PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131
0736-5845/

Full length article

Towards optimal task positioning in multi-robot cells, using nested meta-heuristic swarm algorithms

S. Mutti [a,*], G. Nicola [a,b], M. Beschi [a,c], N. Pedrocchi [a], L. Molinari Tosatti [a]

[a] Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, National Research Council of Italy, 20133, Milan, Italy [b] Università di Padova, Department of Information Engineering (DEI), via Gradenigo 6/A, Padua, Italy
[c] Università di Brescia, Dipartimento di Ingegneria Meccanica ed Industriale, via Branze 38, Brescia, Italy

A B S T R A C T

While multi-robot cells are being used more often in industry, the problem of work-piece position optimization is still solved using heuristics and the human experience and, in most industrial cases, even a feasible solution takes a considerable amount of trials to be found. Indeed, the optimization of a generic performance index along a path is complex, due to the dimension of the feasible-configuration space. This work faces this challenge by proposing an iterative layered-optimization method that integrates a Whale Optimization and an Ant Colony Optimization algorithm, the method allows the optimization of a user-defined objective function, along a working path, in order to achieve a quasi-optimal, collision free solution in the feasible-configuration space.

1. Introduction

Working cells composed of multiple robots are being used more often for a huge variety of industrial tasks (*e.g.*, welding, laser cutting, painting, additive manufacturing *etc.*) [1–4], for their dexterity and re-configurability (see for example Fig. 1). In such a configuration, the position where the robot holds the work-piece strongly affects the operation feasibility (*e.g.*, respect of the joints limits) as well as the performance in term of path tracking as the direct consequence of the different robot kinematics (*e.g.*, different joints speed, accelerations, friction, *etc.*). The identification of a good position is a time consuming procedure even for skilled operators. The operator, indeed, has to find collision free trajectories for the robots, that is far to be easy for many operation, especially when the tool has to be re-oriented. In such a case, according to the tool dimension, a small Cartesian movement of the tool center point (TCP) often forces the robot to span a large movement at the joint level,[1] and the identification of a configuration for which the operation is feasible is challenging due to the limited range of motion of many axes, and the likely collision between the links during the movement. Many off-the-shelves software tools (*e.g.*, [5,6]) allow the operator to relax the constraints along the paths (see Fig. 2), and thanks to local gradient-based optimization, they slightly modify the orientation of the tool to avoid collisions and to limit the joints motion as much as possible. However, these tools are not able to find solutions when the cell is complex or in case of a multi-robotic arm setup. In literature, considering a single-arm workcell, this problem is generally tackled by the maximization of dexterity along the path [7–11], by using performance evaluation indexes [12], or controlling functional redundancies [13]. However, such methods do not consider problems such as backlashes or transmissions elasticity that cause poor robot accuracy even after static calibration.

Fig. 1. Laser Cutting performed by two cooperative Robot (LT360 from BLM Group, [1]).

To overcome such limitations, many works [14–18] propose methods tailored on specific tasks and setup that may be extended to multi-arm work-cells. Specifically, in [14], the optimization is performed using a gradient method, that, however, may lead to the identification

* Corresponding author.
E-mail address: stefano.mutti@stiima.cnr.it (S. Mutti).

[1] Consider Fig. 1. To contour a tube of 50 mm the wrist center has to follow a circumference of about 1 m as diameter.

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131

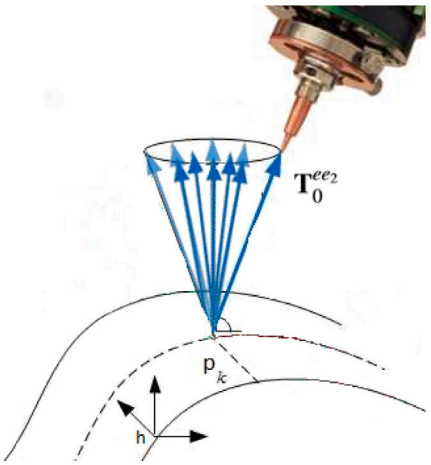

Fig. 2. Cladding application: The axis of the deposition head must lie in a cone centered with the surface normal direction, without quality deterioration.

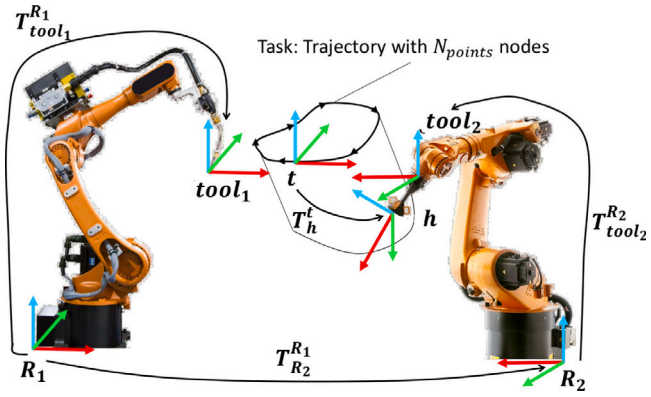

Fig. 3. Robotic cell scheme.

of a local minimum, without the possibility to span all the alternative robot configurations. Pamanes et al. [15] proposes a non-linear programming technique requiring at the beginning a candidate solution that satisfies all the constraints. The work relaxes the velocity constraints, *i.e.*, the robot is supposed to be able to change configuration between two consecutive nodes, but the approach is not full-automatic and it is not suitable for when the problem is computationally complex. Karamani et al. [16] proposes a response surface method but, since the objective function and the output are interpolated, the optimal solution might be very far from the global minimum due to the inaccuracy introduced by interpolation model adopted. Furthermore, they superimpose the robot configuration making bijective the inverse kinematics function. Santos et al. [17] proposes a tunneling method for searching the global minimum, but since the algorithm numerically solves a highly non linear equation many times, it is not suitable for all those application in which the computational time is relevant. In [18], instead, the optimization is performed by two consecutive genetic algorithms, making complex the balancing between exploration and exploitation.

Many works focus on stiffness-oriented methods [19–21], where the robot configuration is optimized in order to raise the stiffness, with benefits in applications that involve vibrations.

As alternative to the path optimization, a few works [22–25] address this problem focusing on the cell layout optimization, but despite the methods are interesting this practice is unlikely applicable in industry, where the robotic cell is used for many different operations.

