machines

MDPI

*Article*

# Memetic Algorithm for Energy Optimization in Point-to-Point Robotized Operations

**Sandi Baressi Šegota [1,*], Domagoj Frank [2], Ivan Lorencin [1] and Nikola Anđelić [3]**

[1] Faculty of Informatics, Juraj Dobrila University of Pula, 52100 Pula, Croatia
[2] Department of Computer Science and Informatics, University North, 48000 Koprivnica, Croatia
[3] Department of Computer Engineering, Faculty of Engineering, University of Rijeka, 51000 Rijeka, Croatia
* Correspondence: sandi.baressi.segota@unipu.hr

**Abstract**

This paper presents a memetic algorithm (MA) for energy cost estimation of a robot path. The developed algorithm uses a random recombination genetic algorithm (GA) as the basis for the first stage of the algorithm and performs a local search based on feature importances determined from the data in the second stage. To allow for the faster determination of the solution quality, the algorithm uses an ML-driven fitness function, based on MLP, for the determination of path energy. The performed tests show that not only does the GA itself optimize the point-to-point paths well, but the usage of MA can lower the energy use by 58% on average (N = 100) when compared to a linear path between the same two points.

**Keywords:** energy optimization; industrial robots; memetic algorithm

## 1. Introduction

The number of industrial robots (IRs) present in manufacturing plants is ever growing [1]. A common use of IRs is in the so-called point-to-point operation mode, where the movement between two points does not have to follow a specific path; instead, it can move freely between two points in space—with most operations applying a simple movement in between those two points, such as a linear or semi-circular motion. Due to the number of robots and the large amount of working hours applied, the energy required to operate these machines is not insignificant. Assuming a 5 kW power per robot and 1000 work hours a year, the consumption in the European Union area exceeds 15 TWh. It is not surprising that a large number of papers cover the need for energy optimization during IR operations. For this reason, there are many papers that explore the path tuning of IRs with the goal of lowering the energy use. Vysockyy et al. [2] applied a particle swarm optimization (PSO) algorithm for point-to-point robot path optimization, focusing on non-technological movements—segments where the robot is not directly involved in a production task. The optimized Bezier-curve-based trajectories were verified on a UR3 robotic arm, demonstrating energy savings between 10 and 40%. The authors suggest that similar approaches could be extended to technological segments when process constraints allow. Garriz and Domingo [3] optimized sealant application in the automotive industry using a Kalman-based trajectory planner, achieving up to 20% reduction in energy use under realistic constraints, though at the cost of increased cycle time. The authors note that this trade-off may be undesirable in assembly-line environments where timing affects subsequent tasks, indicating that approaches minimizing the energy without prolonging the operation time would be more generally applicable. Shrivastava and Dalla [4] utilized

Check for updates

Academic Editor: Luca Bruzzone

Received: 18 November 2025
Revised: 12 December 2025
Accepted: 24 December 2025
Published: 25 December 2025

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

a classical genetic algorithm (GA) to optimize the energy consumption of a multi-axis manipulator based on bond-graph-simulated kinematic and dynamic data. Using joint positions as optimization parameters, their method achieved a 38% improvement compared to direct path planning while maintaining compliance with physical constraints. Similarly, Nonoyama et al. [5] used the K-ROSET simulation environment to apply GA and PSO algorithms to tune the parameters of a PID controller in a SCARA robot. Their results showed that the GA yielded a lower computational cost and improved energy efficiency by 18% relative to the untuned trajectory, though the method's applicability to other robot types may be limited by the difficulty of adjusting the controller parameters. Luneckas et al. [6] extended heuristic optimization to hexapod robots, applying a Red Fox algorithm to enhance the gait-switching efficiency, resulting in energy savings of up to 21%. These findings indicate that more advanced heuristic approaches can outperform classical algorithms such as GA and PSO in trajectory optimization. Lu et al. [7] further demonstrated the use of a memetic algorithm (MA) combining GA and variable neighborhood search (VNS) for collaborative welding applications, where a local neighborhood search after global optimization improved the results by up to 10%. Despite its success, the study did not explore alternative algorithm combinations for the two stages, even though prior work suggests that other evolutionary algorithms, such as differential evolution (DE), may yield superior results [8]. The main findings of these works, including the highest reported improvements in energy efficiency, are summarized in Table 1. The reported improvements are given as a percentage of the improvement from the initial state. It is important to note that different studies express the values of the fitness function in different ways, resulting in different baselines and physical quantities. The assumption used for the presented value is that with the given optimization strategy a similar enough fitness function would yield results close to what was presented in Table 1 [9]. The main findings of these works, including the highest reported improvements in energy efficiency, are summarized in Table 1.

**Table 1.** The best results from the reviewed research focused on application of evolutionary computing algorithms on robot energy efficiency improvement. Improvement between the unoptimized and optimized paths expressed as a percentage and rounded to the closest full value.

| Reference | Approach | Improvement [%] |
|---|---|---|
| Vysocky et al. [2] | GA, PSO, Bezier curves | 40 |
| Garriz and Domingo [3] | Kalman | 20 |
| Shrivastava and Dalla [4] | GA | 38 |
| Nonoyama et al. [5] | K-ROSET, GA, PSO | 18 |
| Luneckas et al. [6] | Red Fox | 21 |
| Lu et al. [7] | MA − GA + VNS | 10 |

The research review shows that there is a clear gap in the application of memetic algorithms (MA) in the process of the energy path optimization of IRs. Still, another issue exists in the application of these algorithms, which is the need for the precise and fast evaluation of the actual energy use. ML algorithms have been shown to have a wide array of applications in both robotics and energy use prediction. Gadaleta et al. [10] present an experimental framework for studying and optimizing the energy consumption of IRs. Their setup, based on a KUKA KR210 R2700 Prime robot performing various operations under different loads, serves as a foundation for building data sets that enable analysis of factors influencing energy use. The results indicate that even basic operational adjustments, such as improved temperature control, can reduce energy consumption by up to 50%, while higher-level strategies like trajectory optimization could yield further improvements. This study highlights the importance of creating comprehensive experimental data sets to identify variables with the largest effect on robotic energy efficiency. Zhang and Yan [11] propose a

data-driven approach using a back-propagation artificial neural network (ANN) to predict the energy consumption of an Epson C4 IR operating at different speeds and accelerations. Their model achieves an R-value of 0.9699, although the lack of cross-validation and limited data set diversity suggest possible overfitting. Gao et al. [12] employ a Long Short-Term Memory (LSTM) neural network for the torque-based energy prediction of a planar parallel manipulator (PPM), using torque values derived from Newton–Euler dynamics. The model achieves a root mean square error (RMSE) of 0.000041 and a mean absolute percentage error (MAPE) of 2.5%. Lin et al. [13] extend this approach with a batch-normalized (BN) LSTM, trained on 64,600 samples from the Yaskawa GP7 data set, and report an RMSE of 3.67—representing a 23% improvement over previous results. Similarly, Jiang et al. [14] apply an LSTM-based deep neural network to predict the energy consumption of a KUKA KR60-3 IRM, achieving an MAPE of 4.21%. Although LSTM architectures have shown excellent performance in modeling temporal energy trends, they are limited in real-time adaptability. Jaramillo-Morales et al. [15] demonstrate an alternative numeric model for the direct prediction of instantaneous power consumption in a differential-drive robot, the Nomad Super Scout II. While this approach enables use in dynamic environments, its accuracy drops to 81.25% on curved paths, remaining inferior to ML-based models. Overall, these works underline the growing effectiveness of machine learning methods, particularly ANN and LSTM architectures, for predicting robotic energy use, though further refinement is needed for real-time and generalized applications. The summarized results of the reviewed research are presented in Table 2.

**Table 2.** The best results from the reviewed research focused on the energy use modeling of robots. RMSE—root mean square error, MAPE—mean absolute percentage error.

| Reference | Approach | Score |
|---|---|---|
| Zhang & Yan [11] | ANN | $R$-score = 0.97 |
| Gao et al. [12] | LSTM | MAPE = 2.5% |
| Lin et al. [13] | BN-LSTM | RMSE = 3.67 |
| Jiang et al. [14] | LSTM | MAPE = 4.24% |
| Jaramillo et al. [15] | Numeric | Accuracy = 82.15% |

Based on this, the following hypotheses can be set:

- An ML-based model can be used for regressing the value of energy use in the given moment, using the information about IR speed and position,
- An MA algorithm can be developed to lower the energy use of IR in point-to-point-based operations.

The paper will first demonstrate the build of the algorithm—from the data collection and analysis to the modeling approach, feature importance estimation, and construction of the MA algorithm. Then, the results of each of these steps will be provided and commented on prior to drawing conclusions.

## 2. Materials and Methods

### *2.1. Determining the Energy Use of the Robot to Be Used as a Fitness Function*

The energy use of the IR can be defined as the total sum of the products of torques and velocities in each individual joint [16]:

$$E_{total} = \sum_{i=0}^{i=6} E_i = \sum_{i=0}^{i=6} \tau_i \omega_i \Delta t, \quad (1)$$

where:

- $E_{\text{total}}$—total energy consumed by the robot during the movement, $J$;
- $E_i$—energy consumed by the $i$-th joint, $J$;
- $\tau_i$—torque of the $i$-th joint, N, m;
- $\omega_i$—angular velocity of the $i$-th joint, rad/s;
- $\Delta t$—time step between two measurements, s.

