OPTIMIZING ROBOT POSITIONING AGAINST PLACEMENT INACCURACIES: A STUDY ON THE FANUC CRX10IA/L

Nicolas Gautier [1,2], Yves Guillermit [1], Mathieu Porez [2], David Lemoine [2], Damien Chablat [2,*]

[1]Weez-U Welding, 101 rue de Coulmiers, 44 000 Nantes, France
[2]Nantes Université, École Centrale Nantes, IMT Atlantique, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

BSTRACT

This study presents a methodology for determining the optial base placement of a Fanuc CRX10iA/L collaborative robot or a desired trajectory corresponding to an industrial task. The roposed method uses a particle swarm optimization algorithm hat explores the search space to find positions for performing the rajectory. An α-shape algorithm is then used to draw the borders f the feasibility areas, and the largest circle inscribed is calcuated from the Voronoi diagrams. The aim of this approach is to rovide a robustness criterion in the context of robot placement naccuracies that may be encountered, for example, if the robot is laced on a mobile base when the system is deployed by an opertor. The approach developed uses an inverse kinematics model o evaluate all initial configurations, then moves the robot endffector along the reference trajectory using the Jacobian matrix nd assigns a score to the attempt. For the Fanuc CRX10iA/L obot, there can be up to 16 solutions to the inverse kinematics odel. The calculation of these solutions is not trivial and reuires a specific study that planning tools such as MoveIt cannot ully take into account. Additionally, the optimization process ust consider constraints such as joint limits, singularities, and orkspace limitations to ensure feasible and efficient trajectory xecution.

eywords: Welding application, serial robot, trajectory planing, simulation

1. INTRODUCTION

Optimization of a robot's placement for complex, sometimes one-off, industrial tasks such as welding or cutting represents a major challenge [1], particularly when the workpieces are large relative to the robot and the trajectory to be followed is highly constrained in terms of orientation. For example, a trajectory resulting from the intersection of two cylinders imposes strict requirements on both position and orientation, making it difficult to determine an optimal placement that ensures feasible movements. These constraints are even more critical as positioning inaccuracies can affect execution robustness, e.g. when the robot is mounted on a mobile platform. This issue remains a major challenge, and robot placement still relies largely on iterative testing and human intuition.

In the literature, several approaches have been proposed to address this problem. The approach presented in [2] is based on optimizing the base position of an industrial robot to minimize cycle time while ensuring the absence of collisions with the environment. Similarly, [3] focuses on optimizing the number of robots and their positioning to maximize paint coverage while reducing the risk of collisions between robots. Other studies have focused on minimizing the robot's dynamic effects. In [4], the authors propose to place the workpiece to minimize joint efforts, thereby improving task quality. Likewise, [5] aims to reduce joint torques in a milling application due to the low rigidity and high flexibility of robots compared to traditional CNC machines. For mobile manipulation tasks, [6] proposes a multi-objective optimization approach to improve coverage and the efficiency of the robot's placement on its mobile base. Finally, [7] develops a strategy that combines planning and heuristics to optimize robot placement in object-grasping tasks, taking into account robot reach and surrounding obstacles.

In this paper, we propose a new approach that combines kinematic trajectory simulation (including Inverse Kinematics Model (IKM), motion constraints, singularities, joint limits, and robot self-collision detection), workspace exploration using a Particle Swarm Optimization (PSO) algorithm, and the evaluation of the largest feasibility area. Our method stands out by introducing a robustness criterion to compensate for positioning inaccuracies, ensuring better adaptability to real execution conditions. Moreover, its computational efficiency allows quick integration of a robot into industrial environments requiring regular adjustments.

In the next two sections, we detail our methodology and the different steps of our approach. We then present the experimental results obtained before concluding with perspectives for improvement and future applications.

*Corresponding author: Damien.Chablat@cnrs.fr
Documentation for `asmeconf.cls`: Version 1.40, November 20, 2025.

