CIRP Annals - Manufacturing Technology 74 (2025) 19–23

Contents lists available at ScienceDirect

CIRP Annals - Manufacturing Technology

journal homepage: https://www.editorialmanager.com/CIRP/default.aspx

Beyond proxies: A direct time-optimal approach to robot cell layout optimization

Jan Baumgärtner*, Alexander Puchta, Jürgen Fleischer (1)

WBK Institute of Production Science, Kaiserstr. 12, Karlsruhe, 76131, Baden-Württemberg, Germany

ARTICLE INFO

Article history:
Available online 29 May 2025

Keywords:
Design optimization
Handling
Cycle time optimization

ABSTRACT

Cycle time is critical in robotic cells where material handling often presents a bottleneck and in turn depends on the cell layout. Optimizing robot cell layouts is therefore essential for improving cycle time. We show that the proxy objectives used for this optimization are poor approximations of the real cycle time and introduce a optimizer that directly measures the duration of the time-optimal robot trajectory. We show that it can offer competitive performance even faster than proxy-based optimization. We validate this performance on multiple problems and show how to integrate it into the cell design process of a manufacturing cell.

1. Introduction

Throughput and cycle time are important metrics in manufacturing, and it is self-evident that the physical layout of a robotic production cell has a profound impact on both. Cycle time optimization of robot cell layouts is typically based on two main proxy objectives: one aimed at minimizing Euclidean distances between workstations and robots [1–3], and the other focused on minimizing joint-space distances between robot configurations [4–6]. However, these proxies are simplifications that often overlook critical real-world factors, such as obstacles the robot must navigate. As the following examples will demonstrate, their failure to account for such effects can lead to suboptimal cell designs: Fig. 2 for example shows different cell layouts for a simple robot with a revolute (R) and a prismatic (P) joint. In the layout on the left, the Euclidean distance between the workstations and the robot is minimized; in the layout on the right, it is maximized. Yet, the right cell (which optimizes the opposite of the proxy) achieves a significantly shorter cycle time because the revolute joint requires less angular movement, while the prismatic joint movement remains unchanged. This demonstrates the inadequacy of Euclidean distance as a proxy objective as cycle time depends not on distance but on the duration of the movement between stations. While the joint-space distance metric performs better in this scenario, it fails to account for collisions, as illustrated in Fig. 1. On the left, two workstations are positioned to minimize the joint distance between the yellow target positions, but the wall on the red workstation forces the robot to move clockwise, again producing the worst solution. Furthermore, joint distance is often not uniquely defined and has a large influence on cycle time as noted in [7], a factor overlooked in most works such as [1,2], and [3]. While these examples may seem contrived, they show that neither measure reliably correlates with cycle time, which ultimately depends on the robot's entire joint trajectory rather than just its start and endpoints (note that this implies a static trajectory for each execution). There have been a few works trying to calculate the cycle time of a cell but only in the context of time optimally placing a single workpiece to reach several welding points [8,9]. Yet, the true cycle time has not been used as an objective function for optimizing cells with multiple workstations.

This is likely because calculating cycle time requires calculating a robot trajectory which is prohibitively time-consuming (as noted

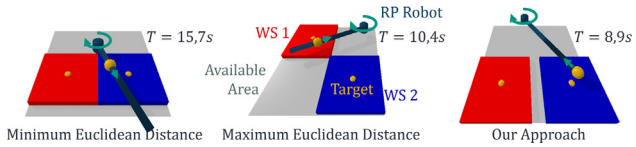

Fig. 1. PR-robot cell with two workstations (blue and red). The left cell minimizes the joint distance between the yellow targets, but a wall collision forces the robot to rotate around the other side, making this the worst possible layout. In contrast, our proposed integrated optimization method rotates the workstation to avoid collisions and thus improves performance.