Generally, optimization techniques are un-satisfactory for multi-robot tasks. First, the domain search space is huge, and may be a non-connected manifold since the constraints are always expressed in the Cartesian space but the inverse kinematics is not a bijective function. Also, even infinite solutions may be possible if the task is lazy-constrained (*e.g.*, when the tool can rotate around its own axis). A tentative to cope with these issue is in [26]. The work focuses on the optimization of the workpiece positioning, taking in account only the kinematic redundancy of the robots involved. The methodology consists of splitting the problem in two sub-problems, and to run iteratively two nested optimizers: first the problem of the object positioning is solved (*e.g.*, definition of the Robot-2 holding position), then, the Robot-1 redundant path is optimized. Finally, an iteration over the two steps is computed. This work, however, do not consider the technological redundancy (see Fig. 2), and the trajectory is fully constrained, *i.e.*, the robot tool orientation must be properly aligned as the Frenet frame in each trajectory frame. On the one hand, this assumption keeps limited the search space. On the other hand, it reduces dramatically the applicability of the methodology. Among the plenty of optimization algorithms, [26] proposes to use a Whale Optimization Algorithm (WOA) and an Ant Colony Optimization algorithm (ACO) for the first and second optimization steps respectively. On the one hand, Mirjalili et al. [27] prove that WOA is among the best-performing meta-heuristic algorithms on a large set of mathematical problems. On the other hand, the ACO is extremely efficient when a combinatory problem has to be solved [28].

Starting from [26], this work brings two major upgrades: it introduces a new method to deal with the dimension of the search space allowing the integration in the model of the task redundancies, and it integrates an efficient collision checking strategy to deal with unfeasible trajectories. As remark, the collision checker ensures that the search in the redundant space avoids the exploration in proximity of unfeasible paths.

The paper is organized as following: in Section 2 the optimization problem is mathematically formulated, in Section 3 the optimization methodology is presented, highlighting the advancement with respect to [26], in Section 4 the analysis of a paradigmatic use case is reported, and some general considerations on the methodology performance are discussed. .

2. Problem formulation

Consider the generic robotic cell composed by a robot performing a specific task (Robot-1) and a robot holding the work-piece in static position (Robot-2) as shown in Fig. 3. The affine transformation matrix $\mathbf{T}_{R_1}^{R2}$ between the 2 robots base frames $R1$ and $R2$ is known. For the ith robot denote $\mathbf{q}^{Ri} \in \mathbb{R}^{dof_i}$ as the vector of joint angles, where dof_i is the joints number of the ith robot, and $\mathbf{T}_{Ri}^{ee^i}$ as the transformation from the base frame Ri to the end-effector frame ee^i according to the forward kinematics, *i.e.*,

$$\mathbf{T}_{Ri}^{ee^i} \equiv FK_{Ri}\left(\mathbf{q}^{Ri}\right) \tag{1}$$

Regarding the inverse kinematics, denote $\mathcal{Q}^{Ri}$ as the set of size N_{sol} of the solutions in the considered pose,

$$\mathcal{Q}^{Ri} \triangleq \left\{\mathbf{q}_1^{Ri}, \ldots, \mathbf{q}_{N_{sol}}^{Ri}\right\} = IK_{Ri}\left(\mathbf{T}_{Ri}^{ee^i}\right). \tag{2}$$

As a remark, the size N_{sol} depends mainly on the kinematics, and on the range of motion of the axes.[2]

Consider 0 and h the absolute and the work object frame respectively, where the Work Object Frame is the frame that is used to describe the path that the robot has to perform. The path is a sequence of N_p different frames p_k, each described by a transformation $\mathbf{T}_h^{p_k}$ with $k = 1, \ldots, N_p$.

[2] If the axes are multi-turn, and/or if there are self-collision in some configurations.

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131
0736-5845/

The basic assumption that the Robot-2 is not moving while the Robot-1 performs the operation corresponds to impose that, at the kth trajectory node, the robots tool frame $\mathbf{T}_0^{ee_1}|_k$ and $\mathbf{T}_0^{ee_2}|_k$

$$\mathbf{T}_0^{ee_1}|_k \equiv \mathbf{T}_0^h\,\mathbf{T}_h^{p_k}, \tag{3a}$$

$$\mathbf{T}_0^{ee_2}|_k \equiv \mathbf{T}_0^h. \tag{3b}$$

The assumption in (3a) can be relaxed for many nodes of the trajectory. Indeed, the tool pose often can differ from the Frenet frame assigned nominally to the node, and technological redundancies can be exploited. Define $Alt\left(p_k\right)$ as the function that for each p_k computes a set of possible alternatives satisfying given constraints (*e.g.*, the tool lies inside a cone centered around the normal to the surface), using the discretization method as in 3.1:

$$A_k \triangleq \left\{ \mathbf{T}_h^{a_{k,1}}, \ldots, \mathbf{T}_h^{a_{k,N_A}} \right\} = Alt\left(p_k\right), \tag{4}$$

where N_A is a constant over k denoting the dimension of the discretization of the searching space,

$$\mathbf{T}_h^{a_{k,1}} \triangleq \mathbf{T}_h^{p_k}$$

Once the set A_k is computed for each kth node of the trajectory, we can denote C_k as the set of dimension N_{C_k} of the feasible joint configurations $\mathbf{q}_1^{R1}|_k$, as

$$\begin{aligned} C_k &\triangleq \left\{ \mathbf{q}_1^{R1}, \ldots, \mathbf{q}_{N_{C_k}}^{R1} \right\}|_k \\ &= \bigcup_{s \le N_{A_k}} Q_k^{R1}\left(\mathbf{T}_{R1}^0\,\mathbf{T}_0^h\mathbf{T}_h^{a_{k,s}}\right) \end{aligned} \tag{5}$$

Worthily, C_k is function of $\mathbf{T}_0^{ee_2}$, *i.e.*, of the holding position as lead by Robot-2. However, in order to span only the configuration space of the Robot-1, it is possible to super-impose that the holding position is generated as starting position from Robot-1. Denote $\mathbf{q}_{start}^{R1}$ as a starting joint configuration for the Robot-1, and p_1 as the first node of the trajectory. Closing the kinematics loop as show in Fig. 3, results in

$$\begin{aligned} Q^{R2} &\triangleq IK_{R2}(\mathbf{T}_0^h) = IK_{R2}(\mathbf{T}_{R2}^0\,\mathbf{T}_0^{ee_1}\,\mathbf{T}_{p_1}^h) \\ &= IK_{R2}(\mathbf{T}_{R2}^{R1}\,FK_{R1}\left(\mathbf{q}_{start}^{R1}\right)\mathbf{T}_{p_1}^h) \end{aligned} \tag{6}$$

Once defined $\mathbf{q}_{start}^{R1}$ is therefore possible to compute the search grid Γ as

$$\Gamma\left(\mathbf{q}_{start}^{R1}\right) \triangleq \left\{ C_1, \ldots, C_{N_p} \right\} \tag{7}$$

The size of the search grid may be very large. As matter of example, consider a trajectory with 100 frames. Assume that meanly for each node, 10 poses are feasible alternative poses, with a mean of 4 inverse-kinematics solutions each. Therefore, the number of possible trajectories are $(4 \times 10)^{100}$ for each $\mathbf{q}_{start}^{R1}$.

