedded Systems*, 1st ed.; O'Reilly Media, Inc.: Sevastopol, CA, USA, 2011; ISBN 978-1449302146.
13. Heath, S. *Embedded Systems Design*, 2nd ed.; Butterworth-Heinemann: Waltham, MA, USA, 2002.
14. Luo, G.; Guo, B.; Shen, Y.; Liao, H.; Ren, L. Analysis and Optimization of Embedded Software Energy Consumption on the Source Code and Algorithm Level. In Proceedings of the 2009 Fourth International Conference on Embedded and Multimedia Computing, Jeju, Korea, 10–12 December 2009; pp. 1–5. [CrossRef]
15. Engin, M. Energy Efficiency of Embedded Controllers. In Proceedings of the 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2019; pp. 1–4. [CrossRef]
16. Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. A survey on sensor networks. *IEEE Commun. Mag.* **2002**, *40*, 2–114. [CrossRef]
17. Yılmaz, M.; Günel, B. Investigation of Acoustic Source Localization algorithm performances for Gunshot Localization on helicopters. In Proceedings of the 2016 24th Signal Processing and Communication Application Conference (SIU), Zonguldak, Turkey, 16–19 May 2016; pp. 1885–1888. [CrossRef]
18. Kotus, J.; Lopatka, K.; Czyzewski, A. Detection and localization of selected acoustic events in acoustic field for smart surveillance applications. *Multimed. Tools Appl.* **2014**, *68*, 5–21. [CrossRef]
19. Colliera, T.C.; Kirschel, A.N.G.; Taylor, C.E. Acoustic localization of antbirds in a Mexican rainforest using a wireless sensor network. *J. Acoust. Soc. Am.* **2010**, *128*, 182–189. [CrossRef] [PubMed]

20. Mumolo, E.; Nolich, M.; Vercelli, G. Algorithms for acoustic localization based on microphone array in service robotics. *Robot. Auton. Syst.* **2003**, *42*, 69–88. [CrossRef]
21. Alsina-Pagès, R.M.; Alías, F.; Socoró, J.C.; Orga, F. Detection of Anomalous Noise Events on Low-Capacity Acoustic Nodes for Dynamic Road Traffic Noise Mapping within an Hybrid WASN. *Sensors* **2018**, *18*, 1272. [CrossRef]
22. Hollosi, D.; Nagy, G.; Rodigast, R.; Goetze, S.; Cousin, P. Enhancing Wireless Sensor Networks with Acoustic Sensing Technology: Use Cases, Applications and Experiments. In Proceedings of the IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, China, 20–23 August 2013; pp. 335–342. [CrossRef]
23. Quintana-Suárez, M.A.; Sánchez-Rodríguez, D.; Alonso-González, I.; Alonso-Hernández, J.B. A Low Cost Wireless Acoustic Sensor for Ambient Assisted Living Systems. *Appl. Sci.* **2017** *7*, 877. [CrossRef]
24. Yang, X.-S. *Nature-Inspired Optimization Algorithms*; Elsevier Inc.: Amsterdam, The Netherlands, 2020; 300p, ISBN 978-0-12-416743-8.
25. Zhang, Y.; Wang, S.; Ji, G. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Math. Probl. Eng.* **2015**, *2015*, 38. [CrossRef]
26. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 1942–1948.
27. Wang, G.; Deb, S.; Gao, X.; Coelho, L.S. A new metaheuristic optimisation algorithm motivated by elephant herding behavior. *Int. J. Bio-Inspir. Comput.* **2017**, *8*, 394–409. [CrossRef]
28. Correia, S.D.; Beko, M.; Cruz, L.A.S.; Tomic, S. Elephant Herding Optimization for Energy-Based Localization. *Sensors* **2018**, *18*, 2849. [CrossRef]
29. Correia, S.D.; Beko, M.; Tomic, M.; Da Silva Cruz, L.A. Energy-Based Acoustic Localization by Improved Elephant Herding Optimization. *IEEE Access* **2020**, *8*, 28548–28559. [CrossRef]
30. Mishra, S.; Singh, N.K.; Rousseau, V. Debug Interfaces. In *System on Chip Interfaces for Low Power Design*; Morgan Kaufmann: Burlington, MA, USA, 2016; Chapter 12, pp. 359–366, ISBN 9780128016305. [CrossRef]
31. Broekman, B.; Notenboom, E. *Testing Embedded Software*; Addison-Wesley: Boston, MA, USA; Sogeti Nederland: Vianen, The Netherlands, 2003.
32. IEEE. *Standard Test Access Port and Boundary-Scan Architecture*; IEEE std 1149.1-2001; IEEE: Piscataway, NJ, USA, 2001.
