Adaptive Modular Differential Evolution Framework for Robotic Trajectory Optimization

Jan Fiala, Martin Juříček, and Jakub Kůdela*
Institute of Automation and Computer Science, Brno University of Technology
Technicka 2, 621 00 Brno, Czech Republic
Email: 217141@vutbr.cz, 200543@vutbr.cz, Jakub.Kudela@vutbr.cz*

***Abstract*—Optimization of robotic arm trajectories represents a complex computational challenge critical for efficient and safe automation. While traditional metaheuristic approaches have achieved notable success, they often require extensive manual parameter tuning and lack robustness when problem characteristics change. This work introduces a novel approach to robotic trajectory optimization based on the Modular Differential Evolution (MODDE) framework. Instead of designing a single, problem-specific algorithm, we employ a hyperheuristic framework that autonomously manages and configures the components of MODDE to identify optimal trajectories regarding execution time and motion smoothness. We demonstrate that employing this hyperheuristic control results in more robust and adaptive solutions compared to manually tuned metaheuristics, representing a significant advancement in the practical application of evolutionary computation.**

***Index Terms*—Evolutionary computation, metaheuristics, modular design, robotic arm, trajectory optimization, Hyperheuristics.**

I. Introduction

Optimization algorithms are facing increasingly complex challenges in real-world applications, requiring robust and adaptive solutions [16], [38]. To tackle these issues, hyperheuristics have emerged as a highly effective and flexible approach [2]. These enable the use of adaptive mechanisms to achieve generality across diverse problem domains. One direction involves using online learning for heuristic selection and adaptive acceptance criteria [27]. Other studies integrate hyperheuristic frameworks with standard metaheuristics, such as Differential Evolution (DE), using history-based success mechanisms [45]. Parameter adaptivity within metaheuristics remains a crucial research area aimed at enhancing the robustness and efficiency of the optimization process [42].

This challenge of robust adaptation is evident in robotic trajectory optimization, a key prerequisite for successful deployment in complex environments [14]. Classical deterministic planning algorithms can ensure collision-free paths [39], but they often fail to simultaneously satisfy multiple practical requirements, such as trajectory smoothness or multi-criteria optimality [32], without time-consuming manual parameter tuning [6]. Evolutionary algorithms (EAs) represent a promising alternative due to their global search capabilities [38]. However, EA performance is very sensitive to their configuration [4], essentially creating a new, complex optimization problem. These challenging robotics problems are also used to assess transfer learning of surrogate models [28].

This work was supported by the project GACR No. 24-12474S "Benchmarking derivative-free global optimization methods".

Building on the principles of hyperheuristics and modular frameworks [2], this paper presents a fully modular computational framework that treats the design of the optimizer itself as a high-level selection problem. Our approach is based on a Modular Differential Evolution (MODDE) [40]. We frame the challenge of algorithm configuration as a hyperheuristic task to select the optimal combination of DE components and parameters and geometry-specific parameters (that influence the size of the search space) for a given robotic task.

To automate this process, our framework integrates the Irace tool [31]. Irace functions as the high-level selector that automatically configures and tunes the underlying MODDE [37]. To efficiently address the high computational demands, the framework is designed for massive parallelization, ensuring scalability from multi-core desktops to cloud clusters. Finally, we present a comparative study demonstrating the efficiency and robustness of our automatically tuned, modular approach on practical robotic planning tasks.

II. Related Work

A. Classical Algorithms for Trajectory Optimization

Classical approaches to trajectory optimization for robotic manipulators focus on key objectives such as minimizing execution time and jerk, ensuring motion smoothness, and avoiding obstacles [44]. A common goal is to generate time-optimal smooth trajectories, which can be achieved through a variety of methods. These include the use of B-spline curves to guarantee continuity and smoothness, or sinusoidal jerk models to shape the acceleration profile [12]. Other strategies apply input shaping techniques to suppress vibrations rather than explicitly constraining jerk [44].

For complex environments with dynamic obstacles, hybrid strategies are frequently employed. One representative approach combines the Rapidly-exploring Random Tree (RRT) algorithm for global path planning with the Artificial Potential Fields (APF) method for real-time local replanning [43]. Sampling-based motion planning methods, often implemented through libraries such as the Open Motion Planning Library (OMPL), also play a crucial role [21]. The resulting trajectories can be further refined using specialized trajectory optimization algorithms, such as STOMP, within motion planning pipelines [21].

Although originally developed for mobile robotics, methods based on Generalized Voronoi Diagrams (GVD) are also relevant to robotic arms trajectory planning. These techniques aim to maximize clearance from obstacles, thereby increasing path safety and efficiency [41].

B. Neural Network based Optimization

Neural networks are also used in this setting, employing strategies that range from direct trajectory generation to hybrid optimization frameworks [11]. Deep Reinforcement Learning has emerged as a particularly powerful approach for multi-objective trajectory optimization, aiming to minimize factors such as energy consumption, smoothness, and accuracy. Some methods even construct hybrid dynamic models, so called digital twins, where a Deep Neural Network compensates for unmodeled dynamics, and an RL agent subsequently exploits this model to find energy and time efficient paths [29].

Another common strategy involves hybrid systems in which neural networks serve a specific role within a broader optimization pipeline. For instance, Long Short-Term Memory networks can model complex relationships such as energy consumption, and the resulting model is then optimized using genetic algorithms like NSGA-II [34]. Similarly, networks trained via backpropagation can enhance the efficiency of sampling-based planners such as Rapidly-exploring Random Tree Star (RRT*) by guiding the exploration process [8].

Beyond hybrid systems, neural networks are increasingly applied in direct trajectory generation. Modern deep learning frameworks can compute inverse kinematics over entire paths, enabling precise and adaptable trajectory generation without requiring large existing datasets [25].