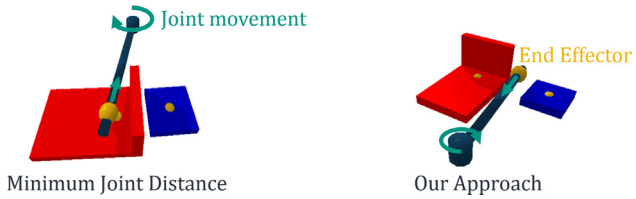

Fig. 2. A PR-robot cell with two workstations (blue and red). The left cell minimizes Euclidean distance, the middle cell maximizes it—yet the middle achieves a shorter cycle time T. The right cell, optimized with the proposed integrated optimizer method, outperforms both proxy objectives. Its apparent area violation here is a perspective artefact.

* Corresponding author.
E-mail address: jan.baumgaertner@kit.edu (J. Baumgärtner).

https://doi.org/10.1016/j.cirp.2025.04.079

in [8]) and the proxies have been judged "good enough". This work aims to challenge the widespread use of these proxy objectives by introducing two new approaches. First, we present a reference method that simply replaces the proxies with true cycle time model, showing that even this straightforward adjustment yields significantly better results. Second, we propose a much faster, integrated optimization method that simultaneously optimizes the layout and trajectory, outperforming proxies in both quality and speed. Additionally, leveraging the speed of our integrated optimization approach, we show that layout optimization can be seamlessly integrated into the cell design process. This enables rapid evaluation of layout and cycle time changes in response to other design decisions such as robot choice, making the entire cell design process more dynamic and effective.

2. Related work

The first serious works on robotic cell layout optimization emerged during the 1990s with foundational contributions such as [1]. Since then, two main variants have emerged: optimizing cycle time for handling tasks or improving manufacturing performance along a given toolpath. In this work, we will focus on the former, while drawing valuable insights from the latter. While a few studies [8,9] have modelled cycle time directly they typically compute only a collision-free path rather than a time-optimal trajectory and are limited to optimizing the placement of a single workpiece. By contrast, most studies addressing the layout of multiple workstations rely on proxy objectives, such as minimizing Euclidean distance [1–3] or joint distance [4–6]. Bachmann et al. [2] and Tubaileh et al. [5] are representative of these proxy-driven approaches. While their methods do not model the influence of obstacles along the robot's entire path, they at least address collisions at workstation interaction points by approximating obstacles as spherical constraints. This simplification is useful for quickly computing collisions and will be adopted in our approach to model collisions along the entire robot trajectory. However instead of manually defining these collision spheres, like the previous works, we will develop a method to automatically generate them from the true object meshes. Motivated by the tenfold speedup achieved in [5] compared to [2] we will also employ gradient-based optimization for our integrated solver. Here we were inspired by [10] which combined the optimization of robot poses over a given toolpath with layout optimization into a single problem. We will build on this method using the spherical collision hulls to plan collision-free time-optimal trajectories between workstations. Our goal is to use the resulting integrated method to optimize cycle time directly without the use of any proxies while producing results in a reasonable time frame.

3. Problem formulation

Before introducing the new optimization method, we will first mathematically define our optimization problem. Here we will deviate from the cell design optimization literature in three main ways: We will use the duration of the time optimal robot trajectory directly as our objective function, we will compute the spherical constraints based on the true collision meshes and we will not confine ourselves to cartesian design parameters. Instead of specifying fixed dimensions of motion (eg, 2D, 3D, or 6D), we adopt an approach that reflects real-world design constraints. Workstations often have unique requirements, such as maintaining a minimum distance from a wall or proximity to other equipment. To accommodate this, we use a "kinematic harness" based on [10] to define permissible motion for each workstation and optimize over the joint space of this virtual mechanism. For example, planar motion with rotation can be modelled by two prismatic and one revolute joint, while two revolute joints can be used to model a fixed distance from the robot.