2. SIMULATION OF ROBOTIC TRAJECTORIES

To model, simulate and plan a robotic task, the robot's trajectories in space are defined by a set of end-effector poses. Each pose includes the vector position $\mathbf{P} = [x, y, z]$ and the set of Euler angles $[\varphi, \theta, \psi]$ (following the 'Z-Y-X' convention) in the world frame. The pose p_i is represented by the homogeneous transformation matrix ${}^{W}\mathbf{T}_{E}^{i}$, as follow [8]

$$ {}^{W}\mathbf{T}_{E}^{(p_i)} = \begin{bmatrix} \mathbf{R}^{(p_i)} & \mathbf{P}^{(p_i)} \\ 0 & 1 \end{bmatrix}, \tag{1} $$

with

$$ \begin{aligned} \mathbf{R} &= Rot(z,\psi) \cdot Rot(y,\theta) \cdot Rot(x,\varphi) \\ &= \begin{bmatrix} C_\theta C_\psi & S_\varphi S_\theta C_\psi - C_\varphi S_\psi & C_\varphi S_\theta C_\psi + S_\varphi S_\psi \\ C_\theta S_\psi & S_\varphi S_\theta S_\psi + C_\varphi C_\psi & C_\varphi S_\theta S_\psi - S_\varphi C_\psi \\ -S_\theta & S_\varphi C_\theta & C_\varphi C_\theta \end{bmatrix}, \end{aligned} \tag{2} $$

where, C_α denotes the cosine of α, and S_α denotes the sine of α. For a pose of the robot base ${}^{W}\mathbf{T}_0$ (which is the parameter to optimize) and a configuration of the Terminal Center Point (TCP or end-effector) ${}^{6}\mathbf{T}_E$, each point of the trajectory can be expressed in the robot's base frame by the following relation:

$$ {}^{0}\mathbf{T}_{6}^{(p_i)} = {}^{W}\mathbf{T}_{0}^{-1} \cdot {}^{W}\mathbf{T}_{E}^{(p_i)} \cdot {}^{6}\mathbf{T}_{E}^{-1}. \tag{3} $$

In the paper, we will use the Fanuc CRX10iA/L robot as an example. The modified Denavit Hartenberg (DH) parameters of this robot (Khalil-Kleinfinger convention, see [9]) are given in Table 1.

i	d_i (m)	a_i (m)	α_i (rad)	θ_i (rad)
1	0.245	0	0	q_1
2	0	0	$\pi/2$	q_2
3	0	0.710	0	q_3
4	0.540	0	$-\pi/2$	q_4
5	0.150	0	$\pi/2$	q_5
6	0.160	0	$-\pi/2$	q_6

TABLE 1: MODIFIED DH PARAMETERS OF THE FANUC CRX10IA/L ROBOT.

In the rest of the document the position of joint i will be noted q_i. We will fix the values of the twist angles α, while the values of the joint-to-joint distance d and the offset a will remain symbolic. The developed equations will thus be valid for all robots with a similar kinematic architecture. It is worth noting that 6R anthropomorphic robotic arms with a spherical wrist are a special case of this architecture, where $d_5 = d_6 = 0$. For numerical applications, we will use the length values from the Fanuc CRX10iA/L robot. The joint limits of the Fanuc CRX10iA/L robot are given in Table 2 for this set of DH parameters.

i	1	2	3	4	5	6
min (rad)	$-\pi$	$-\pi/2$	$-\pi$	$-19\pi/18$	$-\pi$	$-5\pi/4$
max (rad)	π	$3\pi/2$	2π	$19\pi/18$	π	$5\pi/4$

TABLE 2: JOINT LIMITS OF THE FANUC CRX10IA/L ROBOT

Finally, to calculate the position of the end-effector according to its base, we use transformation matrices between each link. Between two bodies i and j, this matrix is given by:

$$ {}^{i}\mathbf{T}_{j} = \begin{bmatrix} C_{\theta_j} & -S_{\theta_j} & 0 & a_j \\ C_{\alpha_j} S_{\theta_j} & C_{\alpha_j} C_{\theta_j} & -S_{\alpha_j} & -d_j S_{\alpha_j} \\ S_{\alpha_j} S_{\theta_j} & S_{\alpha_j} C_{\theta_j} & C_{\alpha_j} & d_j C_{\alpha_j} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4} $$