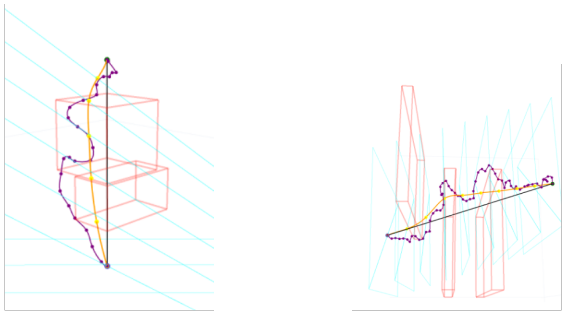
C. Evolution Algorithms Optimization

Metaheuristic and evolutionary algorithms are widely employed for optimizing robotic arm trajectories with the aim of minimizing various objective functions, such as time, joint travel distance, energy consumption, or trajectory tracking error, even in environments with obstacles [5], [35]. Commonly applied methods include Particle Swarm Optimization (PSO) [5], Artificial Bee Colony (ABC) algorithms [35], and Genetic Algorithms (GA), including their multi-objective variants such as NSGA-II, which have proven particularly effective for multi-objective optimization and collision avoidance [14].

Several studies focus on benchmarking the performance of different metaheuristics. For instance, in [17], seven algorithms were compared, with variants of DE and Covariance matrix adaptation evolution strategy (CMA-ES) [10] achieving the best results, a finding confirmed using statistical tests such as the Friedman test, commonly employed for comparing stochastic algorithms [1], [15].

III. Hybrid Modular Optimization Framework for Planning Trajectories

The trajectory optimization problem for robotic manipulators can often be formulated as a black-box optimization task, as finding an analytical solution is practically infeasible due to high dimensionality of the configuration space, nonlinear kinematics and dynamics of the robot, and the presence of obstacles. To efficiently explore this complex search space and obtain an initial, kinematically feasible trajectory, we employ the RRT algorithm. This probabilistic method rapidly generates collision-free paths but does not guarantee optimality. Therefore, the trajectory produced by RRT serves as an initial solution, which is subsequently refined using MODDE, which systematically improves the key waypoints of the trajectory with the objective of minimizing a problem-specific fitness function, such as total path length or motion smoothness (see Figure 1).



(a) Scene 1. (b) Scene 2.

Fig. 1: Case study environments. Red obstacles and the black line (the direct line of sight between the start and the goal, on which blue planar slices that cut through the free space are projected). The purple polyline shows the RRT pre–generated trajectory, and the orange curve highlights the final shortest trajectory found by the evolutionary planner.

A. Methods

1) RRT.: This algorithm is an efficient probabilistic method for trajectory planning, particularly well suited for high-dimensional spaces. Its core principle lies in incrementally building a tree that rapidly expands into unexplored regions of the configuration space. The tree is initialized at the start configuration and iteratively extended [23].

In each iteration, a random point q_{rand} is sampled from the space, and the nearest node q_{near} of the existing tree is identified. From this node, the tree is extended by a fixed step size ε in the direction of the random point, thereby generating a new node q_{new}, provided that the resulting edge does not intersect any obstacles [20], [23]. The expansion process can be described by the following equation:

$$q_{\text{new}} = q_{\text{near}} + \frac{q_{\text{rand}} - q_{\text{near}}}{\|q_{\text{rand}} - q_{\text{near}}\|}\,\varepsilon. \tag{1}$$

2) DE.: DE is an iterative method based on the enhancement of a population of candidate solutions. In each generation, the algorithm creates new candidate vectors by combining information from existing individuals and subsequently compares their performance with the original solutions [3], [33], [37]. If a newly created individual is better, it replaces its predecessor in the population. The key operations of this process are **mutation**, **crossover**, and **selection**.

Mutation. For each target vector $\mathbf{x}_i^{(t)}$, a mutant vector $\mathbf{v}_i^{(t+1)}$ is created by adding one randomly selected vector $\mathbf{x}_{r1}^{(t)}$ to the scaled difference of two other randomly selected vectors $\mathbf{x}_{r2}^{(t)}$ and $\mathbf{x}_{r3}^{(t)}$. This process is described by the following equation:

$$\mathbf{v}_i^{(t+1)} = \mathbf{x}_{r1}^{(t)} + F \cdot (\mathbf{x}_{r2}^{(t)} - \mathbf{x}_{r3}^{(t)}), \tag{2}$$

where $r1, r2, r3$ are mutually distinct random indices from the population, and F is the scaling factor that controls the mutation strength.

Crossover. Subsequently, a trial vector $\mathbf{u}_i^{(t+1)}$ is created by combining elements from the mutant vector $\mathbf{v}_i^{(t+1)}$ and the original target vector $\mathbf{x}_i^{(t)}$, which is controled by the crossover rate parameter Cr. This process ensures the exchange of information between successful solutions.

Selection. In the final step, the trial vector $\mathbf{u}_i^{(t+1)}$ is compared to the target vector $\mathbf{x}_i^{(t)}$. If the trial vector has a better objective (fitness) function value, it replaces the original vector in the population for the next generation. Otherwise, the original vector is retained:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{u}_i^{(t+1)} & \text{if } f(\mathbf{u}_i^{(t+1)}) < f(\mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t)} & \text{otherwise,} \end{cases} \tag{3}$$

where $f(\cdot)$ is the objective function. This simple yet effective mechanism allows the algorithm to efficiently explore the solution space.

3) MODDE.: It represents an extension of classical DE through the principle of modularization. While standard DE employs fixed mutation and crossover operators, the modular approach treats the individual components of the algorithm—such as mutation strategies, crossover types, selection mechanisms, and constraint-handling methods—as interchangeable modules [13], [40].