While obtaining the velocity values is simple and direct, the torque of each individual joint is complex—it requires the use of an iterative algorithm such as Newton–Euler or Lagrange–Euler, and the resulting model is often computationally very complex [17]. The complexity issue is non-trivial here, as the applied evolutionary algorithms require the calculation of the total energy a large number of times. If a computationally complex function is used, the time required for optimization could increase significantly [18]. Due to this, the authors used an ML approach, that, while more complex to train, allows for a much faster inference time [19]. The data for training are obtained from the ABB IRB 120 IR. The data are collected from measurements given by the IR control unit, with the code given in Appendix A. The data were collected on the movement between 1000 randomly selected data points. This approach was selected in order to include a higher variety of movement between these data points. As the frequency of collection was 40 Hz—40 data points per second, the final data set consists of 75,790 data points. The torque measurements were obtained from the IR firmware (so-called RobotWare) as provided by the manufacturer, through the manufacturer's software for control of robots, ABB RobotStudio (version 2024.2) [20]. The values are derived from motor current internally measured by the IR control unit and the motor model (e.g., the torque constants of the motors and filters) [21]. To ensure the results of the models are correct, the data collection experiment was collected to obtain the test set. The test set was generated on 100 points in space, which returned 7298 data points. The models that were trained on the original data in the cross-validation scheme were then re-evaluated on this data set. The collected values in each of these points are as follows:

- Tool center position (TCP) given as the $x$, $y$, and $z$ coordinates of the end of the IR relative to the origin placed in the base of the IR;
- Quarternions ($q_1$, $q_2$, $q_3$, and $q_4$) and Euler angles ($\phi$, $\theta$, $\psi$—encoded as $X$, $Y$, and $Z$, respectively) that define the rotation of the TCP;
- Speed of each individual joint;
- Torque of each individual joint.

In addition to this, the following values were calculated for each collected data point:

- $\Delta t$—the measurement time between two data points,
- Acceleration of each axis,
- The change in the linear position of TCP since the last measurement,
- The change in each of the Euler's angles,
- The linear speeds and accelerations of TCP;
- The angular speeds and accelerations,
- The total energy that the robot used in the movement per Equation (1).

The measurement was performed according to the diagram given in Figure 1. The torque data were primarily collected by the IRC5, which measures the current sent to the ABB IRB 120 motors through the A05 connection. Other used information, such as the robot position and speed was collected and measured by the Serial Measurement Board (SMB) located in the base of the IRB 120. The SMB sends the collected data back to the IRC5 control unit through an A06 connection. Both of these sets of values are then sent to the RobotStudio instance on a PC over an ethernet connection, where the data can be logged for later use in modeling.

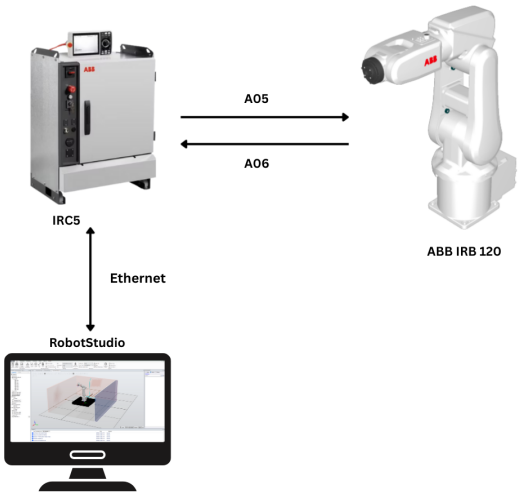



**Figure 1.** Diagram of the data collection process.

These values were calculated in order to allow for simpler regression of the target output. The training data set is provided in Supplementary Materials Table S1, while the test data set is provided in Supplementary Materials Table S2. All methods have been trained using five-fold cross-validation with grid search for hyperparameter determination. Four ML methods have been tested. The first is multilayer perceptron (MLP). MLP is one of the most fundamental neural network architectures. It consists of an input layer, one or more hidden layers, and an output layer, where each neuron applies a nonlinear activation function to a weighted sum of its inputs [22]. The training process involves adjusting the connection weights through backpropagation to minimize a defined loss function, often using stochastic gradient descent algorithm or its variants [23]. The main advantage of the MLP is its ability to approximate complex nonlinear mappings, making it suitable for regression and classification tasks. However, due to its dense structure, it may require large data sets and regularization to avoid overfitting. The hyperparameters tested within the grid search, for the MLP, are given in Table 3.

**Table 3.** Hyperparameter values tested in the GS process for MLP.

| Hyperparameter | Values |
|---|---|
| Hidden layer sizes | (10), (50), (100), (10, 10), (50, 50), (100, 100), (10, 10, 10), (50, 50, 50), (100, 100, 100), (10, 10, 10, 10), (50, 50, 50, 50), (100, 100, 100, 100) |
| Activation function | `identity`, `logistic`, `tanh`, `relu` |
| Solver | `lbfgs`, `adam` |
| Alpha | 0.0001, 0.001, 0.01, 0.1 |
| Learning rate type | `constant`, `adaptive`, `invscaling` |
| Initial learning rate | 0.01, 0.1, 1 |

Support Vector Regression (SVR) extends the principles of the Support Vector Machine to continuous-valued prediction tasks [24]. It constructs a regression function that deviates from the true targets by at most a predefined margin, while maintaining the model simplicity by maximizing the margin between data points and the regression boundary [25]. The optimization process relies on solving a convex quadratic programming problem that balances the model flatness and tolerance to error. By using kernel functions, the SVR can capture nonlinear dependencies between variables, making it an effective tool for high-dimensional and complex regression problems. The hyperparameter search space for this method is given in Table 4.

**Table 4.** Hyperparameters and their values for SVM regressor.

| Hyperparameter | Values |
|---|---|
| Kernel | `linear`, `poly`, `rbf`, `sigmoid` |
| Degree | 2, 3, 4 |
| $\gamma$ | `scale`, `auto` |
| $C$ | 0.1, 0.5, 1, 10 |
| $\varepsilon$ | 0.1, 0.2, 0.5, 1 |

The passive-aggressive regressor (PAR) belongs to a family of online learning algorithms that updates the model parameters incrementally as new data become available [26]. Unlike batch learners, it does not retrain on the entire data set but instead performs minimal corrections to the model after each sample. The "passive" aspect reflects that no update is made when the prediction error is within a defined tolerance, whereas the "aggressive" aspect refers to the immediate correction applied when the error exceeds this threshold [27]. This makes the method well-suited for real-time or streaming regression tasks, where adaptability and computational efficiency are essential. The tuned hyperparameters and their possible values are presented in Table 5.

**Table 5.** Hyperparameters and their values for PAR.

| Hyperparameter | Values |
|---|---|
| $C$ | 0.1, 0.5, 1, 10 |
| `fit_intercept` | `True`, `False` |
| Tolerance | $10^{-3}$, $10^{-4}$, $10^{-5}$ |
| Loss | `epsilon_insensitive`, `squared_epsilon_insensitive` |
| $\varepsilon$ | 0.1, 0.2, 0.5, 1 |

Gradient Boosted Trees (XGB) are an ensemble learning method that combines the outputs of multiple weak learners, typically decision trees, into a strong predictive model [28]. The approach incrementally builds the model in a stage-wise fashion, where each new tree is trained to minimize the residual errors of the previous ensemble using the gradient descent in function space [29]. This allows GBTs to achieve high predictive accuracy and robustness, particularly on structured and tabular data. The trade-off lies in the relatively high computational cost of training, which scales with the number of boosting stages and the depth of the individual trees. The hyperparameters of this method are given in Table 6.

**Table 6.** Hyperparameters and their values for XGB.

| Hyperparameter | Values |
|---|---|
| `n_estimators` | 10, 50, 100 |
| `max_depth` | 3, 4, 5, 6, 7 |
| `learning_rate` | 0.01, 0.1, 1 |
| `subsample` | 0.5, 0.75, 1 |
| `colsample_bytree` | 0.5, 0.75, 1 |
| `colsample_bylevel` | 0.5, 0.75, 1 |
| `colsample_bynode` | 0.5, 0.75, 1 |
| `reg_alpha` | 0, 0.1, 0.5, 1 |
| `reg_lambda` | 0, 0.1, 0.5, 1 |

### *2.2. Feature Importance*

Feature importance can be determined through several complementary approaches, each quantifying the contribution of individual predictors to the model output in a different manner. The Mean Decrease in Impurity (MDI) evaluates the importance in tree-based

models by averaging the reduction in the impurity of nodes, such as the Gini index or entropy, weighted by the number of samples reaching each node [30]. It can be expressed as follows.

$$I_j = \sum_{t \in T_j} p(t), \Delta i(t), \tag{2}$$

where $p(t)$ is the proportion of samples at node $t$, and $\Delta i(t)$ is the decrease in impurities. The feature permutation method estimates importance by measuring the change in model performance when the values of a given feature are randomly permuted, effectively breaking its relationship with the target variable [31]. The resulting importance is computed as

$$I_j = \mathcal{M}\text{baseline} - \mathcal{M}\text{perm}(j), \tag{3}$$

where $\mathcal{M}$ denotes the performance metric chosen. Statistical correlation-based methods provide model-agnostic alternatives. The Pearson correlation coefficient measures linear relationships as follows:

$$r_{xy} = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}},$$

while the Spearman rank correlation uses ranked variables to capture monotonic associations, given by

$$\rho_{xy} = 1 - \frac{6 \sum_i d_i^2}{n(n^2 - 1)},$$

where $d_i$ is the rank difference for sample $i$ [32]. Finally, the Kendall correlation coefficient quantifies the concordance between pairs of observations as

$$\tau = \frac{n_c - n_d}{\frac{1}{2} n(n-1)},$$

where $n_c$ and $n_d$ are the numbers of concordant and discordant pairs, respectively [33]. These methods collectively offer a thorough insight into the importance of variables, integrating both model-specific and statistical viewpoints.