2.1. Computation of inverse kinematic model

To create a continuous trajectory, we must avoid singularities, joint limits, and self-collision. However, these factors depend on the robot's initial position, and different initial positions can lead to different outcomes. Therefore, it is essential to evaluate all possible starting postures, i.e., all solutions to the inverse kinematics problem for the starting point of the trajectory. Inverse kinematics is a fundamental problem in robotics, which involves calculating the joint angles necessary for a robot to reach a given position and orientation of its end-effector. This problem is often complex, especially for robots with a cuspidal morphology [10], such as the Fanuc CRX10iA/L, which can have up to 16 distinct solutions, and for which no general analytical solution exists [11]. To address this problem, several methods have been proposed. One of them relies on the use of iterative algorithms that exploit higher-order derivatives to converge towards a solution [12]. Other approaches include polynomial resolution, such as the one applied to the Kinova Gen3 Lite manipulator, where a univariate polynomial equation in q_1 (i.e. the first joint position of the robot) is solved, with the other variables determined by backward substitution [13]. We used a method similar to the one employed for an anthropomorphic robot. According Paul's method, position and orientation are decomposed to separate the problem into independent solutions for q_1, q_2, and q_3, and then for q_4, q_5, and q_6 [8]. However, the wrist offset at joint 5 makes the position of the wrist dependent on q_4. To address this, we discretize q_4 over the interval $[-\pi, \pi]$ in order to compute q_1, q_2, and q_3. From a final equation derived from the rotation matrix, we can compute a residual that indicates a pair of solutions when it vanishes. This method has the advantage of obtaining the set of solutions to the inverse kinematics problem, provided that the discretization step of q_4 is fine enough. It allows solving 4 equations instead of a single one of degree 16. This is particularly useful when using non-mathematical programming languages such as Python or C.

The position equations of ${}^{0}\mathbf{T}_5$, centre of the wrist, are given by:

$$ \begin{cases} P_x = x - d_6 r_{13} \\ P_y = y - d_6 r_{23} \\ P_z = z - d_6 r_{33} \end{cases}, \tag{5} $$

where r_{ij} is the component in row i and column j of the rotation matrix $\mathbf{R}$. By pre-multiplying (5) by ${}^{1}\mathbf{T}_0$, the position equations (5) become:

$$ \begin{cases} C_1 P_x + S_1 P_y = a_3 C_2 - d_4 S_{23} + d_5 C_{23} S_4 \\ -S_1 P_x + C_1 P_y = -d_5 C_4 \\ P_z - d_1 = d_4 C_{23} + a_3 S_2 + d_5 S_{23} S_4 \end{cases}. \tag{6} $$

The second equation of (6) is of the form $XS_1 + YC_1 = Z$, where X, Y and Z are generalized variables. The solutions to solve for q_1 are given by:

$$q_1 = \text{atan2}(\text{S}_1, \text{C}_1) \ , \tag{7}$$

where:

$$\begin{aligned} S_1 &= \frac{XZ + \epsilon_1 Y\sqrt{X^2+Y^2-Z^2}}{X^2+Y^2} \ , \text{and} \\ C_1 &= \frac{YZ - \epsilon_1 X\sqrt{X^2+Y^2-Z^2}}{X^2+Y^2} \ , \end{aligned}$$

with $\epsilon_1 \in \{-1, 1\}$. Since q_1 is known, we can pre-multiply our matrix ${}^1\mathbf{T}_5$ by ${}^2\mathbf{T}_1$. The position equations become:

$$\begin{cases} (P_z - d_1)S_2 + (P_xC_1 + P_yS_1)C_2 = a_3 - d_4S_3 + d_5S_4C_3 \\ (P_z - d_1)C_2 - (P_xC_1 + P_yS_1)S_2 = d_4C_3 + d_5S_4S_3 \\ P_xS_1 - P_yC_1 = d_5C_4 \end{cases} ,$$

which can be written in the form:

$$\begin{cases} W_1C2 + W_2S_2 = XC_3 + YS_3 + Z_1 \\ W_1S2 - W_2C_2 = -XS_3 + YC_3 \end{cases} .$$