Mathematically we thus optimize the constant "design joints" q_d introduce by [10] that describe the state of these mechanisms in addition to the robot joints $q_m(t)$. The time-optimal problem for each transition between workstations can thus be formulated as follows:

$$\begin{aligned} &\underset{u, q_d, T_i}{min} && \int_0^{T_i} 1 + L dt \\ &subject\ to: && \dot{q}_m(t) = u(t) \\ & && p(q_m(0)) = s_{i-1f}(q_d) \\ & && p(q_m(T)) = s_{is}(q_d) \\ & && \| p(q_m(t)) - c_k(q_d) \| \geq r_k \, \forall \, t \in [0, T_i] \, \forall \, k \in K \\ & && u_{\min} \leq u(t) \leq u_{\max} \ \forall \, t \in [0, T_i] \end{aligned} \tag{1}$$

where $p\left(q_m(t)\right)$ is the pose of the robot at time t, s_{i-1_f} is the final pose of the robot at the previous workstation, s_{i_s} is the start pose of the robot at the current workstation, $c_k(q_d)$ are the centers of the spherical constraints approximating the collision shapes and r_k are their radii. The robot is controlled through joint changes $u(t)$ and our optimization objective is the time T_i it takes for the i-th transition between workstations. If desired we can also include extra costs L such as manipulability: $L = det\left(J\left(q_m(t)\right)\right)$ with J being the Jacobian of the robot. This optimization is called for every transition i resulting in the total optimization problem:

$$\begin{aligned} &\underset{u, q_d, T_i}{min} && \sum_i \int_{T_{i-1}}^{T_i} 1 + L dt \\ &subject\ to: && \text{constraints from} (1) \end{aligned} \tag{2}$$

We automatically generate collision spheres (c_k, r_k) from each workstation's mesh by converting it into a point cloud and applying k-means clustering to determine sphere centers and radii. This method lets us use larger spheres for broad shapes and smaller spheres for intricate details. Compared to using a uniform grid, our method achieves higher fidelity with fewer spheres, reducing the optimization's complexity. Each additional sphere adds a constraint, so choosing k balances collision accuracy and solver performance. When necessary, increasing k refines the approximation for more detailed collision shapes (See Fig. 3).

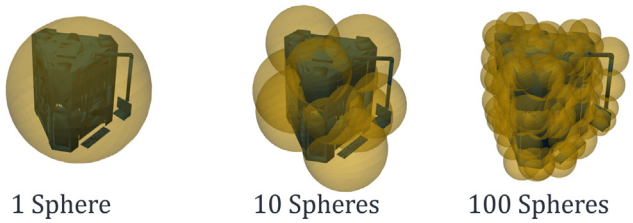

Fig. 3. Spherical collision hull of the Ponticon additive manufacturing system [14] for different numbers of spheres.

Note that since the collision spheres always protrude a bit out of the object it is highly likely that they keep the robot from reaching some pick-up or drop-off location at the workstation.

Following [2], we thus define each interaction with the workstation not by a single final pose but by a small, manually defined approach trajectory like a linear interpolation for inserting a slot or a curved path for a locking hook. Consequently, we assume that the approach trajectory, defined relative to the workstation, is collision-free and thus does not need to be constrained. Note that if there is no special interaction required this trajectory can be a single point, as we will show later. A graphic illustration of the final problem can be seen in Fig. 4.

4. Proposed method

Before introducing the integrated optimization method, we first outline the time-optimal trajectory optimization whose trajectory duration could replace the proxies in an iterative solver. We will then show how this approach naturally extends to the integrated method. To do this, we discretize the time horizon into N segments of length

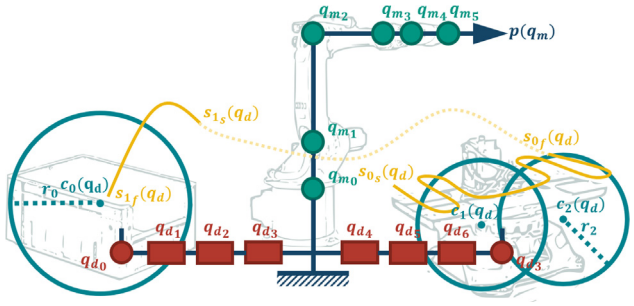

Fig. 4. Schematic of the optimization problem in (1), with design joints q_d representing permissible workstation movement, and robot joints q_m representing the joints of the robots while collision spheres (c_k, r_k) modell collisions and approach trajectories s modelling workstation interactions.