### *2.3. Evolutionary Optimization Algorithms*

A genetic algorithm (GA) is a search heuristic and optimization technique inspired by the principles of natural selection and genetics. It is used to solve complex optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

GA is an iterative process that starts with the initialization of multiple randomized solutions. For each of these solutions, the quality of the solution is calculated. In the presented research, the solution is the trajectory of the robot in the joint space, defined with six values per each joint:

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 & e_1 & f_1 \\ a_2 & b_2 & c_2 & d_2 & e_2 & f_2 \\ a_3 & b_3 & c_3 & d_3 & e_3 & f_3 \\ a_4 & b_4 & c_4 & d_4 & e_4 & f_4 \\ a_5 & b_5 & c_5 & d_5 & e_5 & f_5 \\ a_6 & b_6 & c_6 & d_6 & e_6 & f_6 \end{bmatrix}. \tag{4}$$

This allows for the joint path (in the joint space) to be defined as

$$q(t) = a^6 + b^5 + c^4 + d^3 + e^2 + f. \tag{5}$$

After the generation, the iterative part begins. Each of the solutions gets their fitness—the quality of the solution estimated. In the presented case, the function for this is obtained as a model from the previously defined ML algorithm. Then, iteratively, solutions are selected randomly with a bias towards the higher-quality solutions [34,35]. These solutions have the evolutionary operations applied to them: crossover and mutation. Crossover (also refered to as recombination) selects chromosomes and pairs them to produce an offspring solution [36]. This involves swapping portions of their (individual values within the previously shown matrix) structure to create new chromosomes. In mutation, the offspring chromosomes may undergo random changes (mutations) in some of their elements to maintain genetic diversity within the population and to explore new areas of the search space. Finally, the solutions newly generated from this process are used to replace the old solutions. This process is repeated for either a fixed number of generations or until a satisfactory solution is reached.

The GA is a relatively simple algorithm that allows for optimization through the selection of higher quality solutions. There exist many algorithms that use this basic principle and expand upon it, two of which were tested in this research—differential evolution (DE) and memetic algorithm (MA).

DE works in the same manner as the GA, with the simple difference of the recombination and mutation process being replaced by the differential calculation [37]:

$$n_i = a_{i,1} + F \cdot (a_{i,2} - a_{i,3}), \tag{6}$$

where $n_i$ is the newly calculated solution, based on three randomly selected solutions $a_{i,x}$. $F$ is the differential weight that controls the influence of the calculated difference—in other words, it controls the distance from the original solution chosen, $a_{i,1}$. DE can be thought of as a special case of GA with a different type of recombination. The mutation is not included, as the randomness of the search for the solution space is introduced by differentiation between $a_{i,2}$ and $a_{i,3}$.

The MA is also an extension of the GA. Instead of changing the way the genes are recombined, the MA adds an additional step—local search. The main idea behind the MA is that the process GA uses for the search does not guarantee finding the optimal solution [38], and the solutions found can often be assumed to be near optimal [39]. Then, by using the domain-specific knowledge related to the problem that is being solved, an additional search is constructed, in order to find possible better solutions near the ones found by the GA. In this sense, the MA is a two stage process, in which the first stage consists of the classical GA application, while the second stage deals with the specific search of the local solution space surrounding the local solution [40]. In the presented research, this is performed by adjusting the parameters connected to those joints that had the highest influence on the total energy—information obtained from the analysis using the methods described in Section 2.2 The trajectory parameters, as given in Equations (2) and (3), are lowered with the idea of lowering the movement, speeds, and accelerations of that particular joint.

The GA functions by generating an initial population of candidate solutions—in the presented work, the paths are defined by their interpolation values. The fitness is then evaluated using the selected method for each individual solution, which starts the loop. While the loop can be exited in different ways, such as reaching the predetermined value or determining that there was no significant change in past iterations, this study utilizes a fixed number of iterations (called "generations"), iterating for 300 times across all of the elements in the population. If the number of generations has not been reached, the selection is performed. The selection, using roulette wheel selection, has a higher likelihood of selecting better-fit individuals (paths that use lower energy. Then, with different probabilities, either the mutation, crossover, or reproduction is performed, creating a new member of the

population, which is inserted into the population—generating the new population set for the given generation. The difference between the GA and MA is illustrated in the Figure 2. The added step (indicated with the red background) allows for the second search stage to be performed. While given generally here, this usually describes a localized deterministic search adjusted to the search space. The need for this second point arises from the nature of the GA. Combining the solutions using crossover allows for the wide search of the space, and while GA converges to a solution, it does not guarantee that the best solution will be found, as the stochastic nature of the search can skip over solutions [35]. The benefit of the added second stage in the MA is that it is relatively quick (the simplest search would simply include observing the immediate surrounding values in the search space). Due to this secondary search having constant or linear complexity ($O(1)$ or $O(n)$), it allows for a quick check of the local solution space (especially since it can be performed only for the best solution in the generation, if additional speed is necessary). Since the optimization of the GA is comparatively relatively long, adding the MA does not significantly lengthen the process, meaning that there is not a significant loss of performance even if the local search introduced by the MA does not determine a better solution [35].

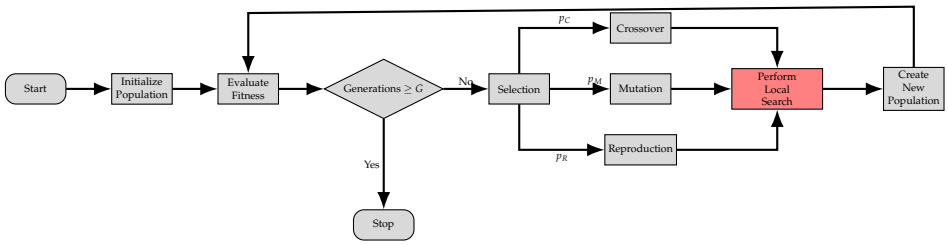



**Figure 2.** Scheme of the memetic algorithm used in the presented research.

## 3. Results

### *3.1. Regression Model Results*

The results of the models during the cross-validation process are given in Table 7.

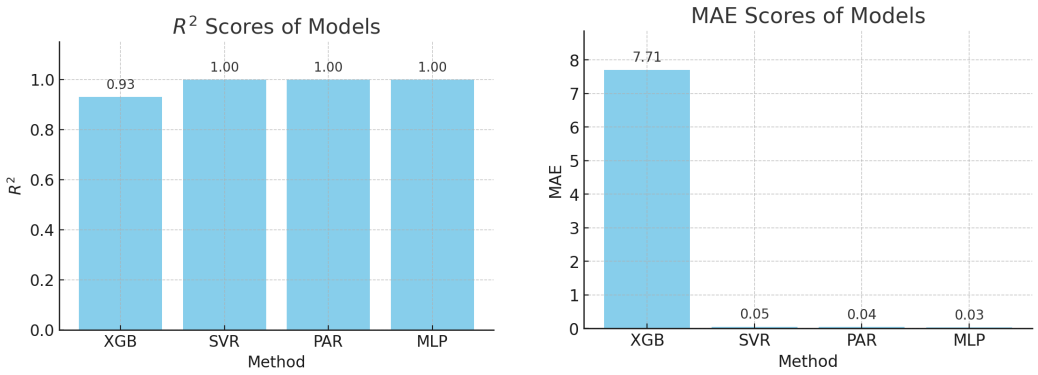
**Table 7.** Comparison of model performance metrics and optimized hyperparameters for each regression method. The table reports the mean coefficient of determination $\overline{R2}$, its standard deviation $\sigma_{\overline{R2}}$, the mean absolute error $\overline{MAE}$, and its standard deviation $\sigma_{\overline{MAE}}$ across validation folds. For each method, the optimal hyperparameters are shown: MLP (multilayer perceptron)—HLS: hidden layer sizes; $\phi$: activation function; $\alpha_{type}$: learning rate type; $\alpha_{init.}$: initial learning rate; Solver: optimization solver. PAR (passive-aggressive regressor)—C: regularization parameter; $\varepsilon$: tolerance margin; F.I.: fit intercept flag; $\varepsilon_{ins}$: $\varepsilon$-insensitive loss flag; Tol.: solver tolerance. SVR (support vector regressor)—C: penalty parameter; Degree: polynomial kernel degree; $\varepsilon$: epsilon-tube within which no penalty is applied; $\gamma$: kernel coefficient; Kernel: kernel type. XGB (extreme gradient boosting)—$X_L$: learning rate; $X_N$: number of estimators; $X_T$: tree depth; $\alpha$: learning rate; max(d): maximum tree depth; $n$: number of boosting rounds; $L_1$: L1 regularization weight; $L_2$: L2 regularization weight; Subs.: subsample ratio.