The main advantage of this architecture is its flexibility and suitability for automated algorithm configuration [40]. Instead of manually tuning parameters and selecting strategies for a specific problem, specialized configuration tools, such as Irace, can be used to systematically explore the space of possible module combinations and their parameter settings. The objective is to automatically assemble the variant of the evolutionary algorithm that achieves the best performance for the given optimization tasks.

4) Irace.: Irace is one of the modern tools for automatic algorithm parameter configuration. The procedure follows a stochastic, iterative process based on so-called races. In each race, a set of candidate parameter configurations is evaluated across multiple problem instances, while configurations that exhibit statistically inferior performance are progressively eliminated. After each iteration, new, potentially better configurations are generated from the best (elite) configurations and used in the subsequent race [24], [30].

Through this mechanism, Irace efficiently explores the configuration space and converges towards a robust parameter setting that maximizes algorithmic performance for the given class of problems [30], [31]. The parameters used in the Irace tuning process are divided into three main groups: general optimization parameters, DE-specific settings, and geometry-related parameters [3], [37].

Main parameters. The mutation factor `F` and crossover rate `CR` both vary between 0.1 and 1.0. The base mutation can be set to `rand`, `best`, or `target`. The population size ranges from 5 to 1000 individuals.

DE parameters. The computational budget was set between 5000–50000 evaluations depending on experimets. The offspring size λ denotes the number of trial (offspring) candidates generated and evaluated per generation in the MODDE framework, and the next population is selected from the union of parents and offspring. We vary λ between 5 and 50.

DE/MODDE parameters. We use a modular DE variant where the main choices control how strongly the search explores or exploits [13], [40]. For mutation, the base vector can be selected either randomly from the population (more exploration), as the currently best individual (more exploitation), or as the current target individual (a more local, target-guided search). The mutation can also use different reference pools: either fully random parents, the current best, or a pool of top-ranked individuals ("p-best"), where the pool size is defined by value (0.1–1.0) [13]. For crossover, we switch between a coordinate-wise mixing (each dimension can be inherited independently) and a block-wise mixing (contiguous blocks are copied), which changes how disruptive the recombination is. When a new candidate leaves the search bounds, we either keep it as is, clip it back to the bounds, or resample the violating coordinates uniformly [3]. We optionally maintain an archive of previously replaced solutions and reuse them during mutation to help preserve diversity and reduce premature convergence. Finally, we allow occasional generation of mirrored ("opposite") candidates with probability 0.1–1.0 to boost exploration, and we vary how many mutation components are combined (1–2, optionally weighted) as well as an oversampling factor (0.1–1.0) that controls how many trials are produced relative to the population size.

Geometry-specific parameters. The geometric parameters control how the trajectory optimization problem is constructed and, consequently, the dimensionality of the search space. We first consider a fixed straight line connecting the start and goal positions. This line is divided by the split factor (geometric division factor) in the range 5–25, which determines how many segments (and thus how many intermediate locations) are created along the line. For each division point, we construct (project) a tangent plane that represents a local sampling region in free space. The tangent size (2–4) defines the size of these planes and therefore the spatial extent in which candidate trajectory points are generated and optimized. Collision validation is performed by sampling 1000 spline points along the interpolated trajectory, and infeasible trajectories are penalized using a fixed penalty value of 200. By changing the split factor and tangent size, the same scene can be solved at different granularity levels, resulting in variable-dimensional optimization problems [18], [36].

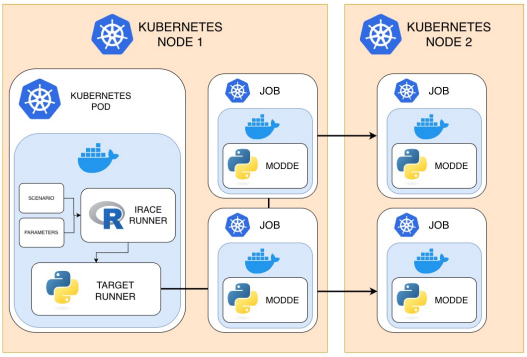


Fig. 2: Deployed system scheme.

B. System Design

The architecture of the proposed system is structured around three tightly integrated layers, jointly forming a robust and scalable framework for large-scale optimization. At the core of this framework lies a modular evolutionary algorithm, designed with an emphasis on flexibility, extensibility, and component reusability. This modularity enables the dynamic composition and systematic evaluation of diverse evolutionary operators and strategies without requiring modifications to the algorithm's core structure. In this paper, a *job* (or *pod*) simply denotes one container instance executing a single configuration evaluation in parallel.

On top of the optimization layer resides the automation layer, implemented using Irace. This component orchestrates the hyperparameter tuning process, efficiently exploring the high-dimensional configuration space of MODDE. The entire system is deployed within a distributed computing environment built on Kubernetes [22]. Within this architecture, Irace operates as the central controller, generating candidate configurations and dynamically allocating computational resources by launching multiple instances of the evolutionary algorithm as independent, containerized pods. The overall system scheme, which enables massive parallelization of the experiments, is shown in Figure 2.

C. Stage I: Initial Trajectory Generation with RRT

To identify an initial feasible trajectory within each scenario, we implemented the RRT algorithm, with a budget of 10,000 iterations. For each scenario, the coordinates of the start and goal positions were first loaded, together with the definitions of rectangular obstacles and the boundaries of the sampling space. In each iteration, a random point was sampled, the nearest node of the existing tree was identified, and a new node was created in the direction of the sampled point. A collision check was carried out by sampling along the line segment between the existing and the new node. If the segment did not intersect any obstacles, the new node was permanently added to the tree. Once the newly generated node approached the goal and a direct, collision-free connection could be established, a valid trajectory was considered found.