dt, so the total time is $T = N \cdot dt$. Let q_m^n be the robot's joint configuration at time $n \cdot dt$, and let u^n be the corresponding control input. The forward kinematics can then be approximated by:

$$p^n = p(q_m^n) = p(q_m^{n-1}) + dt\, J(q_m^{n-1})\; u^{n-1} \tag{3}$$

Where J is the Jacobian of the robot. The problem now becomes a classic trajectory optimization: minimize our discretized objective function $\sum_n w\ dt + L(q_m^n, q_d)$ subject to the kinematic model (3) and the constraints of (1). The parameter w balances time optimality and the optional secondary objective $L(q_m^n, q_d)$. Decreasing dt effectively "accelerates" the motion within the constraints of the kinematic model. Following [10], we use direct collocation [11], approximating the control input as a degree-l polynomial and learning its coefficients—an approach that increases sparsity and simplifies solving larger design spaces [11].

The solution to this problem still depends on the cell design q_d, because the trajectory must pass through the start and end of the approach trajectories s_i. A straightforward way to include q_d is nested optimization where an iterative solver, in this case particle swarm optimization (PSO) [12], proposes candidate designs q_d, each evaluated by the trajectory optimization. Here, trajectory optimization simply replaces traditional proxy objectives. We will call this method the iterative solver. However, this method is inherently slow, as each candidate requires a full trajectory evaluation.

Therefore, we propose a second method that solves both trajectory and layout optimization simultaneously. Building on [10], we treat the design joints q_d as additional joints in the trajectory but constrain them to remain constant over time (using a 0th-order polynomial in the collocation framework). We introduce a tracking cost d to ensure that the gradient of the objective with respect q_d can be computed (even if we do not want to optimize for a secondary objective L). This replaces the constraints enforcing the trajectory to start at s_{i-1_f} and end at s_{i_s}. This yields the following optimization problem:

$$\begin{aligned} \underset{u,q_d,dt_i}{\text{minimize}} \sum_i \sum_n & w\ dt_i + L\left(q_m^{n,i}, q_d\right) + d\left(q_m^{n,i}, q_d, s_i\right) \\ \text{subject to} \quad & p^{n,i} = p\left(q_m^{n-1,i}\right) + dt_i \cdot J\left(q_m^{n-1,i}\right) u^{n-1,i}, \\ & \| p^{n,i} - c_k(q_d) \| \geq r_k \qquad \forall n, i\ \forall k \in K, \\ & u_{min} \leq u^{n,i} \leq u_{max}\, \forall n, i. dt_i \geq 0\, \forall i \end{aligned} \tag{4}$$

Note that instead of minimizing T_i directly, we minimize dt_i for each transition because dt_i is explicitly part of the system's dynamics while T_i is not. We also constrain dt_i to be positive to avoid negative-time solutions. All transitions are discretized with the same number of time steps N so that they contribute equally to the objective. Note also that L can either be evaluated at each transition time step, or only at the approach trajectories s_i, making this approach a direct extension of [10]. The resulting collocation equations can then be solved using a nonlinear programming solver such as in this case Ipopt [13]. Since this method integrates the trajectory optimization directly into the layout optimization, we will call this the "integrated solver".

5. Validation and results

In the introduction, we highlighted key shortcomings of using Euclidean and joint distances as proxy objectives for cycle time. Euclidean distance ignores the robot's joint movements, while joint distance is not uniquely defined since different joint configurations can lead to the same end-effector pose. Additionally, neither accounts for the influence of obstacles on cycle time. Since we measure cycle time directly without relying on any proxies, the ambiguity of joint distance is no longer a concern. However, we still need to ensure that our approach effectively addresses the other two issues: accounting for joint movements and the impact of obstacles on cycle time. To test whether the integrated solver can cope with these two issues we performed two experiments on the examples from the introduction.