| **Method** | **Optimized Hyperparameters** | | | | | | | | | $\overline{R2}$ | $\sigma_{\overline{R2}}$ | $\overline{MAE}$ | $\sigma_{\overline{MAE}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MLP | HLS | | $\phi$ | $\alpha_{type}$ | $\alpha_{init.}$ | Solver | | | | 0.999 | 0.001 | 0.022 | 0.010 |
| | (10, 10) | | identity | const. | 0.1 | LBFGS | | | | | | | |
| PAR | $C$ | $\varepsilon$ | F.I. | $\varepsilon_{ins}$ | Tol. | | | | | 0.999 | 0.001 | 0.044 | 0.011 |
| | 0.1 | 0.1 | True | True | $1 \cdot 10^{-5}$ | | | | | | | | |
| SVR | $C$ | Degree | $\varepsilon$ | $\gamma$ | Kernel | | | | | 0.999 | 0.001 | 0.040 | 0.006 |
| | 10 | 2 | 0.1 | scale | linear | | | | | | | | |
| XGB | $X_L$ | $X_N$ | $X_T$ | $\alpha$ | max(d) | n | $L_1$ | $L_2$ | Subs. | 0.928 | 0.013 | 8.050 | 1.663 |
| | 1 | 1 | 1 | 0.1 | 6 | 100 | 0 | 0.5 | 0.5 | | | | |

The comparison of the regression models presented in Table 7 provides an overview of their predictive performance and the optimized hyperparameters obtained through the validation procedure. The evaluation was based on the mean coefficient of determination ($\overline{R^2}$), its standard deviation ($\sigma_{\overline{R^2}}$), the mean absolute error ($\overline{MAE}$), and the standard deviation of the absolute error ($\sigma_{\overline{MAE}}$). These metrics jointly quantify the average predictive accuracy, stability, and generalization capacity of each model across the validation folds. The MLP achieved the highest performance, with $\overline{R^2} = 0.999$ and $\sigma_{\overline{R^2}} = 0.001$, indicating that the model consistently reproduced the reference values with negligible deviation. The corresponding $\overline{MAE}$ of 0.022 and $\sigma_{\overline{MAE}}$ of 0.010 confirm the model's low prediction error and robustness. The optimized hyperparameters consisted of two hidden layers of ten neurons each (HLS = (10, 10)), an identity activation function, and a constant learning rate ($\alpha_{type}$ = constant) with an initial value of $\alpha_{init.} = 0.1$. The optimization was performed using the LBFGS solver, which is a quasi-Newton approach known for its high convergence efficiency on small data sets. The obtained configuration suggests that a relatively simple and linearly activated network was sufficient to fully model the target variable. The PAR exhibited nearly identical performance, also reaching $\overline{R^2} = 0.999$ and $\sigma_{\overline{R^2}} = 0.001$. Although the $\overline{MAE}$ of 0.044 was slightly higher than that of the MLP, the result still indicates a strong predictive capability. The optimal hyperparameters were $C = 0.1$ and $\varepsilon = 0.1$, with both the fit intercept and $\varepsilon$-insensitive loss options enabled. The tolerance of the solver was set to $1 \cdot 10^{-5}$, ensuring stable convergence. This configuration indicates that the model maintained a balance between aggressive updates and regularization, allowing for high precision while preserving robustness to data variations. The SVR demonstrated equally high performance, with $\overline{R^2} = 0.999$ and $\sigma_{\overline{R^2}} = 0.001$. The mean absolute error of 0.040 ($\sigma_{\overline{MAE}} = 0.006$) reflects a comparable level of predictive precision to the previously discussed models. The optimal hyperparameters included $C = 10$, a polynomial degree of 2, $\varepsilon = 0.1$, kernel coefficient $\gamma$ = scale, and a linear kernel. The selected configuration suggests that a linear mapping of the feature space was sufficient for achieving near-perfect prediction accuracy, further confirming the linearity of the underlying relationships in the data set. In contrast, the XGB model achieved a lower performance, with $\overline{R^2} = 0.928$ and $\sigma_{\overline{R^2}} = 0.013$, indicating reduced consistency across folds. The $\overline{MAE}$ of 8.050 and $\sigma_{\overline{MAE}}$ of 1.663 demonstrate a considerably higher average error and variability. The optimal parameters were defined by a learning rate $X_L = 1$, one estimator ($X_N = 1$), tree depth $X_T = 1$, $\alpha = 0.1$, maximum tree depth of 6, 100 boosting rounds ($n = 100$), $L_1 = 0$, $L_2 = 0.5$, and subsampling ratio of 0.5. This configuration represents a conservative setup, with strong regularization and shallow trees that likely limited the model's expressive power. The combination of a high learning rate and a minimal number of estimators resulted in a model prone to underfitting, which explains its comparatively weaker performance. Overall, the MLP, PAR, and SVR models demonstrated near-ideal regression behavior, characterized by minimal error and near-complete variance explanation, while the XGB model underperformed due to its limited complexity. The observed results imply that the data set exhibits an approximately linear dependency structure, effectively captured by linear and shallow nonlinear models, while more complex ensemble approaches provided no additional benefit under the given parameter constraints.

To confirm the model performance on the validation data, the models which were determined to have best performance have been tested on the previously collected test data, completely separate from the data that the models were trained on. The goal of this is to determine the performance of the model obtained with a given method and hyperparameters, as given in Table 7. As such, Figure 3 demonstrates the performance of these models, as evaluated with $R^2$, Figure 3a, and MAE, Figure 3b, metrics on the test set—evaluating the performance of the models as close as possible to what their performance should be on real data. The results from Figure 3 clearly indicate that the

SVR, PAR, and MLP models reached near-perfect performance, each achieving an $R^2$ value of 1.00 with corresponding mean absolute errors of 0.05, 0.04, and 0.03, respectively. The results from Figure 3 suggest that these models were capable of fully reproducing the target values with numerical precision. In contrast, the XGB model achieved a lower $R^2$ value of 0.93 and a significantly higher MAE of 7.71, indicating reduced accuracy and a considerably larger prediction deviation. The difference in performance between the boosting model and the other approaches implies that the underlying dependency in the data set is predominantly linear, favoring regression methods based on linear mappings or shallow network architectures. The obtained comparison confirms the observations reported in Table 3, where the MLP, PAR, and SVR models exhibited nearly ideal regression behavior, while the XGB configuration remained limited by underfitting.




(**a**) $R^2$ scores on the test data set (**b**) MAE scores on the test data set

**Figure 3.** Comparison of the best-performing regression models on the separate test data set.

Comparing the scores to scores obtained by other researchers who developed Energy-use prediction models, as provided in Table 2, the scores obtained by the best performing models were comparable or better. The study using $R^2$ as the metric achieved a score of 0.97 [11], while the best performing models presented in this study achieve scores of 0.99. As for the error scores, the normalized MAE is in the similar range to the models evaluated by error metrics and obtained [3,13,14], but it must be noted that all of those models use historical time-series data, which are not used in the presented research, which uses momentary values of physical quantities related to the robot arm movement and position, allowing for a wider model application.

### *3.2. Feature Importance Results*

The comparison of feature importance values obtained for the variable $E$ is presented in Figure 4. Figure 4 provides a logarithmic-scale visualization of the relative importance of all the considered input features across multiple evaluation methods, including the Mean Decrease in Impurity ($I_{MDI}$), permutation-based importance ($I_{perm}$), and three correlation-based measures: Pearson's ($\rho^P$), Spearman's ($\rho^S$), and Kendall's ($\rho^K$) coefficients. Such a combined representation enables a comprehensive assessment of both model-driven and statistical dependencies between the predictors and the target variable.

The results indicate that the positional and angular variables—particularly $x$, $y$, and $\psi$—exhibit consistently higher importance values across all evaluation metrics, suggesting that they contribute most significantly to the prediction of $E$. These features maintain high importance levels in both impurity-based and correlation-based analyses, implying that the target is primarily influenced by geometric and orientation-related parameters. The joint variables $q_1$ through $q_6$ also demonstrate non-negligible contributions, although with noticeable variation across methods, reflecting their indirect or secondary influence on the predicted output.