Finally, once the search grid is defined, the last step consists of defining an objective function $Obj(*)$, to be maximized, that returns a scalar value called objective value, for any path over the grid. The definition of the objective function fully defines the behavior of the optimization, and it is strongly related to the selected performance goal.

The full methodology has been called Whale and Ant Colony Optimization (WACO) and its pseudo code is shown in table Algorithm 1.

3. Method

This presented methodology is an improvement of [26].

Specifically, the contribution consists of the robust-statistical discretization of the task-redundant space, the integration of the collision detection and an improved ACO algorithm designed to manage large search graphs.

The algorithm is composed of two nested iterative optimization methods. The outer loop implements a Whale Optimization Algorithm (WOA), an iterative meta-heuristic swarm optimization algorithm with a fixed number N_{whales} of particles called "whales". At every iteration i, takes as input the solution of the previous iteration, and it generates the particles x_i^j (with $j \le N_{whales}$), that corresponds to defining the starting position of the trajectory, and the full nominal trajectory of p_k frames as direct consequence.

As described in the previous Section, for each p_k it is then possible to impose a set of constraints as the maximum misalignment between the tool axis and the surface normal direction, as well as the maximum Cartesian speed according to the user needs. The inner loop, implemented with an Ant Colony Optimization algorithm (ACO), iterates over the multitude of these different paths granting the constraints, and returns the iteration-best path $\mathcal{P}_i^*$ found, and a corresponding scalar value $ObjVal_i^*$ to the outer loop.

3.1. Search space discretization

Halton sequences have been selected to discretize the task redundant space [29]. The effectiveness of these sequences is related to the fact that they span homogeneously a search space, preserving the minimum distance between samples in a k-dimensional space [30]. Specifically, consider an integer number n, and its representation in the base b_j as

$$n \equiv \sum_{i<M} a_{j,n}^i b_j^i,$$

with M is an integer larger enough that all the digits of the number are represented, and each coefficient a^i is $0 \le a_{j,n}^i \le b_j$. The Halton sequence is constructed according to a deterministic method that uses coprime numbers as its bases, and following [31], we can compute a single Halton sequence as the s-tuple

$$x_n = \left(\Phi_{b_0}(n), \ldots, \Phi_{b_s}(n)\right)$$

where $\Phi_{b_j}(n)$ is the jth radical inverse function:

$$\Phi_{b_j}(n) = \sum_i a_i(j,n) b_j^{-i-1},$$

To improve the covering of the space, a scramble of the coefficients is necessary, and the method in [32] has been implemented.

3.2. First optimization layer: configuration calculus

Denote N_{WOA} and i as the number of WOA iterations and the iteration index ($i \in 1, \ldots, N_{WOA}$), while denote N_{whales} and j as the number of the particles and the particle index ($j \in 1, \ldots, N_{whales}$). Finally, denote x_i^j as the jth particle whale generated at the ith iteration. Among the different possibility, we selected the Robot-1 joint positions as the WOA particle held value, *i.e.*, $x_i^j \equiv \mathbf{q}_{start}^{R1}|i$, where the value at the iteration 1 is a random sampling of valid joint values. Following the procedure in Section 2 and (7), the search grid results

$$\Gamma_i \longleftarrow \Gamma\left(\mathbf{q}_{start}^{R1}|i\right)$$

where the discretization of the search grid is performed using $Alt\left(p_k\right)$, and depends on the task redundancy. The starting joint position is valid only if at least one path exist, that is

$$\forall k \le N_p \qquad C_k \ne \emptyset$$

and, a feasible path from the starting pose to the final pose of the trajectory is defined as

$$\mathcal{P} = \left\{ \mathbf{q}_{s_1}^{R1}|_1, \ldots, \mathbf{q}_{s_k}^{R1}|_k, \ldots, \mathbf{q}_{s_{Np}}^{R1}|_{N_p} \right\}, \text{ with } \mathbf{q}_{s_k}^{R1}|_k \in C_k,\ \forall k \tag{8}$$

The selection of the highest objective value returning path $\mathcal{P}_i^*$, among the feasible ones, for the ith WOA iteration (outer layer), cannot be computed using any analytical objective function due the excessive number of feasible paths. The Ant Colony Optimization algorithm (ACO) is therefore used as in [26] as the core of inner optimization layer for the computation of $\mathcal{P}_i^*$, and the evolution of the WAO is obtained by the evaluation of the underlying ACO resulted objective value, and the generation of a new vector of robot joints value $\mathbf{q}_{start}^{R1}|i+1$

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131
0736-5845/

Algorithm 1: Whale and Ant Colony Optimization Algorithm (WACO)

Data: Load Task, robotic cell kinematic, and geometric data p_k and N_p
Result: Optimized path $\mathcal{P}^*$

begin
  // Initialize
  $N_{WOA} \leftarrow$max number of iterations before exit
  $N_{whales} \leftarrow$max number of whales computed in parallel
  $ObjVal_i^j \leftarrow 0, \forall i, j$
  $MaxObjVal \leftarrow 0$
  $i \leftarrow 1$

  // WOA Outer Loop until convergence
  repeat
    for $j \leftarrow 1$ **to** N_{whales} **do** // For Loop Computed In Parallel
      $\mathbf{q}_{start}^{R1} \leftarrow x_i^j \leftarrow$ WOA_StateGenerator$(ObjVal_{i-1}^j)$

      // Build the search graph
      $\Gamma_i \leftarrow \Gamma_i\left(\mathbf{q}_{start}^{R1}\right)$

      // Call the inner optimization algorithm. Γ_i is purged by the nodes that are in collisions
      $\left(\mathcal{P}_i^{j*}, ObjVal_i^j, \Gamma_i\right) \leftarrow$ InnerOptimizator(Γ_i)
    end
    $MaxObjVal \leftarrow \max_j \left\{ObjVal_i^j\right\}$
  until $i \leq N_{WOA}$
end

Algorithm 2: Inner Optimization layer: Iteration over Ant Colony Optimization algoritm (ACO)

Input: Γ_i
Output: $\mathcal{P}_i^*, ObjVal_i^*, \Gamma_i$

begin
  Assign weights to all edges in graph Γ_i
  $N_{ants} \leftarrow$ number of ants
  $N_{ACO} \leftarrow$ max number of iterations of the ACO algorithm before exit
  $\epsilon \leftarrow$ convergence threshold to exit
  $ok \leftarrow False$
  if *heuristic offline computation* **then**
    Update the weight of graph Γ_i using a proper heuristic, as in (9)
  end

  repeat // until $\mathcal{P}_i^*$ is not collision free
    if *heuristic online computation* **then**
      Update the weight of graph Γ_i using a proper heuristic, as in (9)
    end
    $(\mathcal{P}_i^*, ObjVal_i^*) \leftarrow$ ACO_PathSelector$(\Gamma_i,$ ObjectiveFunction$, N_{ants}, \epsilon, N_{ACO})$ // it optimizes the obj func on the grid
      // it updates the pheromone
      // it runs all the Ants in Parallel
    $(ok, \Gamma_i) \leftarrow$ CollisionChecker$(\mathcal{P}_i^*, \Gamma_i)$ // Check collision, see algorithm 3
    if $\Gamma_i == \emptyset$ **then**
      $ObjVal_i \leftarrow 0$
      return
    end
  until ok
end