Due to the stochastic nature of the method, each scenario was executed 100 times. For each run, the length of the resulting trajectory was recorded, and the shortest one was selected as the best solution. This trajectory was then stored in the scene's JSON definition, with its vertices serving as the initial population for the MODDE algorithm.

From the RRT polyline we extract the points corresponding to the tangent-plane divisions and form a seeded individual. The initial population is then completed by sampling additional individuals as random perturbations around seed populatio within the tangent-plane bounds (controlled by **tangent_size**); this preserves feasibility while maintaining diversity. For the non-seeded variant, all individuals are sampled uniformly within the same bounds.

D. Stage II: Trajectory Optimization with MODDE

To make the optimization tractable, we optimize a geometric path in the Cartesian workspace (a point moving in 3D), rather than a specific robot joint-space trajectory. Each scene is defined by a start point, a goal point, and a set of forbidden regions (axis-aligned cuboids). The straight line between start and goal is used as a backbone and is uniformly split into segments. At each split position we construct a tangent plane (defined with respect to the local direction of the backbone). On every such plane we generate one control point; its placement is determined by the decision vector $\mathbf{x}$, restricted to a bounded region around the backbone whose extent is controlled by `tangent_size`. The full trajectory is then obtained by spline interpolation (best-spline) through the start point, the generated control points, and the goal point.

The candidate solution (one individual) is therefore a vector of parameters that defines the set of control points:

$$\mathbf{x} = \left[\mathbf{p}_1^\top, \mathbf{p}_2^\top, \ldots, \mathbf{p}_{n_{cp}}^\top\right]^\top \in \mathbb{R}^{3n_{cp}}, \tag{4}$$

where $\mathbf{p}_i(\mathbf{x}) \in \mathbb{R}^3$ is the i-th control point on the i-th tangent plane (the start and goal are kept fixed and are not part of $\mathbf{x}$). The parameter `divided_by` controls how many split positions (and thus tangent planes) are created, which directly determines n_{cp} and the dimensionality of $\mathbf{x}$ (a variable-dimension setting [18], [36]).

The trajectory is evaluated by a weighted spline-length term and fixed collision penalties applied both to control points and to sampled spline points. The objective function is defined as

$$F(\mathbf{x}) \;=\; w\cdot L(\mathbf{x}) + \sum_{i=1}^{n_{\text{cp}}}\Big[\mathbf{q}\big(\mathbf{p}_i(\mathbf{x})\big)\cdot P\Big] + \sum_{j=1}^{n_{\text{spl}}}\Big[\mathbf{q}\big(\mathbf{s}_j(\mathbf{x})\big)\cdot P\Big], \tag{5}$$

where $L(\mathbf{x})$ is the length of the interpolated spline, w is the path-length weight, $\mathbf{p}_i(\mathbf{x})$ are the n_{cp} control points, $\mathbf{s}_j(\mathbf{x})$ are the n_{spl} sampled spline points, $\mathbf{q}(\cdot)$ is an indicator that returns 1 if a point lies in a forbidden zone and 0 otherwise, and P is a fixed penalty value.

E. Automated Configuration with Irace

For the automatic configuration and tuning of MODDE, the Irace framework was employed. This tool was integrated into the computational system to systematically explore the extensive space of possible configurations, eliminating the need

for manual parameter tuning and enabling the identification of optimal settings for specific robotic trajectory optimization tasks.

The tuning process was executed in a parallel computing environment using the containerization platform Kubernetes, where the Irace master process dynamically generated and distributed individual computational jobs for evaluating candidate configurations.

IV. Experimental Evaluation

The experiment design focused on evaluating two key aspects: the effectiveness of the automatically tuned evolutionary planner and the benefit of RRT-generated solutions for initialization. To this end, five trajectory planning scenarios (Scenario 1 to 5) with varying levels of complexity were defined, ranging from relatively open environments to scenes with narrow passages and multiple obstacles.

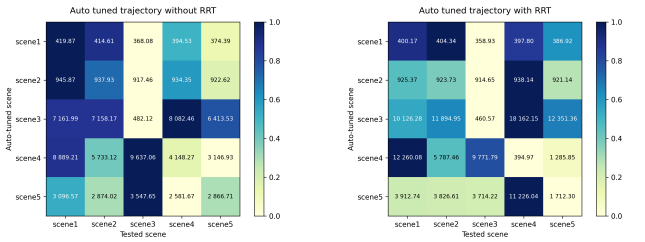
For individual (per–scene) configuration, Irace was run separately on each Scenario 1–5 with identical tuning hyperparameters: 20 initial candidate configurations, a per–candidate budget of 5,000 fitness evaluations, and iterative racing that retained the top five (elite) configurations per scene for final validation on that same scene. The best per–scene configurations were then used in cross–scene tests.

For each configuration we perform 20 independent runs per scene across 5 scenes (6,000 runs total), and a per–candidate budget of 50,000 evaluations, enabling analysis of both median performance and variability

The iterative racing phase produced five elite configurations, which were re–evaluated on a common validation set drawn from all scenes. To ensure statistical significance, each configuration was evaluated on all five scenes with 100 independent repetitions per scene. Including all training datasets, Irace tried in total 3000 experiments per configuration, reaching total of almost 3 millions test on individual scenarios. All large-scale runs were executed on a local MicroK8s cluster spanning two identical Ubuntu 22.04.5 LTS servers (AMD Ryzen 7 2700X, 16 logical cores, 32 GB RAM), with Irace dispatching candidate evaluations as parallel Kubernetes Jobs. Both the code for running the experiments and the data generated in this study are publicly available[1].