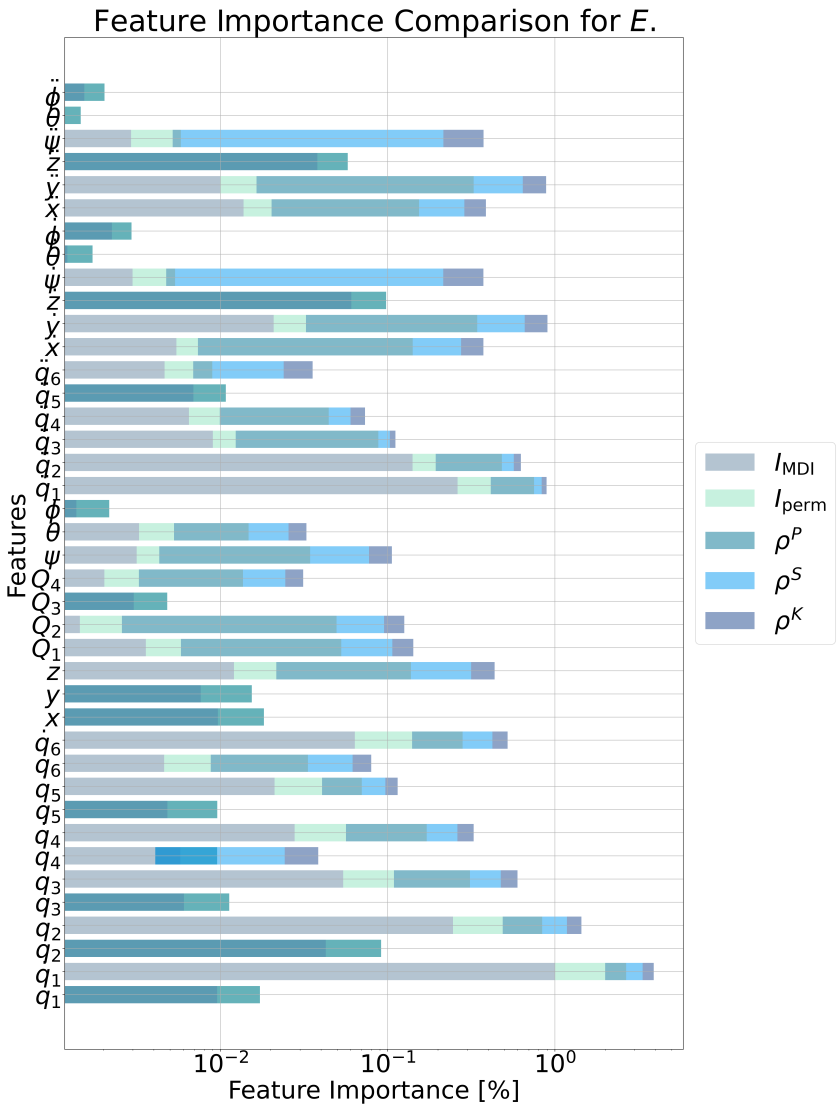



**Figure 4.** The stacked feature importance of each individual variable.

Among the correlation-based measures, Pearson's and Spearman's coefficients show the strongest alignment, indicating that the relationship between the features and the output is predominantly monotonic and approximately linear. Kendall's $\rho^K$ values follow similar trends but remain slightly lower in magnitude, which is expected due to its stricter concordance-based formulation. The permutation importance values $I_{perm}$ largely confirm the same pattern observed in $I_{MDI}$, emphasizing the robustness of the identified key predictors.

Overall, the presented comparison highlights that the predictive behavior of the models is primarily governed by translational and rotational components, while the joint-space variables contribute less directly. The consistency between multiple importance evaluation methods further reinforces the interpretability of the regression models and the reliability of the observed feature relevance structure.

What is important to note is that the joints, whose parameters show the highest influence on energy, are joints $q_1$ and $q_2$ (visible near the bottom of the Figure 4). This indicates that, by controlling the speeds of those individual joints by specifically lowering the parameters controlling their trajectories in joint space, the largest influence should be achievable. For this reason, the implementation of the MA algorithm that will be used in the presented research will aim to lower the parameters connected to those two joints.

### 3.3. Comparison of Evolutionary Algorithms

The results presented in Table 8 summarize the performance of the GA under different recombination strategies and parameter configurations. Three recombination types were examined: random, average, and differential. For each configuration, the probability of crossover $P$, mutation $P(M)$, and, in the case of differential recombination, the scaling factor $F$, were varied to observe their influence on optimization efficiency. The Improvement represents the mean percentage gain achieved across $N = 100$ independent runs, while $\sigma$ denotes the corresponding standard deviation, indicating the consistency of results.

From the data, it can be seen that the random and average recombination approaches generally yielded higher improvements, exceeding 40% in several configurations, whereas the differential recombination achieved more moderate yet consistent results, with improvements in the range of 35–38%. The highest recorded improvement was 50.15% for the random recombination with $P = 0.95$ and $P(M) = 0.05$, while the lowest was 33.01% for random recombination with lower mutation probability. The standard deviation values remain relatively low (0.20–0.27), suggesting that the algorithm maintained stable convergence across repeated runs. Overall, the results indicate that a higher mutation rate tends to improve the performance for random and average recombination, while the differential approach demonstrates robustness to parameter variations.

**Table 8.** The scores of the GA algorithms for given parameters. Recombination—the type of recombination used (random, average or differential). $P$—the probability of recombination happening for a given iteration over the population, $P(M)$—the probability of a mutation happening for a given iteration over the population, Improvement—the average improvement across $N = 100$ runs, $\sigma$—standard deviation.

| Recombination | P | P(M) | F | Improvement [%] | $\sigma$ |
|---|---|---|---|---|---|
| Random | 0.95 | 0.05 | - | 50.15 | 0.23 |
| Random | 0.95 | 0.01 | - | 33.61 | 0.24 |
| Random | 0.90 | 0.05 | - | 38.03 | 0.27 |
| Random | 0.90 | 0.01 | - | 33.01 | 0.22 |
| Average | 0.95 | 0.05 | - | 44.42 | 0.22 |
| Average | 0.95 | 0.01 | - | 33.19 | 0.23 |
| Average | 0.90 | 0.05 | - | 48.85 | 0.22 |
| Average | 0.90 | 0.01 | - | 33.78 | 0.20 |
| Differential | 0.95 | - | 0.05 | 37.84 | 0.27 |
| Differential | 0.95 | - | 0.15 | 36.78 | 0.26 |
| Differential | 0.90 | - | 0.05 | 35.91 | 0.25 |
| Differential | 0.90 | - | 0.15 | 36.18 | 0.26 |

Based on these results (as shown in Table 8), the GA with randomized recombination at $P = 0.95$ and $P(M) = 0.05$ was selected as the basis of the MA algorithm. The results shown in Figure 5 illustrate the convergence dynamics of the MA when an informed local search strategy is employed. Each gray line corresponds to a single optimization path out of the 100 randomly initialized runs, while the red curve shows the mean performance across all runs. The overall trend demonstrates a steep initial reduction in fitness, occurring during the first 20–40 generations, which suggests that the algorithm quickly identifies promising regions of the search space. After this phase, the rate of improvement slows, indicating convergence toward local optima and a stabilization of the solution quality.

The presence of overlapping fitness trajectories in the early generations reflects the stochastic nature of the population-based search, while the convergence of lines in later generations implies the strong exploitation of high-performing individuals. The low variance observed after approximately 150 generations confirms that the informed local search ef-

fectively enhances the convergence stability, minimizing performance fluctuations between independent runs. These findings confirm that incorporating a local search within the evolutionary process improves the convergence efficiency while maintaining the robust exploration of the solution space.

Table 9 presents the optimization results of a few randomly selected paths, comparing the energy consumption that a linear path between the same two points in the space would take, against the energy consumption of the interpolated path whose parameters were obtained via the MA. The optimization process was tested 100 times, repeating the optimization between two randomly selected points in the space (using the same methodology as the random selection of paths for obtaining the training data set), with the average improvement being 58.16%, with a standard deviation of 0.22%.

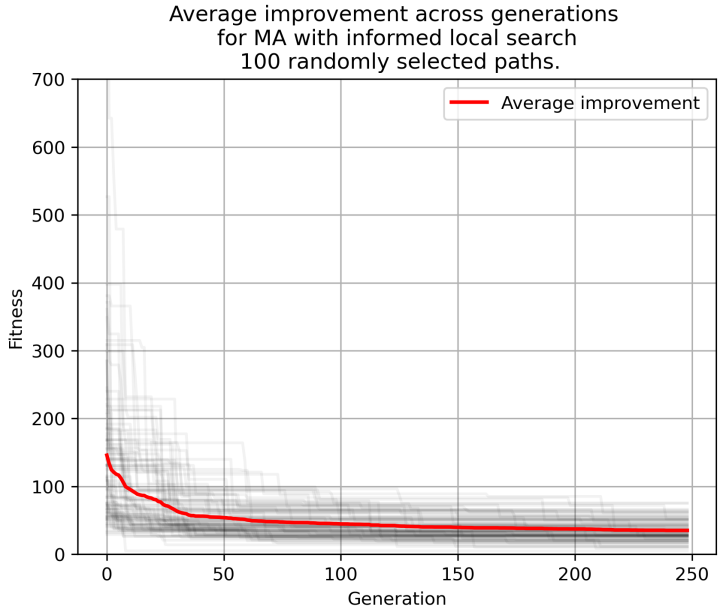



**Figure 5.** Average improvement across generations for the memetic algorithm (MA) with an informed local search, calculated over 100 randomly selected evolutionary paths. The gray lines represent individual fitness trajectories, while the red line denotes the average fitness value across generations. A rapid decrease in fitness is observed within the initial iterations, followed by gradual convergence toward a stable minimum. This behavior indicates that the algorithm effectively exploits the search space early on, with subsequent refinements leading to diminishing returns in improvement.