3.3. Inner optimization layer: the path selection

ACO's aim is that of finding a path that maximizes the objective function on the inputted grid. While the graphs in [26] were almost small, the graphs here considered may be huge in size. The Ant Colony Optimization algorithm (ACO) deployed in [26] is therefore insufficient, and a Elitist Max–Mix ACO is here proposed. Following [33], the pheromone deposited on a node has a minimum and a maximum threshold, and the only solution taken in account at every iteration is the one with the highest objective value. To tune the algorithm's parameters, the average of the number of solutions per point has been used as the number of nodes instead that the whole number of nodes as in [33], since the graph may be not fully-connected.

ACO takes as input a graph, while Γ_i is just a grid of points. In order to transform it in a graph, all the weights of the connections between the nodes have to be set. To transform it in a directed graph, each joint configuration in C_k must be connected to the configurations in set C_{k+1} (see Fig. 4). The connection is formalized through weights, their presence help the algorithm in its search strategy towards the optimum. If a connection with a consecutive node is not possible for any reason (*e.g.*, overcome the maximum speed) the weight is put to null, and it disconnect the nodes. A commonly used formula to assign the weights is the inverse of the norm of the joint distance vector between two joint

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131
0736-5845/

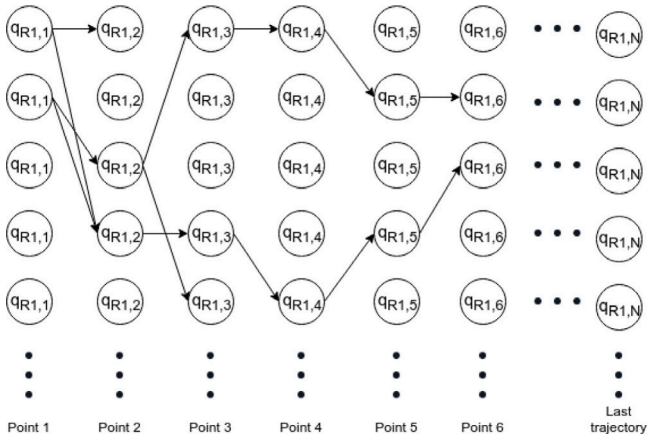

Fig. 4. Simplified graph example. The columns hold different joint configurations for the trajectory frame.

configurations belonging to two consecutive points:

$$weight = \frac{1}{\left\| (q_k^{R_1} - q_{k-1}^{R_1}) \right\|} \tag{9}$$

but many more can be used as heuristic criteria. The computation of the weights can be done online or offline. Computing the values offline before feeding the graph to the ACO algorithm avoids that the heuristic is computed more times for the same node, while computing it online as the ACO performs, allows calculating only the needed weights.

In the case of the offline heuristic, the number of values to compute is equal to:

$$N_{offline} = (N_A\, N_C)^2\, N_p \tag{10}$$

where N_A is the dimension of the set A_k, N_C the dimension of set C_k and N_p the number of points in the trajectory. In the case of the online heuristic, the number of values to compute is equal to:

$$N_{online} = N_{ants}\, N_{ACO}\, N_p\, N_A\, N_C \tag{11}$$

where N_{ACO} and N_{ants} are the maximum number of iterations of ACO and the number of Ants used respectively. Denoting r as

$$r = \frac{N_{offline}}{N_{online}} = \frac{N_A\, N_C}{N_{ants}\, ACO_{iter}}, \tag{12}$$

the online computation is better when $r \geq 1$, otherwise the offline computation is most performing.

The algorithm evolution is defined by the objective function that assigns a scalar value to path $\mathcal{P}$

$$ObjVal = \text{ObjectiveFunction}(\mathcal{P}). \tag{13}$$

The definition of the function must be selected accordingly to the goal performance (*e.g.*, minimization of the length of the path, minimization of the number of velocity inversion *etc.*).

Due to computational effort, it is not possible to check the collision inside the ACO algorithm. Therefore, the collision is performed only on the best so-far path $\mathcal{P}^*$ found by the ACO algorithm. The solution, however, might be not feasible due to collision. In this case, the inner optimization layer iterates over ACO in order to find a feasible path. Remarkably, the collision detection algorithm purges from the graph all the configuration that are in collision, discarding solutions that are not feasible rather than including them in the optimization (see Section 3.4). In algorithm 2 is shown the inner optimization layer pseudo-code.

Algorithm 3: Collision detection

Data: Robot-1 (R1), Robot2 (R2), Environment (E), WorkObject (WB), Tool (T)
Input: Γ_i, $\mathcal{P}_i$
Output: ok, Γ_i

```
1  begin
2      if inCollision(R2(p), E) then
3          ok ← False                    // set graph to unfeasible
4
5          Γ_i ← ∅
6          return
7      end
8
9      for each p in P_i do
10         ok ← not inCollision(T(p), WB) if not ok then
11             for each Tool orientation in p    // check collisions
               between tool and workpiece for every tool
               orientation in p
12             do
13                 if inCollision(T, WB) then
14                     Γ_i ← PurgeNode(Γ_i)
15                 end
16             end
17             return
18         end
19     end
20
21     for each p in P_i do
22         ok ← not inCollision(R1(p), E)
23         if not ok then
24             for each IK solutions of R1 in p    // check collisions
               between Robot1 and Robot2 for R1 IK solution
               in p
25             do
26                 if inCollision(R1, E) then
27                     Γ_i ← PurgeNode(Γ_i)
28                 end
29             end
30             return
31         end
32     end
33
34     for each p in P_i do
35         ok ← not inCollision(R1(p), R2)
36         if not ok then
37             for each IK solutions of R1 in p    // check collisions
               between Robot1 and Robot2 for R1 IK solution
               in p
38             do
39                 if inCollision(R1, R2) then
40                     Γ_i ← PurgeNode(Γ_i)
41                 end
42             end
43             return
44         end
45     end
46     ok ← True
47 end
```

3.4. Collision detection

Collision detection is computationally demanding, and the number of possible combinations of robot cell joint position to be checked at every WOA iteration is equal to $N_{whales} \times \dim(\Gamma)$. Also, it increases exponentially with the number of discrete redundant configurations and points in the trajectory. In this module the collision detection problem was solved by using the state of art of collision detection

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131

libraries and by exploiting the peculiarities the problem studied. The collision detection is performed with the library GPU-Voxels [34] that implements voxel based algorithms and that takes advantage of parallel computation on GPU to increase computational speed.