The results can be seen in Fig. 2 and Fig. 1, which show that our method successfully deals with the shortcomings of the proxies improving on the cycle time of the first example while correctly rotating the obstacle out of the way in the second example. However, a small gap remains between the workstations due to spherical collision approximations. This gap could be reduced by increasing the number of spheres but has been kept larger to highlight a potential limitation of our method, as the accuracy-performance trade-off when choosing the parameter k may not always allow tight fits. Additionally, with thin walls, like in the second example, the number of control intervals N needs to be carefully chosen. If N is too low, the robot might pass through walls since collisions are only checked at discrete points in time. Increasing N solves this problem if it occurs.

5.1. Comparison to proxy-based optimization

It is perhaps not surprising that the true cycle time metric can handle edge cases where proxies fail. However, proxies are valued for their ability to produce results of comparable quality in significantly less time [14]. To thus properly compare our approach to proxy-based solutions, we designed another experiment to quantify both the performance gap and the computational cost trade-off between proxy-based and true cycle-time optimization. To make the comparison as fair as possible and give proxies the best possible conditions to perform well, we deliberately chose a simplified problem: optimizing the placement of four small rectangular workstations in 4D space (position and Z-rotation) without complex collisions. We also use an identically configured PSO for both the iterative solver and the two proxies configured according to [15] using 160 particles over 40 generations. This setup should even the playing field as much as possible while ensuring that any observed performance gap is due solely to the choice of objective function rather than problem complexity. To ensure fairness, all methods were subjected to identical constraints, including spherical collision approximations, and a cycle time penalty of 1000 for constraint violations in proxy-based solvers. Approach trajectories were fixed to single points centered above the workstations, and no secondary objectives were used. Results were averaged over 30 runs and plotted as cycle time versus CPU time in Fig. 5. For proxy-based solvers, the resulting cycle time of their candidate solution was calculated using trajectory optimization with N=100 as done for the integrated solver, ensuring that all solutions were measured against the same performance metric. Solutions that didn't converge as measured by the tracking cost d were discarded. The

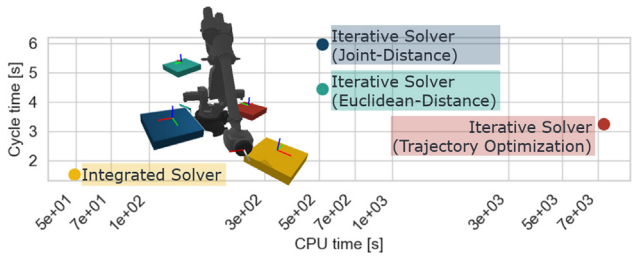

Fig. 5. Comparison of CPU time and resulting cycle time for all optimizers on the four-workstation placement problem. The integrated solver outperforms all other solvers in terms of cycle time and computation time.

results in Fig. 5 show that while the proxies run about 10 times faster than the iterative solver they were 33% worse (a ~1.3-second gap) even in a situation without collisions as seen in Fig. 1. This poor result is especially surprising for the joint distance proxy objective, which is even outperformed by the Euclidean distance objective. This could be related to the multiple inverse kinematics solutions that could discount good layouts because the chosen robot configurations were suboptimal. Irrespective of possible reasons, the experiments indicate that even for simple problems we should not necessarily trust proxies to truly optimize cycle time. Still using the same optimization approach there is a computational cost trade-off. However, comparing proxies and the iterative solver to the integrated solver, we see that the integrated solver not only produces solutions that are 50% better but also achieves them in a significantly shorter computation time. This demonstrates that directly optimizing cycle time does not necessarily come at the expense of speed and can be competitive with, or even faster than, proxy-based approaches. While different proxy-based optimizers exist with varying levels of efficiency, the results show that it is possible to directly optimize cycle time without introducing prohibitive computational costs.