A. Automatic Parameter Tuning

Across the five scenes, the optimal configurations differ notably in several key parameters. Scenes use `rand`, `best`, and `target` strategies depending on the landscape complexity (e.g., Scene 1: rand, Scene 2: best, Scene 4: target). Strong variability in population sizes is present (123–935). Larger populations appear beneficial in Scenes 2 and 4, while Scene 3 performs well with a small population. The values vary widely, e.g., F ranges roughly from 0.19 to 0.40, indicating that each scene benefits from a different exploration/exploitation balance. Some scenes rely heavily on oppositional strategies (Scenes 1 and 5), while others disable them completely. Scenes alternate between `rand`/`best` references and `bin`/`exp` crossovers. These differences indicate that each scene exhibits a unique optimization landscape requiring individually tailored parameter combinations.



(a) Auto-tuned individual configuration trajectory without RRT. (b) Auto-tuned individual trajectory with RRT.

Fig. 3: Cross-scene generalization matrix. Each cell reports PRD (%) of the median cost relative to the best-known solution for the corresponding test scene; darker colors indicate lower PRD (better performance).

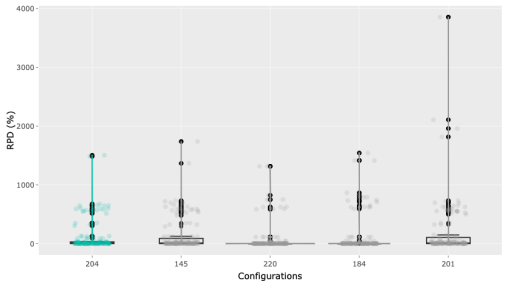


Fig. 4: Cost distribution of trajectories (expressed as PRD from the best known solution) for the five best elite configurations identified by Irace. Configuration 204 shows the lowest median and smallest variance.

Subsequent cross-validation on a shared set of test instances in the figure confirmed that configurations specialized for individual scenarios failed to generalize to other environments. In Figure 3 we can see that even when combining different configurations, no suitable universally performing setup emerges. This further confirms the need for a globally robust configuration. Moreover, the matrix clearly shows that it is not possible to identify a single configuration that would consistently achieve well-rated trajectories across all scenarios, as the performance varies significantly depending on the complexity and structure of each scene.

For the global set of candidates, the configuration labeled `ID204` proved to be the best, achieving the lowest median cost and the smallest performance variance across all test scenarios (Figure 4). This configuration was therefore selected as the default for all subsequent experiments.

The globally best configuration differs from the individual ones in several aspects. It uses consistently `best` as mutation_base and mutation_reference, suggesting that a more exploitative approach generalizes better across all scenes.The global configuration uses a mid-range population (164), unlike

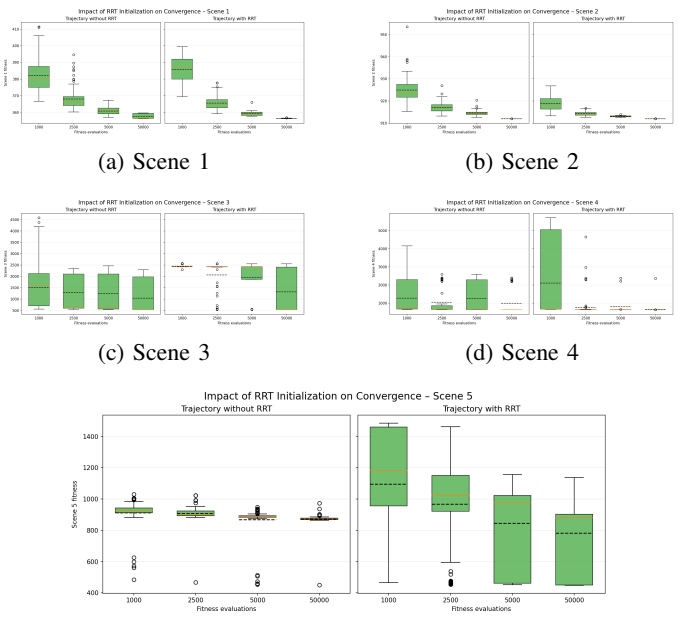
[1]https://zenodo.org/records/17467031

the extremes observed in individual scenes. Both $F = 0.3328$ and $CR = 0.9311$ are balanced values, falling between the extremes used in the individual scenes. Oppositional generation is enabled but with lower probability compared to scenes that strongly relied on it. The configuration keeps oppositional initialization disabled, suggesting this mechanism does not generalize well. Parameters such as `lambda` and `mutation_n_comps` remain consistent and moderate, indicating they contribute to robustness rather than scene-specific optimization. Individually optimized scenes rely on diverse and sometimes extreme parameter choices tuned to their particular landscapes. Globally effective settings must trade maximal per-scene performance for stability and robustness across heterogeneous scenarios.

B. Analysis of the Effect of RRT Initialization

In the next experimental phase, we examined the effect of a hybrid approach where the evolutionary planner is initialized with a trajectory pre-generated using the RRT algorithm. For each scenario, 100 independent RRT runs were executed, and the shortest collision-free trajectory found was inserted into the initial population of the evolutionary algorithm.

Configuration ID204 was evaluated both with and without this RRT initialization, using four different computational budgets: 1,000, 2,500, 5,000, and 50,000 fitness evaluations.



(a) Scene 1 (b) Scene 2

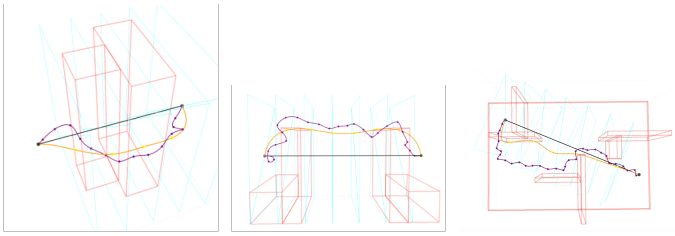
(c) Scene 3 (d) Scene 4

(e) Scene 5

Fig. 5: Comparison of trajectory costs across all scenarios. With RRT initialization (right) and without it (left) at different computational budgets

The results (see Figure 5e) clearly demonstrate that RRT initialization provides a significant advantage.