The collision detection is divided in 4 parts in order to take advantage of the peculiarities of the involved problem, collisions in the multi-robot cell studied can have various sources (robots–environment, robot–robot, tool–work-piece).

Voxel based methods can have different outputs based on the discretization chosen, indeed a coarse discretization has an higher probability of having false positives (i.e. the two object result in collision even if it is not true) but it is computationally less expensive. Moreover the discretization required depends directly from the desirable minimum distance between the objects to be checked. So objects that are required to be closer should have fine discretization while object that are in general farther and not required to be close should have a coarse resolution, for example the tool and the workpiece can have a distance of some millimeters while the robots are not required to be so close. In conclusion, the different sources of collision have different level of discretization based on the aforementioned motivations.

It can be easily noticed that when a collision is found it is possible to delete other nodes close to the one that is in collision. Specifically, a collision between the Robot2 and the environment, since both are static, sets the entire graph as unfeasible, meanwhile a collision between the tool and the workpiece automatically deletes from the graph a number of nodes equal to the inverse kinematic solution for that specific position and orientation of the tool-head.

The collision detection routine check all the possible sources of collision: Robot2–Environment, Tool–Workpiece, Robot1–Environment, Robot1–Robot2. The routine is composed by four different steps with two different level of discretization:

1. Robot2–Environment (low discretization)
2. Tool–Workpiece (high discretization)
3. Robot1–Environment (low discretization)
4. Robot1–Robot2 (low discretization)

If a collision is found, except the case 1, all the graph nodes belonging to the same trajectory frame are checked before computing a new ACO iteration. If no collision free node is found the graph is set as unfeasible. The order of the collision detection steps has been chosen to perform firstly the steps that can reduce the graph dimension more quickly. The pseudo-code is shown in Algorithm 3

4. Performance analysis

4.1. Experimental setup

The software designed to perform the tests uses the ROS framework, in order to import robot parameters from the ROS industrial [35] project repository, and to take advantage of the ROS middle-ware libraries, it also makes use of a modified version of the ikfast openrave [36] library for computing the inversion kinematic of a robots in a parallel way on the GPU, its architecture is shown in Fig. 5.

The setup is composed by two ABB IRB4600 20.5 (see Fig. 1) configured for pipes laser cutting.

Different paths with varying number of points of have been investigated, from simple circular holes and pegs on the pipes side, to a complete trimming of the tips. The trimming of the tips is considered a difficult operation due to the circular path of the robot around the pipe, and the collision possibility during the path execution.

For the presented cases, the number of points in the trajectory is 50, while the task redundancy consists of a free rotation around the z tool axis.

In the experiments, the objective function to maximize is the inverse of the quadratic sum of the joints speed of the Robot1 along the trajectory, with a penalization term that takes in account the proximity

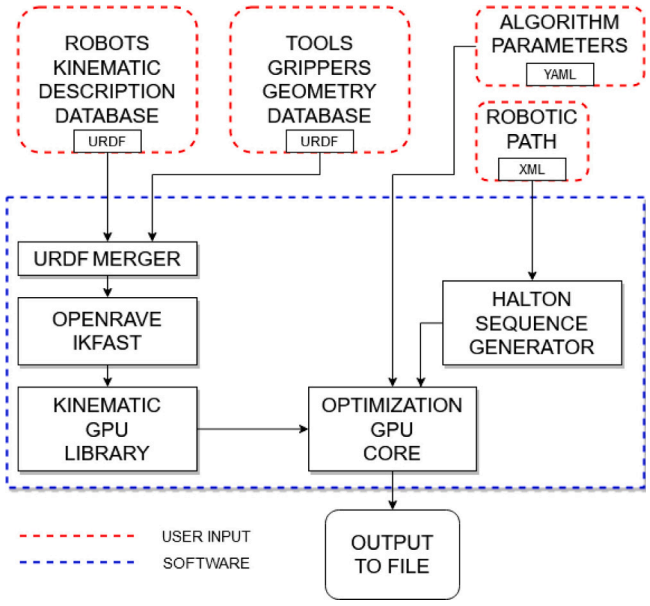

Fig. 5. Software architecture.

to joint limits, due to the degraded performance of the robot near the limits.

The trial has been run with different parameters combination:

a. For the outer optimization layer:
   - N_{WOA}, the iterations number of WOA
   - N_{whales}, the number of the whales particle

b. For the inner optimization layer:
   - N_{ACO} iterations number
   - N_{ants}, Ant number

Different redundancies have also been tested, from the rotation around the cutting head Z axis, to a more relaxed conic redundancy in which the head Z axis can freely move inside a given cone with a small aperture. The trials have been made on a desktop computer equipped with a NVIDIA GeForce gtx 1060 GPU, on which the algorithm has been designed, taking advantage of the CUDA libraries for parallel computation. Due to the size of the memory needed for running the algorithm, which raises exponentially with the discretization parameters and trajectory frames, the tests performed have a limited number of parameters combination. However, as shown by the collected results, the problem can be adequately solved with a limited amount of resources. The code implementation used, its configuration and the resulted data of the tests can be accessed at [37].

4.2. Sampled results considerations

All the trials gave a feasible , collision free path as output. In Figs. 6,7,8, are shown some significant results extracted from algorithms run on the same task with different WOA parameters. The dotted lines show the best so-far value of the objective function among all the WOA particles at every iteration, while the dashed line shows the average of the dotted lines. The continuous lines show the average value of the objective function hold by all the whales particles at every iteration. All the performed trials, regardless of the parameters value, have shown a predictable trend, in which it is noticeable the presence of 2 distinct phases. During the first phase, the whales particles “explore” the joint space of the robots, the hold solutions are diverse(as shown by the difference between the average value, continuous lines, and the

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131

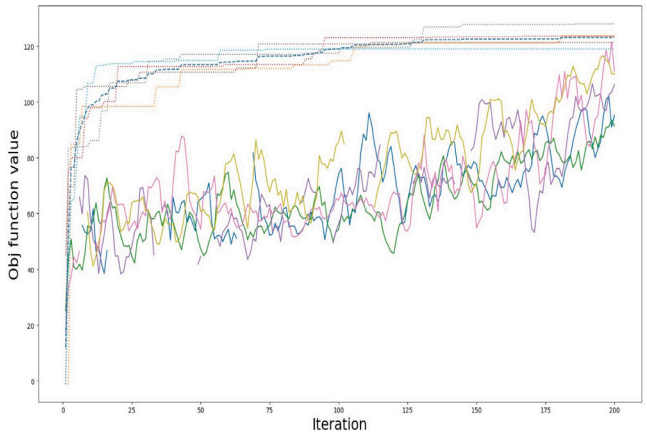

Fig. 6. Objective value trends for peg hole task with 10 whales particles for 200 iterations.