33. Whetsel, L. A High Speed Reduced Pin Count JTAG Interface. In Proceedings of the 2006 IEEE International Test Conference, Santa Clara, CA, USA, 22–27 October 2006; pp. 1–10. [CrossRef]
34. Singh, B.; Patil, S. Single Wire Debug Interface. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 9–12 August 2020; pp. 814–817. [CrossRef]
35. Gialelis, J.; Fokaeos, M.; Theodorou, G.; Paparizos, C.; Tsafas, N. A Low Cost Energy Efficient IoT Node Utilized in the Agricultural Field. In Proceedings of the 2020 9th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 8–11 June 2020; pp. 1–6. [CrossRef]
36. Zhang, Y.; Wang, Y.; Huang, F. A Platform with JTAG Debugging of SoC Based on System Level Verification. In Proceedings of the 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), Xián, China, 12–14 June 2019; pp. 1–2. [CrossRef]
37. Park, H.; Xu, J.; Park, J.; Ji, J.-H.; Woo, G. Design of On-Chip Debug System for embedded processor. In Proceedings of the 2008 International SoC Design Conference, Busan, Korea, 24–25 November 2008; pp. 11–12. [CrossRef]
38. Hansen, M.T. Multi-purpose passive debugging for embedded wireless. In Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, Chicago, IL, USA, 12–14 April 2011; pp. 155–156.
39. Shen, C.; Herman, H.; Charbiwala, Z.; Srivastava, C.; Shen, H.; Herman, Z.C.; Srivastava, M.B. Demo abstract: MiDebug: Microcontroller integrated development and debugging environment. In Proceedings of the 2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN), Beijing, China, 16–19 April 2012; pp. 133–134. [CrossRef]
40. Capella, J.V.; Campelo, J.C.; Bonastre, A.; Ors, R. A Reference Model for Monitoring IoT WSN-Based Applications. *Sensors* **2016**, *16*, 1816. [CrossRef]

41. Guerrero-Rodríguez, J.M.; Cobos-Sánchez, C.; González-de-la-Rosa, J.J.; Sales-Lérida, D. An Embedded Sensor Node for the Surveillance of Power Quality. *Energies* **2019**, *12*, 1561. [CrossRef]
42. Ferracuti, F.; Freddi, A.; Monteriú, A.; Prist, M. An Integrated Simulation Module for Cyber-Physical Automation Systems. *Sensors* **2016**, *16*, 645. [CrossRef] [PubMed]
43. Kim, K.; Choi, K.Y.; Lee, J.W. Fault Localization Method by Partitioning Memory Using Memory Map and the Stack for Automotive ECU Software Testing. *Appl. Sci.* **2016**, *6*, 266. [CrossRef]
44. Li, D.; Hu, Y.H. Energy Based Collaborative Source Localization Using Acoustic Micro-Sensor Array. *Appl. Signal Process* **2003**, 321–337. [CrossRef]
45. Sheng, X.; Hu, Y.H. Maximum likelihood multiple-source localization using acoustic energy measurements with wireless sensor networks. *IEEE Trans. Signal Process* **2005**, *53*, 44–53. [CrossRef]
46. Sun, G.; Chen, J.; Guo, W.; Liu, K.J.R. Signal processing techniques in network-aided positioning: A survey of state-of-the-art positioning designs. *IEEE Signal Process. Mag.* **2005**, *22*, 12–23. [CrossRef]
47. Ho, K.C.; Sun, M. An accurate algebraic closed-form solution for energy-based source localization. *IEEE Trans. Audio Speech Lang. Process.* **2007**, *15*, 2542–2550. [CrossRef]
48. Beck, A.; Stoica, P.; Li, J. Exact and Approximate Solutions of Source Localization Problems. *IEEE Trans. Signal Process* **2008**, *56*, 1770–1778. [CrossRef]
49. Meesookho, C.; Mitra, U.; Narayanan, S. On energy-based acoustic source localization for sensor networks. *IEEE Trans. Signal Process.* **2008**, *56*, 365–377. [CrossRef]
50. Wang, G.; Yang, K. A Semidefinite Relaxation Method for Energy-based Source Localization in Sensor Networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Taipei, Taiwan, 15 April 2011; pp. 2257–2260. [CrossRef]
51. Beko, M. Energy-based localization in wireless sensor networks using semidefinite relaxation. In Proceedings of the IEEE Wireless Communications and Networking Conference, Cancun, Mexico, 28–31 March 2011; pp. 1552–1556. [CrossRef]
52. Beko, M. Energy-Based Localization in Wireless Sensor Networks Using Second-Order Cone Programming Relaxation. *Wirel. Pers. Commun.* **2014**, *77*, 1847–1857. [CrossRef]
53. Correia, S.D.; Beko, M.; Cruz, L.A.S.; Tomic, S. Implementation and Validation of Elephant Herding Optimization Algorithm for Acoustic Localization. In Proceedings of the 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 20–21 November 2018; pp. 1–4. [CrossRef]