RRT seeding yields lower median costs across all scenes and budgets. The variance reduction is most pronounced in complex scenes and at low budgets (e.g., Scene 4), while in simpler scenes the spread can be comparable; importantly, the seeded variant also produces fewer extreme outliers at higher budgets (see Fig. 5e).



(a) Scene 3. (b) Scene 4. (c) Scene 5.

Fig. 6: Best trajectories for scenes 3, 4, and 5. All scenes shown side by side.

This benefit was most pronounced at low budgets (1,000 and 2,500 evaluations), which simulate time-critical applications. While the standard approach had to spend a large portion of the budget exploring the space, the RRT-initialized planner quickly converged to high-quality solutions. Even at the highest budget of 50,000 evaluations, the RRT variant maintained its lead and showed fewer outliers and extremely poor solutions. Cross-scenario analysis revealed that the benefit of RRT increases with environmental complexity, particularly in scenes with narrow passages (Scenarios 3, 4, and 5).

C. Comparison with Baseline Evolutionary Algorithms

To verify the competitiveness of the proposed approach, a new, challenging test scenario was created, combining characteristics of all previous environments.

Figure 6 illustrates the best trajectories in the most constrained environments (Scenes 3–5), where the optimizer smooths the initial polyline while maintaining clearance in narrow passages. Figure 8 visualizes the best trajectory on the new composite scenario from top and side views.

On this scenario, our MODDE algorithm (with and without RRT) was compared to three widely used evolutionary algorithms: **CMA-ES**, **GA**, and **PSO**.

The baseline algorithms were configured with standard parameter settings and identical resource conditions of the same 50,000 fitness evaluations.

All algorithms were executed with the same budget of 50,000 fitness evaluations and repeated for 100 independent runs with different random seeds. We report the median and IQR of the final cost. We assessed statistical differences using the Friedman test across algorithms, followed by Holm-corrected pairwise Wilcoxon tests, following established methodological guidelines [19]. Unless stated otherwise, results are obtained from multiple independent runs.

The CMA-ES implementation pycma library [9] was run with a population size $\lambda = 50$ and an initial step size $\sigma_0 = 0.1$. All CMA-ES internal adaptation parameters were left at their recommended defaults. The GA was implemented via the DEAP framework [7] with a real-valued chromosome encoding. Crossover was performed with a blend operator BLX-α (with $\alpha = 0.5$) applied at rate $p_c = 0.9$, and mutation

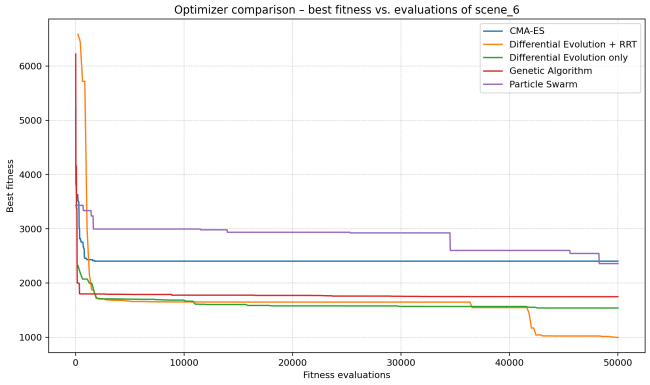


Fig. 7: Convergence curves for the compared algorithms on the new test scenario. The orange curve (MODDE + RRT) achieves the lowest costs and demonstrates the best performance.

was Gaussian with standard deviation 0.05 applied at rate $p_m = 0.2$. Selection in the GA used tournaments of size 3. The PSO was implemented as a global-best (gbest) variant using the pyswarms library [26]. We set the PSO hyperparameters to ensure a stable convergent swarm: cognitive and social acceleration coefficients $c_1 = c_2 = 1.496$ and inertia weight $w = 0.7298$ (this choice corresponds to the constriction factor formulation of PSO, which guarantees convergence properties). The swarm size was 50 particles, and each particle's velocity was clamped to half the range of the search space in each dimension to prevent excessive roaming beyond feasible bounds.

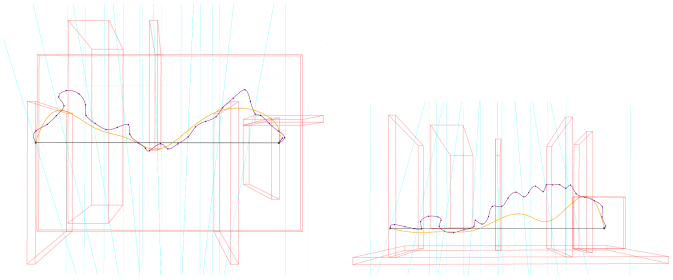
The convergence results shown in Figure 7 reveal a clear performance hierarchy.

- **MODDE with RRT** (orange curve) achieved the best result and found the lowest-cost trajectory.
- **MODDE without RRT** (green curve) converged to a local optimum with slightly higher costs, confirming the benefit of RRT for escaping suboptimal regions.
- **CMA-ES** (blue curve) and **GA** (red curve) showed rapid initial improvement but stagnated at higher cost levels, indicating premature convergence.
- **PSO** (purple curve) exhibited the slowest convergence and did not reach the solution quality of the other methods, likely due to the swarm getting stuck in suboptimal regions of the complex search space.

This comparative analysis confirms that the proposed autotuned MODDE planner, particularly when combined with RRT initialization, outperforms standard evolutionary techniques in solving complex robotic trajectory planning problems.