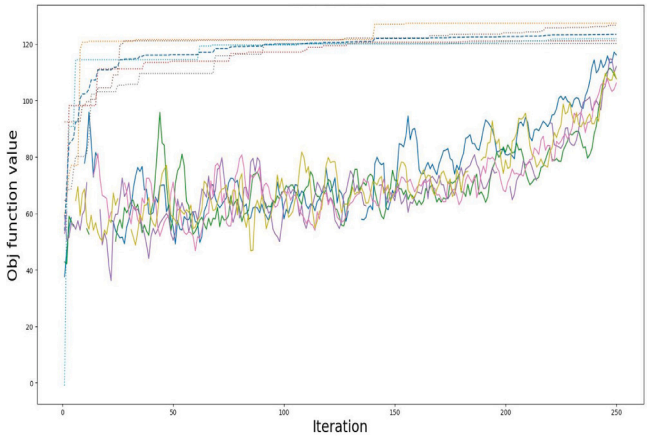

Fig. 7. Objective value trends for peg hole task with 20 whales particles for 250 iterations.

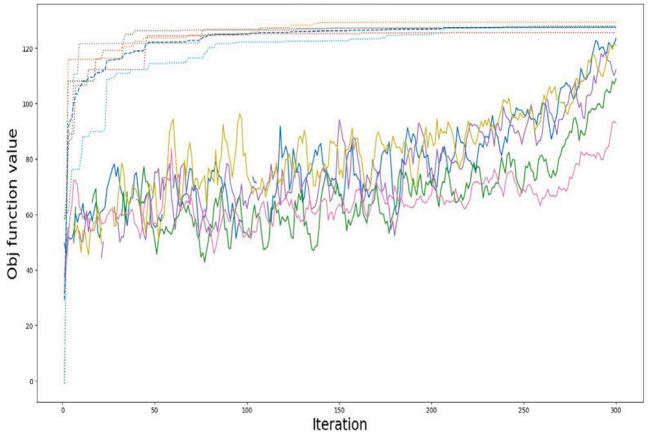

Fig. 8. Objective value trends for peg hole task with 25 whales particles for 300 iterations.

best so-far values, dotted lines) and constantly change due to the WAO algorithm nature and its parameters value. In the second phase the solution are refined, the WOA algorithm particles move in proximity of their solutions and the exploration is concluded, the search is now focusing on optimizing the best solutions found so-far, settling on the nearest optimum points. The experiments are repeated 5 times for each parameters combination to have a statistical sample.

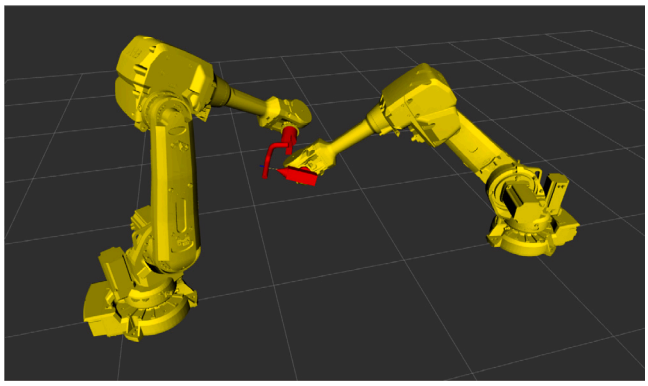

Fig. 9. Cell configuration 1 - peg hole task.

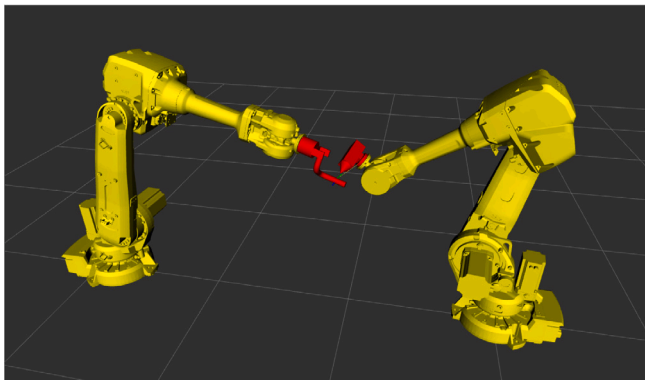

Fig. 10. Cell configuration 2 - tip trimming task.

In Figs. 9 and 10 are shown 2 of outputted result configuration, where the Robot-1 is shown fixed at the first point of the trajectory to perform, the external environment is hidden for sake of visualization simplicity. Figs. 11 and 12 show the resulted path solution for a hole peg and trimming operation, where it is noticeable that the path frames exploit the task redundancy by changing the directions along the path.

Furthermore, the algorithm is shown to hugely exploit the available redundancy, changing the TCP orientation during the movement in order to decrease joint speeds and accelerations during the work head re-orientations. This increases in cases like the tip trimming, where taking advantage of the task redundancy is crucial to have smooth solutions. Fig. 13 shows the average deviation from the nominal Z TCP angle during 2 different types of tasks, collected using 100 experiments per type of task.

4.3. Parameters influence

The parameters of the WOA and ACO algorithms have been tuned following practice from the state-of-the-art, namely, the tuning for the ACO parameters is detailed in [33], for the parameters relative to the pheromone threshold value and decay rate, while WOA parameters adopt the nominal value proposed by the authors in [27]. The decision to consider such values for the parameters, has been taken due to the extensive analysis performed by the respective authors, and by our extensive analysis with different tuning methods. Specifically, the WOA a_{lim} parameter, which sets the proportion between exploration and exploitation, is set to 1, while ACO agents are set to 1024, with an iteration number of 50 and the weights exponential coefficient set to 2. Internal parameters relative to ACO pheromone are related to the graph dimensions, and hence are set dynamically in the algorithm. Besides these parameters, WACO algorithm presents further parameters

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131

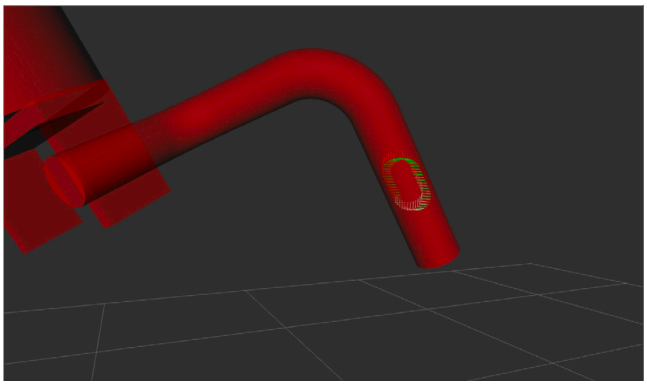

Fig. 11. Path peg hole.