**Table 9.** Comparison of few randomly selected paths, between the energy use of a linear point-to-point movement and the optimized path. Gene—the path parameters obtained via optimization, $q^S$—starting positions in joint space, $q^E$—ending positions in joint space, $E_L$—energy of the linear path, $E_{\text{optimized}}$—energy use of the optimized path, Δ—improvement with optimization.

| Gene | $q^S$ | $q^E$ | $E_L$ | $E_{\text{optimized}}$ | Δ [%] |
|---|---|---|---|---|---|
| 0.00, 0.62, −0.12, −0.12, −1.18, 0.00, | −14.32 | 120.32 | 75.21 | 24.52 | 67.39 |
| −0.00, −0.10, −0.08, 0.08, 0.00, −0.14, | −22.92 | 131.78 | | | |
| 9.45, 1.60, 5.85, 1.25, 7.85, −4.00, | 5.73 | 123.19 | | | |
| 3.30, 7.00, −4.70, −9.20, −3.85, −5.05, | 40.11 | 140.37 | | | |
| 8.15, 1.30, 7.60, 3.65, −7.50, −7.65, | 48.70 | 114.59 | | | |
| 1.00, 6.75, 4.30, 7.45, −5.55, 6.40 | −54.43 | 166.16 | | | |
| 1.12, 1.42, 0.04, 0.74, −0.74, −0.08, | 28.65 | 166.16 | 37.57 | 19.44 | 48.25 |
| −0.16, 0.65, −0.12, −1.06, −0.02, 0.13, | −28.65 | 117.46 | | | |
| −1.25, 8.75, 5.20, 1.40, 2.35, 5.25, | 37.24 | 117.46 | | | |
| −9.40, 0.25, −8.30, 0.25, −2.95, 3.20, | 20.05 | 163.29 | | | |
| 3.05, −5.80, 1.35, −4.35, −6.25, 3.60, | 51.57 | 166.16 | | | |
| 8.25, −7.40, 0.00, −8.95, 4.90, −1.85 | 37.24 | 114.59 | | | |

**Table 9.** *Cont.*

| Gene | $q^S$ | $q^E$ | $E_L$ | $E_{\text{optimized}}$ | Δ [%] |
|---|---|---|---|---|---|
| −0.24, −0.00, 0.98, 0.00, −1.20, 0.43, | 22.91 | 114.59 | 92.20 | 25.16 | 72.71 |
| 0.03, −0.00, −0.85, 1.15, −1.02, 0.38, | −40.11 | 120.32 | | | |
| 2.20, 3.20, 6.60, 2.00, −6.45, 5.95, | 22.92 | 137.51 | | | |
| −3.25, −3.70, 3.90, −1.05, −1.35, −0.05, | −20.05 | 126.05 | | | |
| 4.30, 8.05, 2.95, 1.05, −3.55, −5.20, | −25.78 | 128.92 | | | |
| −9.70, −2.15, −1.25, 8.25, −1.70, 7.45 | −20.05 | 114.59 | | | |
| −0.01, 0.94, 0.00, 0.00, −0.08, 0.15, | −42.97 | 143.24 | 38.76 | 18.00 | 53.56 |
| 0.01, 0.00, 0.00, 0.13, 0.18, 0.00, | 28.65 | 146.10 | | | |
| 6.70, −8.45, −1.30, 0.20, −4.10, −8.00, | 31.51 | 154.69 | | | |
| 4.50, 0.35, −8.25, 7.00, −8.50, 8.65, | 51.57 | 117.46 | | | |
| 1.85, 4.60, −1.90, 4.45, −3.55, 7.45, | −40.11 | 123.19 | | | |
| 6.15, 5.95, −6.50, −7.20, −9.70, 3.90 | 22.92 | 143.24 | | | |
| −0.00, −0.02, 0.00, −0.97, 0.10, −0.19, | −40.11 | 143.24 | 38.39 | 29.26 | 23.78 |
| 0.00, 1.00, −0.01, 0.08, 0.00, 0.00, | −22.92 | 154.69 | | | |
| −7.95, 5.90, 4.70, −9.25, −7.45, −7.05, | 51.57 | 140.37 | | | |
| 8.80, −8.80, −1.35, 7.15, 0.25, 1.45, | 8.59 | 137.51 | | | |
| −4.40, −0.40, −0.15, 5.60, 4.65, −3.55, | 25.78 | 134.65 | | | |
| −3.15, −8.65, −9.05, 6.00, −1.55, −1.70 | 20.06 | 131.78 | | | |

Figure 6 shows the improvement between the best performing GA, as given in Table 8 and the MA, which applies the local search. Both algorithms were tested on 100 randomly generated runs. As the image clearly shows, the addition of the local search to the GA shows a clear improvement of approximately 8% on average. Considering the simplicity and low computational complexity of the local search, this improvement is definitely notable and justifies the usage of the MA over the simple GA for the given problem. Comparing the presented approach with the scores obtained by other researchers on similar problems (provided in Table 1), the results obtained by both the single-stage methodology and the two-stage MA are shown to be at least competitive if not better than the results obtained with the methodologies presented by other researchers in the literature review. The best score presented was a 40% improvement compared to the initial path, obtained by [2], using a combination of GA, PSO, and Bezier curves for path optimization. The scores obtained in this study were 50.15% for the GA and 58.16% for the MA, both of which are notably higher than the presented state-of-the-art score.

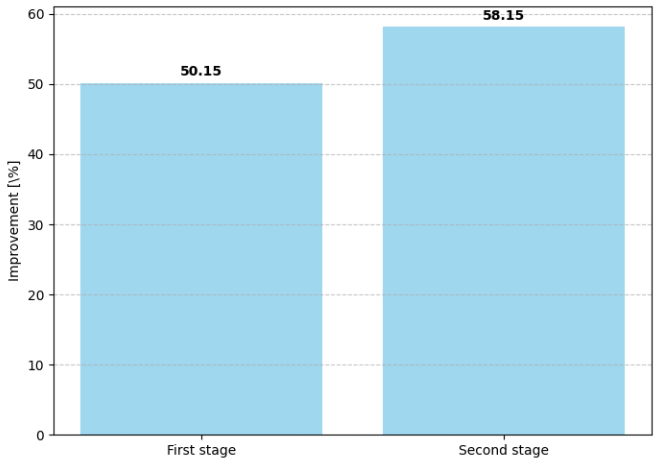



**Figure 6.** Comparison of different stages based on improvement metrics, with the first stage representing the average improvement of 100 test runs for GA with random recombination, and the second stage representing the average improvement of 100 test runs for MA based on the same GA, but extended with a localized search.

## 4. Conclusions

The presented work has shown the process in which an MA algorithm, consisting of an MLP network for the energy cost estimation of a robot path, is developed. The developed algorithm uses the GA with random recombination as the basis for the first stage and performs a local search based on feature importance determined from the data in the second stage.

The performed tests show that not only does the GA itself optimize the point-to-point paths well, but the usage of the MA can lower the energy use by 58% on average) when compared to a linear path between the same two points.

It is important to note that the specified technique has two main limitations. First, it is only applicable to point-to-point paths, where movement in between two points is free. Second, the fitness function based on the MLP model would probably require re-training for application on different IRs, but this has not been tested in the current work.

It must be noted that the different scenarios and environment changes could require adjustments to the proposed method. Further testing, for example in areas with obstacles, leveraging more advanced IRs with higher sensing capabilities, is planned as a part of future work.

**Supplementary Materials:** The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/machines14010035/s1, Table S1: PROCESSED_real; Table S2: TEST.

**Author Contributions:** Conceptualization, S.B.Š. and N.A.; methodology, S.B.Š.; software, I.L.; validation, I.L., D.F. and N.A.; formal analysis, D.F. and N.A.; investigation, S.B.Š.; resources, D.F.; data curation, S.B.Š.; writing—original draft preparation, S.B.Š. and N.A.; writing—review and editing, I.L. and D.F.; visualization, I.L.; supervision, D.F.; project administration, I.L.; funding acquisition, D.F. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded in part by the following projects: SPIN projects IP.1.03.1020, IP.1.03.0158, and IP.1.1.03.0039; UNIRI projects UNIRI-IZ-25-220 and UNIRI-IZ-25-6; University North project UNIN-TEH-25-1-8; the EU NextGeneration under the Juraj Dobrila University of Pula institutional research project number IIP_010144 and IIP_010136; and the EC Digital Europe Programme EDIH Adria 101083838.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in this study are included in the article/Supplementary Material. Further inquiries can be directed to the corresponding author.

**Acknowledgments:** During the preparation of this manuscript/study, the authors used ChatGPT5 and Github Copilot GPT-5 mini for the purposes of generating and editing parts of the text and code generation. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

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

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| ANN | Artificial Neural Network |
| BN | Batch Normalization |
| DE | Differential Evolution |
| GA | Genetic Algorithm |
| GA+VNS | Genetic Algorithm combined with Variable Neighborhood Search |
| IMDI | Impurity-Based Importance (variant of MDI) |
| IR | Industrial Robot |
| IRM | Industrial Robotic Manipulator |
| LBFGS | Limited-memory Broyden–Fletcher–Goldfarb–Shanno |

| | |
|---|---|
| LSTM | Long Short-Term Memory |
| MA | Memetic Algorithm |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| MDI | Mean Decrease in Impurity |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| PAR | Passive-Aggressive Regressor |
| PPM | Planar Parallel Manipulator |
| PSO | Particle Swarm Optimization |
| R2 | Coefficient of Determination |
| RMSE | Root Mean Square Error |
| SMB | Serial Measurement Board |
| SVR | Support Vector Regression |
| SVM | Support Vector Machine |
| TCP | Tool Center Position |
| VNS | Variable Neighborhood Search |
| XGB | Extreme Gradient Boosting |
| $\sigma$ | Standard Deviation |

## Appendix A

*Appendix A.1*

The code below, given in RAPID programming language, was used during data collection in order to move the IR during measurement.