The main computational advantage of the integrated solver is that it optimizes the entire system in a single run avoiding repeated optimizations or inverse kinematics calculations required by the proxies and the iterative solver. This gradient-based approach is also probably better suited to deal with the curse of dimensionality [15] and kinematic singularities often encountered in time-optimal solutions. When motion in certain directions is restricted, gradients needed for optimization become unreliable. The integrated solver mitigates this issue by also optimizing the workpiece position, maintaining useful gradients even when the robot encounters a kinematic singularity—unlike traditional trajectory optimization. That said, poor starting configurations can lead the integrated solver to significantly worse solutions, with cycle times reaching up to 12 s. This issue is particularly pronounced when workstations are initialized too close together or even overlapping, as this creates immediate constraint violations. To address this, we recommend arranging the workstations equidistantly around the robot at an appropriate radius, ensuring no initial overlaps. This approach enhances both the solver's reliability and speed.

5.2. Application case study

Why is speed crucial in layout optimization? While it might seem feasible to design the layout while waiting for components to arrive, a faster optimization process enables a more integrated design approach, as highlighted in the introduction. Robot selection, for instance, depends on both cycle time requirements and reachability, which in turn require an initial layout. Traditionally, this involves manually placing machines and selecting a robot to fit. However, with a fast layout optimizer, these decisions can be automated and streamlined. To demonstrate this, we applied our integrated solver to design a hybrid manufacturing system featuring a DMG Mori CNC machine and a Ponticon metal 3D printer [16]. The robot's task is to transfer material between these machines to form an additive-subtractive process chain. We evaluated two potential industrial robots: the Comau NJ290, which has slower joint speeds (1.57 rad/s for the first axis) but greater reach, and the Kuka KR 180 r2500, which is faster (2.145 rad/s for the first axis) but smaller. While the Comau's larger reach could potentially compensate for its slower joints and make it faster in certain layouts, this depends on the specific arrangement of the workstations. Additionally, we considered whether a Scara robot such as the PVC 52900 could reach into both machines, despite their openings being at vastly different heights. To verify, we defined approach trajectories starting before each machine's loading doors and introduced an additional repeatability cost L [17] to ensure high repeatability at the clamping positions. Our integrated solver quickly assessed the robots, revealing that the Scara robot could not reach both positions (as seen by the large tracking cost d) and that the Kuka only slightly outperformed the Comau in cycle time by 0.2 s. Given the large difference in joint speed, this is much closer as one might assume. Both cells are shown in Fig. 6. The results, summarized in Table 1 highlight how fast layout optimization enables testing of a wide range of robots to identify not just the most suitable but also potentially unexpected candidates like the Comau robot. Note also how the CPU time is similar to Fig. 5 even though we have fewer workstations since more complex collisions need to be considered. Alternatively, one could find the cheapest option that satisfies cycle time requirements.

Table 1
Comparison of cycle times for different robots with an optimized cell layout shows that while the Scara is the fastest, it cannot reach both machines, as indicated by the tracking cost d.

	Kuka KR180	Comau NJ290	Scara
CPU time [s]	1.681	1.584	2.65
Cycle time [s]	1.693	1.893	0.271
Tracking cost $\boldsymbol{d}$ [m]	8e-2	6e-10	1.04

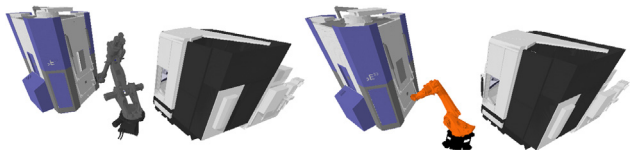

Fig. 6. Left: optimized comau cell; Right: optimized kuka cell. While the layouts appear similar, machine distances differ slightly to accommodate the size variations.