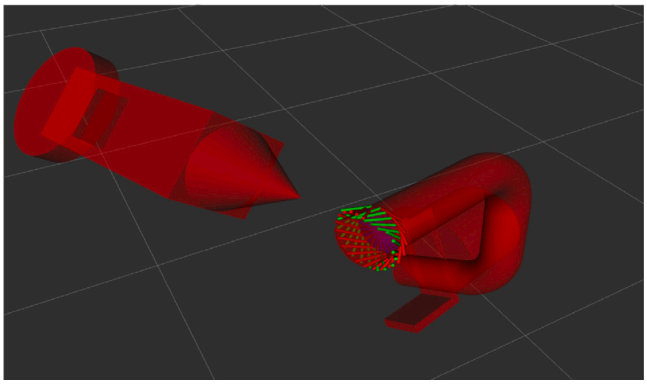

Fig. 12. Path trimming.

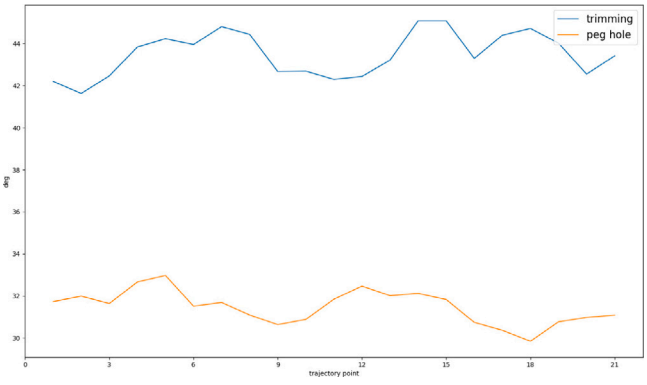

Fig. 13. Average redundant angle deviation in degrees from the Z Frenet frame axis, for different tasks, using different WOA parameters, normalized on a 21 point path.

to be tuned as N_{whales} (the number of whales particles) and the N_{WOA} (number of iterations). The algorithm running time is shown to depend linearly on N_{whales} and N_{WOA}, as shown in Fig. 14, where the average time and its standard deviation are plotted for different parameters. The running time deviation value is due to the collision checking phase, which is the non-deterministic part of the algorithm in terms of execution time. The bottom part of Fig. 14 shows the trend of the objective value per whales and whales iteration numbers, showing that it has an asymptotic behavior. This behavior highlights the fact that the optimization problem is well solved with a limited amount of resources and that the solution space is explored adequately. Fig. 15 shows the relations between the objective value and the computation time, suggesting the parameters zone with a good trade off in term of computation time and result optimality.

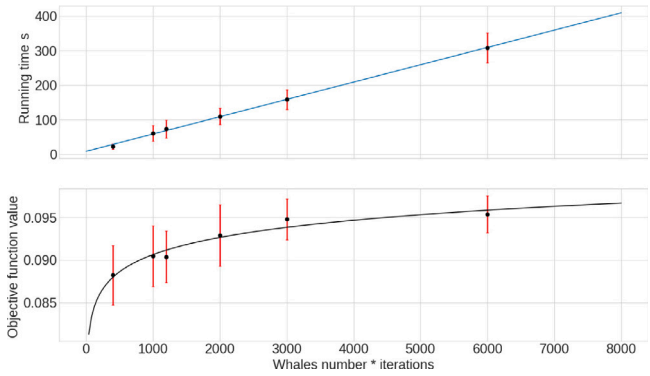

Fig. 14. Algorithm running time(top) and objective function value(bottom), using number of WOA particles times number of iteration on the abscissa. Fixed number of ACO cycles(50), tip trimming task. Upper figure shows data with a linear fitting, while the bottom one a natural logarithmic fitting.

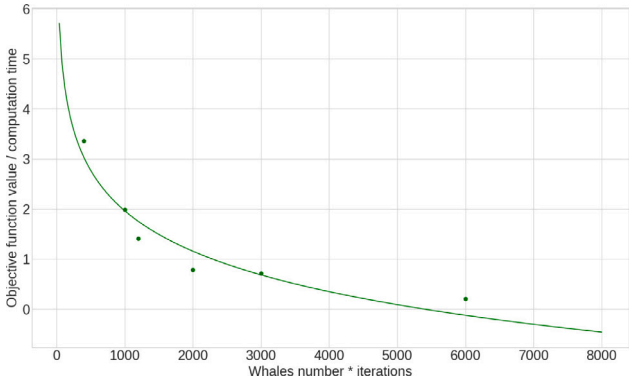

Fig. 15. Objective value over algorithm running time, test case same as Fig. 14, data fitted with a natural logarithm.

5. Conclusions and future developments

In this paper a method named WACO (Whale and Ant Colony Optimization) to optimize task placement for process planning of redundant multi-robot cells is presented, the optimization is performed by 2 nested meta-heuristic algorithms (WOA and ACO) in order to enhance the overall accuracy of the task. The method assures the kinematic feasibility of the task trajectory, the task and kinematic redundancies exploitation, and the optimization of collision free trajectories. Internal parameters of the nested algorithms have been tuned using well known state of the art procedures, which have proved adequate for the problem in question. The WACO has demonstrated convergence to a common sub-optimal solution, in regard to the number of agents involved in the computation, showing furthermore a common optimization threshold, which highlights the satisfactory exploration of the solution space. Furthermore, the rising trend of the solutions found per iteration, suggests that the optimization methods is efficient in the problem optimization. Results show that the exploitation of the task redundancy is a key element to the optimization. The calculation time is still not completely satisfying, due to the high amount of offline data to compute, although, the method is highly parallelizable so the calculation time is expected to decrease. Future methods improvements might involve the use of online statistical tools to avoid offline computation of graph weights, in order to have a better management of the hardware resources used to run the algorithm.

CRediT authorship contribution statement

S. Mutti: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing

PREPRINT VERSION
PUBLISHED VERSION AT https://doi.org/10.1016/j.rcim.2021.102131
Robotics and Computer–Integrated Manufacturing 71 (2021) 102131

- review & editing. **G. Nicola:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing. **M. Beschi:** Conception and design of study, Writing - review & editing. **N. Pedrocchi:** Conception and design of study, Writing - original draft, Writing - review & editing. **L. Molinari Tosatti:** Conception and design of study, Writing - review & editing.

Declaration of competing interest

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.

Acknowledgment

All authors approved the version of the manuscript to be published.

Funding

Made4lo(Metal Additive for Lombardy) project by Regione Lombardia.