**Listing A1.** RAPID module used for generating random robot joint configurations and executing motion commands.

```
MODULE~Module1

! Define the upper and lower bounds of joints, grabbed by observing
    kinematics
CONST num J1_LO := -120;
CONST num J1_HI := 120;
CONST num J2_LO := -55;
CONST num J2_HI := 70;
CONST num J3_LO := -100;
CONST num J3_HI := 40;
CONST num J4_LO := -160;
CONST num J4_HI := 160;
CONST num J5_LO := -120;
CONST num J5_HI := 120;
CONST num J6_LO := -400;
CONST num J6_HI := 400;

! Possible number of zones and speeds
CONST num SPEED_NUMBER := 17;
CONST num ZONE_NUMBER := 14;

! Possible speed and zone values
CONST speeddata SPEED_ARR{SPEED_NUMBER} := [
    v100, v150, v200, v300, v400, v500,
    v600, v800, v1000, v1500, v2000, v2500,
    v3000, v4000, v5000, v6000, v7000
];

CONST zonedata ZONE_ARR{ZONE_NUMBER} := [
    fine, z0, z1, z5, z10, z20, z30, z40,
    z50, z60, z80, z100, z150, z200
```

```
];

! For normalizing RAND() to range <0,1>
CONST num RAND_MAX := 32767;

! Define variables for storing generated random values
VAR num J1;
VAR num J2;
VAR num J3;
VAR num J4;
VAR num J5;
VAR num J6;
VAR speeddata V;
VAR zonedata Z;

VAR string starttime;
VAR string endtime;

VAR num position;
VAR num speed;
VAR num torque;
VAR num exttorque;

! Define number of joint positions to be tested
CONST num NUM_SIMULATION_COUNT := 1000;

PROC main()
    starttime := CTime();

    FOR i FROM 0 TO NUM_SIMULATION_COUNT DO
        ! Write the simulation step for info
        TPWrite("INFO Simulation count=" + NumToStr(i, 0) + "/" +
            NumToStr(NUM_SIMULATION_COUNT, 0));

        ! Randomly select joint 1--6, speed, and~zone
        J1 := ((RAND() / RAND_MAX) * (J1_HI - J1_LO)) + J1_LO;
        J2 := ((RAND() / RAND_MAX) * (J2_HI - J2_LO)) + J2_LO;
        J3 := ((RAND() / RAND_MAX) * (J3_HI - J3_LO)) + J3_LO;
        J4 := ((RAND() / RAND_MAX) * (J4_HI - J4_LO)) + J4_LO;
        J5 := ((RAND() / RAND_MAX) * (J5_HI - J5_LO)) + J5_LO;
        J6 := ((RAND() / RAND_MAX) * (J6_HI - J6_LO)) + J6_LO;

        V := SPEED_ARR{1 + ROUND((RAND() / RAND_MAX) * (SPEED_NUMBER - 1)
            )};
        Z := ZONE_ARR{1 + ROUND((RAND() / RAND_MAX) * (ZONE_NUMBER - 1))
            };

        ! Move to random absolute joint positions
        MoveAbsJ [[J1, J2, J3, J4, J5, J6], [9E9, 9E9, 9E9, 9E9, 9E9, 9E9
            ]], V, Z, tool0;

        ! Optional: monitor torque and power
        ! TPWrite(NumToStr(speed, 2) + "," + NumToStr(torque, 2) + "," +
            NumToStr(torque * speed, 2));
    ENDFOR

    TPWrite("Start time: " + starttime);
    TPWrite("End time: " + CTime());

    EXIT;
ENDPROC

ENDMODULE
```

*Appendix A.2*

The code below was used in parallel with the code above, in order to note the actual values as measured by the IR controller and save them to a file for later analysis and use.

**Listing A2.** RAPID module for real-time measurement and logging of joint and TCP data during execution.

```
MODULE measure

! Define variables for storing position, speed, and torques
VAR num POS1;
VAR num POS2;
VAR num POS3;
VAR num POS4;
VAR num POS5;
VAR num POS6;

VAR num SPD1;
VAR num SPD2;
VAR num SPD3;
VAR num SPD4;
VAR num SPD5;
VAR num SPD6;

VAR num TOR1;
VAR num TOR2;
VAR num TOR3;
VAR num TOR4;
VAR num TOR5;
VAR num TOR6;

VAR num ETOR1;
VAR num ETOR2;
VAR num ETOR3;
VAR num ETOR4;
VAR num ETOR5;
VAR num ETOR6;

VAR IODEV logfile;

VAR robtarget TCP;
VAR clock sim_clock;
VAR num time;

PROC main()
    Open "HOME:" \File := CDate() + "-" +
        NumToStr(GetTime(\Hour), 0) + "-" +
        NumToStr(GetTime(\Min), 0) + "-" +
        NumToStr(GetTime(\Sec), 0) + "_MEASUREMENT.CSV",
        logfile \Write;

    Write logfile, "t,q1,dq1,tau1,q2,dq2,tau2," \NoNewLine;
    Write logfile, "q3,dq3,tau3,q4,dq4,tau4,q5,dq5,tau5," \NoNewLine;
    Write logfile, "q6,dq6,tau6,x,y,z,e1,e2,e3,e4,psi,theta,phi";

    ClkReset sim_clock;
    ClkStart sim_clock;

    WHILE TRUE DO
        GetJointData \MechUnit := ROB_1, 1 \Position := POS1 \Speed :=
            SPD1 \Torque := TOR1 \ExtTorque := ETOR1;
        GetJointData \MechUnit := ROB_1, 2 \Position := POS2 \Speed :=
            SPD2 \Torque := TOR2 \ExtTorque := ETOR2;
```

```
        GetJointData \MechUnit := ROB_1, 3 \Position := POS3 \Speed :=
            SPD3 \Torque := TOR3 \ExtTorque := ETOR3;
        GetJointData \MechUnit := ROB_1, 4 \Position := POS4 \Speed :=
            SPD4 \Torque := TOR4 \ExtTorque := ETOR4;
        GetJointData \MechUnit := ROB_1, 5 \Position := POS5 \Speed :=
            SPD5 \Torque := TOR5 \ExtTorque := ETOR5;
        GetJointData \MechUnit := ROB_1, 6 \Position := POS6 \Speed :=
            SPD6 \Torque := TOR6 \ExtTorque := ETOR6;

        ! Read TCP position, x, y, z
        TCP := CRobT(\Tool := tool0 \WObj := wobj0);

        time := ClkRead(sim_clock);

        Write logfile, NumToStr(time, 5) + "," +
                       NumToStr(POS1, 5) + "," + NumToStr(SPD1, 5) + "," +
                            NumToStr(TOR1, 5) + "," \NoNewLine;
        Write logfile, NumToStr(POS2, 5) + "," + NumToStr(SPD2, 5) + ","
            + NumToStr(TOR2, 5) + "," \NoNewLine;
        Write logfile, NumToStr(POS3, 5) + "," + NumToStr(SPD3, 5) + ","
            + NumToStr(TOR3, 5) + "," \NoNewLine;
        Write logfile, NumToStr(POS4, 5) + "," + NumToStr(SPD4, 5) + ","
            + NumToStr(TOR4, 5) + "," \NoNewLine;
        Write logfile, NumToStr(POS5, 5) + "," + NumToStr(SPD5, 5) + ","
            + NumToStr(TOR5, 5) + "," \NoNewLine;
        Write logfile, NumToStr(POS6, 5) + "," + NumToStr(SPD6, 5) + ","
            + NumToStr(TOR6, 5) + "," \NoNewLine;

        Write logfile, NumToStr(TCP.trans.x, 5) + "," +
                       NumToStr(TCP.trans.y, 5) + "," +
                       NumToStr(TCP.trans.z, 5) + "," \NoNewLine;

        Write logfile, NumToStr(TCP.rot.q1, 5) + "," +
                       NumToStr(TCP.rot.q2, 5) + "," +
                       NumToStr(TCP.rot.q3, 5) + "," +
                       NumToStr(TCP.rot.q4, 5) + "," \NoNewLine;

        Write logfile, NumToStr(EulerZYX(\X, TCP.rot), 5) + "," +
                       NumToStr(EulerZYX(\Y, TCP.rot), 5) + "," +
                       NumToStr(EulerZYX(\Z, TCP.rot), 5);

        ! Pause for 1/40 s
        WaitTime 0.025;
    ENDWHILE

    Close logfile;
ENDPROC

ENDMODULE
```

## References


1. International Federation of Robotics. *World Robotics 2024—Press Conference Presentation*; Technical Report; Presentation Slides; International Federation of Robotics (IFR): Frankfurt, Germany, 2024.
2. Vysocký, A.; Papřok, R.; Šafařík, J.; Kot, T.; Bobovský, Z.; Novák, P.; Snášel, V. Reduction in robotic arm energy consumption by particle swarm optimization. *Appl. Sci.* **2020**, *10*, 8241. [CrossRef]
3. Garriz, A.; Domingo, C. Trajectory Optimization in Terms of Energy and Performance of an Industrial Robot in the Manufacturing Industry. *Sensors* **2022**, *22*, 7538. [CrossRef]
4. Shrivastava, A.; Dalla, P. Failure control and energy optimization of multi-axes space manipulator through genetic algorithm approach. *J. Braz. Soc. Mech. Sci. Eng.* **2021**, *43*, 445. [CrossRef]
5. Nonoyama, Y.; Matsuda, S.; Takahashi, R. Energy-Efficient Robot Configuration and Motion Planning Using Genetic Algorithm and Particle Swarm Optimization. *Energies* **2022**, *15*, 2074. [CrossRef]