V. Conclusion

In this article, we consider trajectory optimization in the 3D workspace with obstacle (forbidden-zone) constraints, which corresponds to end-effector path planning. Extending the evaluation to full joint-space kinematics and robot-specific constraints is part of future work.



(a) Top view. (b) Side view.

Fig. 8: Best trajectory found for the new test scenario using the MODDE + RRT method.

Our proposed solution integrates a two-stage optimization process that employs the RRT algorithm to generate an initial path, followed by MODDE to further refine it.

We distinguish between (i) a *two-stage planning pipeline* (RRT for feasible initialization, then MODDE for refinement), and (ii) a *three-layer system architecture* (optimization layer: MODDE, automation layer: Irace, execution layer: Kubernetes-based parallelism).

By incorporating the Irace tool, our framework automates the complex task of hyperparameter tuning, thereby significantly lowering the entry barrier for non-expert users.

The framework is designed for parallel execution in containerized environments, enabling practical scaling when many configuration evaluations are required.

This combination of automated configuration and distributed computation provides a robust and scalable system for solving complex trajectory planning problems.

Future work will focus on extensive experimental validation of the proposed framework. We plan to conduct a thorough comparative analysis against other methods for hyperheuristic optimization, different modular algorithm design frameworks, and state-of-the-art trajectory optimization methods on standardized benchmark problems. In addition, we aim to explore the adaptability of the framework by integrating further metaheuristic techniques into its modular design and extending its applicability to multi-robot coordination and dynamic environments.

References

[1] Bartz-Beielstein, T., et al.: Benchmarking in optimization: Best practice and open issues. arXiv preprint arXiv:2007.03488 (2020). https://doi.org/10.48550/arXiv.2007.03488

[2] Camacho-Villalón, C.L., Stützle, T., Dorigo, M.: Designing new metaheuristics: Manual versus automatic approaches. Intelligent Computing **2**, 0048 (2023). https://doi.org/10.34133/icomputing.0048

[3] Das, S., Suganthan, P.N.: Differential evolution: A survey of the state-of-the-art. IEEE transactions on evolutionary computation **15**(1), 4–31 (2010). https://doi.org/10.1016/j.engappai.2020.103479

[4] Eiben, A., Smit, S.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation **1**(1), 19–31 (2011). https://doi.org/10.1016/j.swevo.2011.02.001

[5] Özge Ekrem, Aksoy, B.: Trajectory planning for a 6-axis robotic arm with particle swarm optimization algorithm. Engineering Applications of Artificial Intelligence **122**, 106099 (2023). https://doi.org/10.1016/j.engappai.2023.106099

[6] Espeseth, A., Juříček, M., Ludwig, H.M., Tušar, T.: Hybrid optimization of horizontal alignments in european terrains: A comparative study. In: García-Sánchez, P., Hart, E., Thomson, S.L. (eds.) Applications of Evolutionary Computation. pp. 119–136. Springer Nature Switzerland, Cham (2025)

[7] Fortin, F.A., et al.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (jul 2012)

[8] Gao, Q., Yuan, Q., Sun, Y., Xu, L.: Path planning algorithm of robot arm based on improved rrt* and bp neural network algorithm. Journal of King Saud University - Computer and Information Sciences **35**(8), 101650 (2023). https://doi.org/10.1016/j.jksuci.2023.101650

[9] Hansen, N., Akimoto, Y., Baudis, P.: Cma-es/pycma on github. zenodo, doi: 10.5281/zenodo. 2559634.(feb. 2019) (2019)

[10] Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary computation **9**(2), 159–195 (2001). https://doi.org/10.1162/106365601750190398

[11] Horgan, A., Mason, K.: Evolving neural networks for robotic arm control. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 591–607. Springer (2023). https://doi.org/10.1007/978-3-031-30229-9_38

[12] Huang, J., Hu, P., Wu, K., Zeng, M.: Optimal time-jerk trajectory planning for industrial robots. Mechanism and Machine Theory **121**, 530–544 (2018). https://doi.org/10.1016/j.mechmachtheory.2017.11.006

[13] Islam, S.M., Das, S., Ghosh, S., Roy, S., Suganthan, P.N.: An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **42**(2), 482–500 (2012). https://doi.org/10.1109/TSMCB.2011.2167966

[14] Juříček, M., Parák, R., Kůdela, J.: Evolutionary computation techniques for path planning problems in industrial robotics: A state-of-the-art review. Computation **11**(12) (2023). https://doi.org/10.3390/computation11120245

[15] Kononova, A.V., et al.: Benchmarking that matters: Rethinking benchmarking for practical impact. arXiv preprint arXiv:2511.12264 (2025)

[16] Kudela, J.: A critical problem in benchmarking and analysis of evolutionary computation methods. Nature Machine Intelligence **4**(12), 1238–1245 (2022)

[17] Kudela, J., Juříček, M., Parák, R.: A collection of robotics problems for benchmarking evolutionary computation methods. In: International conference on the applications of evolutionary computation (Part of EvoStar). pp. 364–379. Springer (2023). https://doi.org/10.1007/978-3-031-30229-9_24

[18] Kudela, J., Juříček, M., Parák, R., Tzanetos, A., Matoušek, R.: Benchmarking derivative-free global optimization methods on variable dimension robotics problems. In: 2024 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2024). https://doi.org/10.1109/CEC60901.2024.10611780

[19] LaTorre, A., et al.: A prescription of methodological guidelines for comparing bio-inspired optimization algorithms. Swarm and Evolutionary Computation **67**, 100973 (2021)

[20] LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Research Report 9811 (1998)

[21] Liu, S., Liu, P.: Benchmarking and optimization of robot motion planning with motion planning pipeline. The International Journal of Advanced Manufacturing Technology **118**(3), 949–961 (Jan 2022). https://doi.org/10.1007/s00170-021-07985-5