References

[1] BLM Group, 2020, www.blmgroup.com/it/lasertube/lt360. (Accessed 30 April 2020).
[2] Y. Chen, F. Dong, Robot machining: recent development and future research issues, Int. J. Adv. Manuf. Technol. 66 (9–12) (2013) 1489–1497.
[3] W. Ji, L. Wang, Industrial robotic machining: a review, Int. J. Adv. Manuf. Technol. 103 (1–4) (2019) 1239–1255.
[4] P. Urhal, A. Weightman, C. Diver, P. Bartolo, Robot assisted additive manufacturing: A review, Robot. Comput.-Integr. Manuf. 59 (2019) 335–345.
[5] Siemens, PLM NX, 2020, www.plm.automation.siemens.com. (Accessed 30 April 2020).
[6] ABB, RobotStudio, 2020, https://new.abb.com. (Accessed 30 Apri 2020).
[7] T. Yoshikawa, Manipulability and redundancy control of robotic mechanisms, in: Proceedings. 1985 IEEE International Conference on Robotics and Automation, Vol. 2, 1985, pp. 1004–1009.
[8] J. Angeles, C.S. López-Cajún, Kinematic isotropy and the conditioning index of serial robotic manipulators, Int. J. Robot. Res. 11 (6) (1992) 560–571.
[9] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotics, Modelling, Planning and Control, Springer, 2009.
[10] G. Legnani, D. Tosi, I. Fassi, H. Giberti, S. Cinquemani, The “point of isotropy” and other properties of serial and parallel manipulators, Mech. Mach. Theory 45 (10) (2010) 1407–1423.
[11] J. Lachner, V. Schettino, F. Allmendinger, M.D. Fiore, F. Ficuciello, B. Siciliano, S. Stramigioli, The influence of coordinates in robotic manipulability analysis, Mech. Mach. Theory 146 (2020) 103722, http://dx.doi.org/10.1016/j.mechmachtheory.2019.103722.
[12] Y. Lin, H. Zhao, H. Ding, Posture optimization methodology of 6R industrial robots for machining using performance evaluation indexes, Robot. Comput.-Integr. Manuf. 48 (2017) 59–72.
[13] S. Mousavi, V. Gagnol, B.C. Bouzgarrou, P. Ray, Stability optimization in robotic milling through the control of functional redundancies, Robot. Comput.-Integr. Manuf. 50 (2018) 181–192.
[14] J.T. Feddema, Kinematically optimal robot placement for minimum time coordinated motion, in: IEEE International Conference on Robotics and Automation Minneapolis, April, 1996.
[15] G. Pamanes, S. Zeghloul, Optimal placement of robotic manipulators using multiple kinematic criteria, in: Proceedings - IEEE International Conference on Robotics and Automation, Vol. 1, April, 1991, pp. 933–938.
[16] B. Kamrani, V. Berbyuk, D. Wäppling, U. Stickelmann, X. Feng, Optimal robot placement using response surface method, Int. J. Adv. Manuf. Technol. 44 (1–2) (2008) 201–210.
[17] R.R. dos Santos, V. Steffen, S.d.F. Saramago, Optimal task placement of a serial robot manipulator for manipulability and mechanical power optimization, Intell. Inf. Manag. 02 (09) (2010) 512–525.
[18] G.C. Vosniakos, E. Matsas, Improving feasibility of robotic milling through robot placement optimisation, Robot. Comput.-Integr. Manuf. 26 (5) (2010) 517–525.
[19] Y. Guo, H. Dong, Y. Ke, Stiffness-oriented posture optimization in robotic machining applications, Robot. Comput.-Integr. Manuf. 35 (2015) 69–76.
[20] G. Xiong, Y. Ding, L. Zhu, Stiffness-based pose optimization of an industrial robot for five-axis milling, Robot. Comput.-Integr. Manuf. 55 (2019) 19–28.
[21] T. Cvitanic, V. Nguyen, S.N. Melkote, Pose optimization in robotic machining using static and dynamic stiffness models, Robot. Comput.-Integr. Manuf. 66 (2020) 101992.
[22] M. Tay, B. Ngoi, Optimising robot workcell layout, Int. J. Adv. Manuf. Technol. 12 (5) (1996) 377–385.
[23] S.-W. Son, D.-S. Kwon, A convex programming approach to the base placement of a 6-DOF articulated robot with a spherical wrist, Int. J. Adv. Manuf. Technol. 102 (9–12) (2019) 3135–3150.
[24] G. Boschetti, R. Rosa, A. Trevisani, Optimal robot positioning using task-dependent and direction-selective performance indexes: General definitions and application to a parallel robot, Robot. Comput.-Integr. Manuf. 29 (2) (2013) 431–443.
[25] S. Mitsi, K.-D. Bouzakis, D. Sagris, G. Mansour, Determination of optimum robot base location considering discrete end-effector positions by means of hybrid genetic algorithm, Robot. Comput.-Integr. Manuf. 24 (1) (2008) 50–59.
[26] G. Nicola, N. Pedrocchi, S. Mutti, P. Magnoni, M. Beschi, Optimal task positioning in multi-robot cells, using nested meta-heuristic swarm algorithms, Procedia CIRP 72 (2018) 386–391, http://dx.doi.org/10.1016/j.procir.2018.03.081.
[27] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67.
[28] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39, http://dx.doi.org/10.1109/MCI.2006.329691.
[29] H. Chi, M. Mascagni, T. Warnock, On the optimal Halton sequence, Math. Comput. Simulation 70 (1) (2005) 9–21, http://dx.doi.org/10.1016/j.matcom.2005.03.004.
[30] R. Tijdeman, Review: L. Kuipers and H. Niederreiter, Uniform distribution of sequences, Bull. Amer. Math. Soc. 81 (1975).
[31] L. Kocis, W.J. Whiten, Computational investigations of low-discrepancy sequences, ACM Trans. Math. Software 23 (2) (1997) 266–294, http://dx.doi.org/10.1145/264029.264064.
[32] J. Matoušek, On theL2-discrepancy for anchored boxes, J. Complexity 14 (4) (1998) 527–556, http://dx.doi.org/10.1006/jcom.1998.0489.
[33] P. Pellegrini, D. Favaretto, E. Moretti, On MAX – MIN ant system’s parameters, in: International Conference on Swarm Intelligence, Springer, Berlin, Heidelberg, 2006, pp. 203–214, http://dx.doi.org/10.1007/11839088_18.
[34] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, R. Dillmann, Unified GPU voxel collision detection for mobile manipulation planning, in: IEEE International Conference on Intelligent Robots and Systems, 2014, pp. 4154–4160, http://dx.doi.org/10.1109/IROS.2014.6943148.
[35] ROS Industrial Consortium, 2020, https://rosindustrial.org. (Accessed 30 April 2020).
[36] OpenRAVE, 2020, http://openrave.org/docs/0.8.2/openravepy/ikfast/. (Accessed 9 August 2020).
[37] S. Mutti, G. Nicola, N. Pedrocchi, WACO, in: GitHub Repository, GitHub, 2020, https://github.com/CNR-STIIMA-IRAS/WACO.