6. Luneckas, M.; Bacevičius, L.; Damaševičius, R. Hexapod Robot Gait Switching for Energy Consumption and Cost of Transport Management Using Heuristic Algorithms. *Appl. Sci.* **2021**, *11*, 1139. [CrossRef]
7. Lu, Y.; Wang, Y.; Chen, X. Human–robot collaborative scheduling in energy-efficient welding shop. *IEEE Trans. on Ind. Inf.* **2023**, *20*, 936–971. [CrossRef]
8. Baressi Šegota, S.; Horvat, T.; Grbac, I. Path optimization of robotic manipulators using differential evolution for improved energy efficiency. *Int. J. Adv. Robot. Syst.* **2020**, *17*, 1–12. [CrossRef]
9. Jin, Y.; Olhofer, M.; Sendhoff, B. A framework for evolutionary optimization with approximate fitness functions. *IEEE Trans. Evol. Comput.* **2002**, *6*, 481–494. [CrossRef]
10. Gadaleta, M.; Berselli, G.; Pellicciari, M.; Grassia, F. Extensive experimental investigation for the optimization of the energy consumption of a high payload industrial robot with open research dataset. *Robot. Comput.-Integr. Manuf.* **2021**, *68*, 102046. [CrossRef]
11. Zhang, M.; Yan, W. A data-driven method for optimizing the energy consumption of industrial robots. *Energy Rep.* **2021**, *7*, 8747–8756.[CrossRef]
12. Gao, Y.; Li, J.; Sun, B. Energy optimization of a planar parallel manipulator using long short-term memory network. *IEEE Access* **2020**, *8*, 175964–175972.
13. Lin, X.; Zhang, K.; Luo, W. BN-LSTM-based energy consumption modeling approach for an industrial robot manipulator. *Robot. Comput.-Integr. Manuf.* **2024**, *85*, 102629. [CrossRef]
14. Zhang, M.; Jiang, H.; Wang, X.; Chen, J. An integrated deep-learning-based approach for energy consumption prediction of machining systems. *Sustainability* **2023**, *15*, 5781. [CrossRef]
15. Jaramillo-Morales, J.; López, C.; Suárez, J. Energy estimation for differential drive mobile robots on straight and rotational trajectories. *Int. J. Adv. Rob. Sys.* **2020**, *17*, 880–888. [CrossRef]
16. Baressi Šegota, S.; Anđelić, N.; Šercer, M.; Meštrić, H. Dynamics modeling of industrial robotic manipulators: A machine learning approach based on synthetic data. *Mathematics* **2022**, *10*, 1174. [CrossRef]
17. Cardona, M.; Serrano, F.E. Lagrange-Euler Approach. In *Rehabilitation Robotics: Kinematics, Dynamics, and Control Techniques*; Springer: Berlin/Heidelberg, Germany, 2025; pp. 77–89.
18. Adam, M.S.; Nordin, R.; Abdullah, N.F.; Abu-Samah, A.; Amodu, O.A.; Alsharif, M.H. Optimizing disaster response through efficient path planning of mobile aerial base station with genetic algorithm. *Drones* **2024**, *8*, 272. [CrossRef]
19. Dax, M.; Green, S.R.; Gair, J.; Gupte, N.; Pürrer, M.; Raymond, V.; Wildberger, J.; Macke, J.H.; Buonanno, A.; Schölkopf, B. Real-time inference for binary neutron star mergers using machine learning. *Nature* **2025**, *639*, 49–53. [CrossRef]
20. ABB Robotics. *RobotStudio Operating Manual*; Operating Manual for ABB RobotStudio, Document ID 3HAC032104-001, Revision V; ABB Group: Zurich, Switzerland, 2017.
21. ABB Robotics. *Technical Reference Manual: RAPID Instructions, Functions and Data Types*; Technical Reference for the RAPID Programming Language, Document ID 3HAC050917-001, Revision F, RobotWare 6.06; ABB Group: Västerås, Sweden, 2017.
22. Ostovar, A.; Davari, D.D.; Dzikuć, M. Determinants of design with multilayer perceptron neural networks: A comparison with logistic regression. *Sustainability* **2025**, *17*, 2611. [CrossRef]
23. Mekala, S.; Dasari, K.; Katta, D. DNS DDoS Amplification Attack Detection Using Multi-Layer Perceptron Classification Algorithm. In Proceedings of the 2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC), Gwalior, India, 27–28 July 2024; pp. 1355–1360.
24. Ren, X.; Wang, N.; Pan, J.; Bu, L. Combining style generative adversarial networks with particle swarm optimisation-support vector regression to design affective social robot for public health intervention. *J. Eng. Des.* **2025**, *36*, 160–190. [CrossRef]
25. Wang, Y.; Hong, M.; Chai, H.; Zhang, Y.; Wang, G.; Wu, C.; Guo, M. Research on Foot Contact State Detection Technology of Wheel-Legged Robot. *Sensors* **2024**, *24*, 3956. [CrossRef]
26. Alqarni, M.; Ashour, A.A.; Shafie, A.; Alqarni, A.; Felemban, M.F.; Shukr, B.S.; Alzubaidi, M.A.; Algahtani, F.S. Intelligence computational analysis of letrozole solubility in supercritical solvent via machine learning models. *Sci. Rep.* **2024**, *14*, 21677. [CrossRef]
27. Rahaman, M.; Cajes, N.; Gupta, B.B.; Nedjah, N.; Chui, K.T. Real-Time Machine Learning Techniques in Robotics for Detecting Phishing Attacks. In Proceedings of the 2025 IEEE International Conference on Real-time Computing and Robotics (RCAR), Toyama, Japan, 1–6 June 2025; pp. 1124–1129.
28. Khajezadeh, H.; Pour, M.S.; Manthouri, M. Effective Detection and Classification of Robot Execution Failures using Extreme Gradient Boosting (XGboost) Algorithm. In Proceedings of the IEEE 2024 10th International Conference on Control, Instrumentation and Automation (ICCIA), Kashan, Iran, 5–7 November 2024; pp. 1–6.
29. Kaushal, A.; Sharma, A.K.; Gupta, K. Decentralised coordination in swarm robots through xgboost-enhanced colour light communication. *Arab. J. Sci. Eng.* **2024**, *49*, 16253–16269. [CrossRef]
30. He, Y.X.; Lyu, S.H.; Jiang, Y. Interpreting deep forest through feature contribution and mdi feature importance. *ACM Trans. Knowl. Discov. Data* **2024**, *20*, 15. [CrossRef]
31. Geeitha, S.; Ravishankar, K.; Cho, J.; Easwaramoorthy, S.V. Integrating cat boost algorithm with triangulating feature importance to predict survival outcome in recurrent cervical cancer. *Sci. Rep.* **2024**, *14*, 19828. [CrossRef]

32. O'brien, P.C. Comparing two samples: Extensions of the t, rank-sum, and log-rank tests. *J. Am. Stat. Assoc.* **1988**, *83*, 52–61.
33. Ragano, A.; Martinez, H.B.; Hines, A. Beyond Correlation: Evaluating Multimedia Quality Models with the Constrained Concordance Index. *IEEE Trans. Multimed.* **2025**, *27*, 5604–5616. [CrossRef]
34. Alhijawi, B.; Awajan, A. Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evol. Intell.* **2024**, *17*, 1245–1256. [CrossRef]
35. Baressi Šegota, S. Determining the Energy-Optimal Path of Six-Axis Industrial Robotic Manipulators Using Machine Learning and Memetic Algorithms. Ph.D. Thesis, Faculty of Engineering, University of Rijeka, Rijeka, Croatia, 2025.
36. Londe, M.A.; Pessoa, L.S.; Andrade, C.E.; Resende, M.G. Biased random-key genetic algorithms: A review. *Eur. J. Oper. Res.* **2025**, *321*, 1–22. [CrossRef]
37. Pesaramelli, R.S.; Sujatha, B. Principle correlated feature extraction using differential evolution for improved classification. *Aip Conf. Proc.* **2024**, *2919*, 070001.
38. Alexakis, K.; Benekis, V.; Kokkinakos, P.; Askounis, D. Genetic algorithm-based multi-objective optimisation for energy-efficient building retrofitting: A systematic review. *Energy Build.* **2025**, *328*, 115216.[CrossRef]
39. Mahmud, T.; Datta, N.; Chakma, R.; Das, U.K.; Aziz, M.T.; Islam, M.; Salimullah, A.H.M.; Hossain, M.S.; Andersson, K. An approach for crop prediction in agriculture: Integrating genetic algorithms and machine learning. *IEEE Access* **2024**, *12*, 173583–173598. [CrossRef]
40. Ab Wahab, M.N.; Nazir, A.; Khalil, A.; Ho, W.J.; Akbar, M.F.; Noor, M.H.M.; Mohamed, A.S.A. Improved genetic algorithm for mobile robot path planning in static environments. *Expert Syst. Appl.* **2024**, *249*, 123762. [CrossRef]


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.