54. Kim, C.T.; Choi, T.Y.; Choi, B.; Lee, J.J. Robust estimation of sound direction for robot interface. In Proceedings of the 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 19–23 May 2008; pp. 3475–3480. [CrossRef]
55. Badali, A.; Valin, J.M. ; Michaud, F.; Aarabi, P. Evaluating real-time audio localization algorithms for artificial audition in robotics. In Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, USA, 10–15 October 2009; pp. 2033–2038. [CrossRef]
56. Hilsenbeck, B.; Kirchner, N. Listening for people: Exploiting the spectral structure of speech to robustly perceive the presence of people. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 2903–2909. [CrossRef]
57. Deng, F.; Guan, S.; Yue, X.; Gu, X.; Chen, J.; Lv, J.; Li, J. Energy-Based Sound Source Localization with Low Power Consumption in Wireless Sensor Networks. *IEEE Trans. Ind. Electron.* **2017**, *64*, 4894–4902. [CrossRef]
58. Faraji, M.M.; Shouraki, M.M.; Iranmehr, E. Fuzzy based algorithm for acoustic source localization using array of microphones. In Proceedings of the 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2–4 May 2017; pp. 2102–2105. [CrossRef]
59. Wang, T.; Choy, Y. An approach for sound sources localization and characterization using array of microphones. In Proceedings of the 2015 International Conference on Noise and Fluctuations (ICNF), Xian, China, 2–6 June 2015; pp. 1–4. [CrossRef]
60. Vikhar, P.A. Evolutionary algorithms: A critical review and its future prospects. In Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Jalgaon, India, 22–24 December 2016; pp. 261–265. [CrossRef]
61. Strumberger, I.; Bacanin, N.; Tomic, S.; Beko M.; Tuba, M. Static drone placement by elephant herding optimization algorithm. In Proceedings of the 25th Telecommunication Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017; pp. 1–4. [CrossRef]

62. Alihodzic, A.; Tuba, E.; Capor-Hrosik, R.; Dolicanin, E.; Tuba, M. Unmanned aerial vehicle path planning problem by adjusted elephant herding optimization. In Proceedings of the 25th Telecommunication Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017; pp. 1–4. [CrossRef]
63. Bentouati, B.; Chettih, S.; El Sehiemy, R.; Wang, G.G. Elephant herding optimization for solving non-convex optimal power flow problem. *J. Electr. Electron. Eng.* **2017**, *10*, 31–36.
64. Shankar, K.; Elhoseny, M.; Perumal, E.; Ilayaraja, M.; Sathesh Kumar, K. An Efficient Image Encryption Scheme Based on Signcryption Technique with Adaptive Elephant Herding Optimization. In *Cybersecurity and Secure Information Systems*; Hassanien A., Elhoseny M., Eds.; Advanced Sciences and Technologies for Security Applications; Springer: Cham, Switzerland, 2019. [CrossRef]
65. Sambariya, D.K.; Fagna, R. A robust PID controller for load frequency control of single area re-heat thermal power plant using elephant herding optimization techniques. In Proceedings of the International Conference on Information, Communication, Instrumentation and Control (ICICIC), Indore, India, 17–19 August 2017; pp. 1–6. [CrossRef]
66. Sambariya, D.K.; Fagna, R. A novel Elephant Herding Optimization based PID controller design for Load frequency control in power system. In Proceedings of the International Conference on Computer, Communications and Electronics (Comptelix), Jaipur, India, 1–2 July 2017; pp. 595–600. [CrossRef]
67. Strumberger, I.; Beko, M.; Tuba, M.; Minovic, M.; Bacanin, N. Elephant Herding Optimization Algorithm for Wireless Sensor Network Localization Problem. In *Technological Innovation for Resilient Systems*; Camarinha-Matos L., Adu-Kankam K., Julashokri M., Eds.; DoCEIS 2018; IFIP Advances in Information and Communication Technology; Springer: Cham, Switzerland, 2018; Volume 521. [CrossRef]
68. Strumberger, I.; Minovic, M.; Tuba, M.; Bacanin, N. Performance of Elephant Herding Optimization and Tree Growth Algorithm Adapted for Node Localization. *Sensors* **2019** *19*, 2515. [CrossRef]
69. Cohen, R.; Davy, G.; Féron, E.; Garoche, P; Formal Verification for Embedded Implementation of Convex Optimization Algorithms. *IFAC-PapersOnLine* **2017**, *50*, 5867–5874. [CrossRef]
70. Diamond, S.; Boyd, S.; CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *J. Mach. Learn. Res.* **2016**, *17*, 1–5.