6. Conclusion and future work

The main claim of this paper is that the two common proxies for cell optimization are poor approximations of cycle time, in part because they fail to account for the significant impact of workstation obstacles on cycle time. Using the examples of Fig. 1 and Fig. 2, we show that the use of proxies leads to much worse results compared to our proposed approach. We have also shown that even when constructing an example that gives proxies every advantage, simply replacing the proxies with a true cycle time calculation yields better results. Although this involves a trade-off in speed, our newly proposed integrated trajectory optimization gives even better results while achieving competitive runtimes. In fact, we showed that it is fast enough to open the door to a more integrated automated cell design process by also automating robot selection. While these results establish the viability of direct cycle-time optimization, they do not fully define its limitations. Our study focused on only four cell design examples, which does not reveal how well the approach scales with more complex workstation geometries, cluttered environments, and brownfield applications. Further work is needed to test these conditions. Some assumptions, such as the existence of collision-free approach trajectories, need further validation. Currently, robot self-collisions cannot be directly checked and are modelled with joint constraints, which might exclude valid solutions.

Ongoing work is also looking into incorporating acceleration limits by extending Eq. (4) to a second-order model.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Jan Baumgärtner: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alexander Puchta:** Writing – review & editing, Project administration. **Jürgen Fleischer:** Writing – review & editing, Supervision, Funding acquisition.

Acknowledgment

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German ResearchFoundation) – SFB-1574 – 471687386.

References

[1] Tay M, Ngoi B (1996) Optimising robot workcell layout. *International Journal of Advanced Manufacturing Technology* 12:377–385.

[2] Bachmann T, Nottensteiner K, Roa MA (2021) Automated planning of workcell layouts considering task sequences. *2021 IEEE international conference on robotics and automation (ICRA)*, 12662–12668. https://doi.org/10.1109/ICRA48506.2021.9561831.

[3] Zhang J, Li A-P (2009) Genetic algorithm for robot workcell layout problem. *2009 WRI World Congress on Software Engineering*, IEEE, , 460–464.

[4] Zhang D, Qi L (2009) Virtual engineering: optimal cell layout method for improving productivity for industrial robot. *2008 IEEE Conference on Robotics, Automation and Nechatronics*, IEEE, , 6–11.

[5] Tubaileh AS (2015) Layout of robot cells based on kinematic constraints. *International Journal of Computer Integrated Manufacturing* 28(11):1142–1154.

[6] Sharma A, Jha AK, Halder A (2017) Layout optimization of a robotic cell for foundry application by CAD based point cloud modeling—a case study. *Industrial Robot: An International Journal* 44(6):788–797.

[7] Erdős G, Kovács A, Váncza J (2016) Optimized joint motion planning for redundant industrial robots. *CIRP Annals* 65(1):451–454. https://doi.org/10.1016/j.cirp.2016.04.024.

[8] Spensieri D, Carlson JS, Bohlin R, Kressin J, Shi J (2016) Optimal robot placement for tasks execution. *Procedia CIRP* 44:395–400. https://doi.org/10.1016/j.procir.2016.02.105.

[9] Bu W, Liu Z, Tan J (2009) Industrial robot layout based on operation sequence optimisation. *International Journal of Production Research* 47(15):4125–4145.

[10] Baumgärtner J, Puchta A, Fleischer J (2024) One problem, One solution: unifying robot design and cell layout optimization. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, https://doi.org/10.1109/IROS58592.2024.10801417.

[11] Kelly M (2017) An introduction to trajectory optimization: how to do your own direct collocation. *SIAM Review* 59(4):849–904.

[12] Kennedy J, Eberhart R (1995) Particle swarm optimization. In: *Proceedings of ICNN'95-International Conference on Neural Networks*, 1942–1948.

[13] Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* 106(1):25–57. https://doi.org/10.1007/s10107-004-0559-y.

[14] Desale S, Rasool A, Andhale S, Rane P (2015) Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *International Journal of Computer Engineering in Research Trends* 351(5):2349–7084.

[15] Chen S, Montgomery J, Bolufé-Röhler A (2015) Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence* 42:514–526.

[16] Fleischer J, et al. (2024) Self-learning and autonomously adapting manufacturing equipment for the circular factory. *Automatisierungstechnik* 72(9):861–874. https://doi.org/10.1515/auto-2024-0005.

[17] Baumgärtner J, Gönnheimer P, Fleischer J (2023) Optimal robot workpiece placement for maximized repeatability. *Advances in System-Integrated Intelligence* 546:252–261.