[22] Luksa, M.: Kubernetes in Action. Simon and Schuster (2017). https://doi.org/10.5555/3162421

[23] Luo, S., Zhang, M., Zhuang, Y., Ma, C., Li, Q.: A survey of path planning of industrial robots based on rapidly exploring random trees. Frontiers in Neurorobotics **Volume 17 - 2023** (2023). https://doi.org/10.3389/fnbot.2023.1268447

[24] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002

[25] Lúčny, A., Antonj, M., Mazzola, C., Hornáčková, H., Farkaš, I.: Generating and customizing robotic arm trajectories using neural networks (2025), https://arxiv.org/abs/2506.20259

[26] Miranda, L.J.V.: PySwarms, a research-toolkit for Particle Swarm Optimization in Python. Journal of Open Source Software **3** (2018). https://doi.org/10.21105/joss.00433, https://doi.org/10.21105/joss.00433

[27] Mısır, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V.: A new hyper-heuristic as a general problem solver: an implementation in hyflex. Journal of Scheduling **16**(3), 291–311 (Jun 2013). https://doi.org/10.1007/s10951-012-0295-8

[28] Pan, S., Vermetten, D., López-Ibáñez, M., Bäck, T., Wang, H.: Transfer learning of surrogate models via domain affine transformation. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 385–393 (2024). https://doi.org/10.1145/3638529.3654032

[29] Parák, R., Kůdela, J., Matoušek, R., Juříček, M.: Deep-reinforcement-learning-based motion planning for a wide range of robotic structures. Computation **12**(6) (2024). https://doi.org/10.3390/computation12060116

[30] Pérez Cáceres, L., López-Ibáñez, M., Stützle, T.: An analysis of parameters of irace. In: Blum, C., Ochoa, G. (eds.) Evolutionary Computation in Combinatorial Optimisation. pp. 37–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[31] Pérez Cáceres, L., Pagnozzi, F., Franzin, A., Stützle, T.: Automatic configuration of gcc using irace. In: Lutton, E., Legrand, P., Parrend, P., Monmarché, N., Schoenauer, M. (eds.) Artificial Evolution. pp. 202–216. Springer International Publishing, Cham (2018)

[32] Pires, E., Oliveira, P., Machado, J.: Multi-criteria optimization manipulator trajectory planning. InTech (2010). https://doi.org/10.5772/9331

[33] Reyes-Davila, E., Haro, E.H., Casas-Ordaz, A., Oliva, D., Avalos, O.: Differential evolution: A survey on their operators and variants. Archives of Computational Methods in Engineering **32**(1), 83–112 (Jan 2025). https://doi.org/10.1007/s11831-024-10136-0

[34] Rezali, B., Ibari, B., Hebali, M., Berka, M., Bennaoum, M., Bouzgou, K., Ayad, R., Benchikh, L.: Optimal trajectory planning for industrial robots: Minimizing time, jerk, and energy consumption using lstm for energy profile modeling. Journal of Vibration and Control **0**(0), 10775463251333481 (0). https://doi.org/10.1177/10775463251333481

[35] Savsani, P., Jhala, R.L., Savsani, V.J.: Comparative study of different metaheuristics for the trajectory planning of a robotic arm. IEEE Systems Journal **10**(2), 697–708 (2016). https://doi.org/10.1109/JSYST.2014.2342292

[36] Shehadeh, M.A., Kudela, J.: Benchmarking global optimization techniques for unmanned aerial vehicle path planning. Expert Systems with Applications p. 128645 (2025). https://doi.org/10.1016/j.eswa.2025.128645

[37] Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (Dec 1997). https://doi.org/10.1023/A:1008202821328

[38] Stripinis, L., Kudela, J., Paulavičius, R.: Benchmarking derivative-free global optimization algorithms under limited dimensions and large evaluation budgets. IEEE Transactions on Evolutionary Computation **29**(1), 187–204 (2024). https://doi.org/10.1109/TEVC.2024.3379756

[39] Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine **19**(4), 72–82 (December 2012). https://doi.org/10.1109/MRA.2012.2205651, https://ompl.kavrakilab.org

[40] Vermetten, D., Caraffini, F., Kononova, A.V., Bäck, T.: Modular differential evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 864–872. GECCO '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3583131.3590417

[41] Wen, J., Zhang, X., Bi, Q., Liu, H., Yuan, J., Fang, Y.: G^{2}vd planner: Efficient motion planning with grid-based generalized voronoi diagrams (2024), https://arxiv.org/abs/2201.12981

[42] Xie, L., et al.: A novel adaptive parameter strategy differential evolution algorithm and its application in midcourse guidance maneuver decision-making. Complex & Intelligent Systems **10**(1), 847–868 (Feb 2024). https://doi.org/10.1007/s40747-023-01186-1

[43] Yu, W., Qu, D., Xu, F., Zhang, L., Zou, F., Du, Z.: Time-optimal multi-point trajectory generation for robotic manipulators with continuous jerk and constant average acceleration. Control Engineering Practice **154**, 106154 (2025). https://doi.org/10.1016/j.conengprac.2024.106154

[44] Zhang, T., Zhang, M., Zou, Y.: Time-optimal and smooth trajectory planning for robot manipulators. International Journal of Control, Automation and Systems **19**(1), 521–531 (Jan 2021). https://doi.org/10.1007/s12555-019-0703-3

[45] Zhong, R., Yu, J.: Dea^{2}h^2: differential evolution architecture based adaptive hyper-heuristic algorithm for continuous optimization. Cluster Computing **27**(9), 12239–12266 (Dec 2024). https://doi.org/10.1007/s10586-024-04587-0