By squaring and adding the two equations, we are led to solve an equation of the form $B_1S_3 + B_2C_3 = B_3$ with $B_1 = 2Z_1Y$, $B_2 = 2Z_1X$, and $B_3 = W_1^2 + W_2^2 - X^2 - Y^2 - Z_1^2$. The solutions for q_3 are given by:

$$q_3 = \text{atan2}(\text{S}_3, \text{C}_3) \ , \tag{8}$$

where:

$$\begin{aligned} S_3 &= \frac{B_1B_3 + \epsilon_2 B_2\sqrt{B_1^2+B_2^2-B_3^2}}{B_1^2+B_2^2} \ , \text{and} \\ C_3 &= \frac{B_2B_3 - \epsilon_2 B_1\sqrt{B_1^2+B_2^2-B_3^2}}{B_1^2+B_2^2} \ , \end{aligned}$$

with $\epsilon_2 \in \{-1, 1\}$. Knowing q_3, the two equations previously used in (8) become of the form:

$$\begin{cases} X_1S_2 + Y_1C_2 = Z_1 \\ X_2S_2 + Y_2C_2 = Z_2 \end{cases} . \tag{9}$$

The solution for q_2 is:

$$q_2 = \text{atan2}(\text{S}_2, \text{C}_2) \ , \tag{10}$$

where:

$$S_2 = \frac{Z_1Y_2 - Z_2Y_1}{X_1Y_2 - X_2Y_1} \ , \text{and } C_2 = \frac{Z_2X_1 - Z_1X_2}{X_1Y_2 - X_2Y_1} \ .$$

To obtain a new equation to calculate the residual of q_4, we use the orientation equations of the rotation matrix ${}^3\mathbf{R}_6$. The r_{13} and r_{33} components of this matrix give us the following equations:

$$\begin{cases} -C_4S_5 = r_{33}S_{23} + r_{13}C_1C_{23} + r_{23}S_1C_{23} \\ S_4S_5 = r_{13}S_1 - r_{23}C_1 \end{cases} . \tag{11}$$

By multiplying the equations of (11) by S_4 and C_4 respectively and adding them, we obtain the following equation, named residual equation and denoted by G:

$$\begin{aligned} G(q_4, \epsilon) &= S_4(r_{33}S_{23} + r_{13}C_1C_{23} + r_{23}S_1C_{23}) \\ &+ C_4(r_{13}S_1 - r_{23}C_1) \ . \end{aligned} \tag{12}$$

According to the four possible combinations of sign coefficients $[\epsilon_1, \epsilon_2]$, from (12), we obtain four equations. When the residual equation is zero, the quadruple $\{q_1, q_2, q_3, q_4\}$ constitutes a solution of the inverse kinematic. Each equation has at most four solutions. Figure 1 illustrates an example of residual evolutions as a function of q_4. This example is given for the pose $[x, y, z, \varphi, \theta, \psi]$ = $[-0.2406, -0.1188, 0.5603, 2.6204, 1.1236, 0.4276]$ where 16 real solutions exist, as each of the four equations has four real solutions.

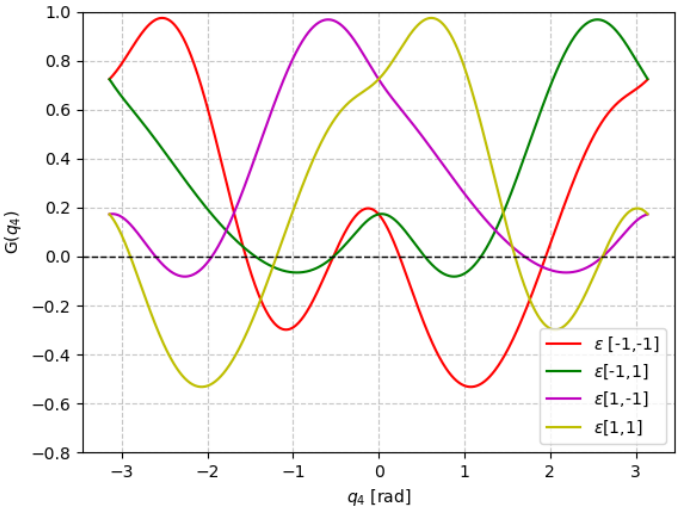


FIGURE 1: FOUR RESIDUAL CURVES CORRESPONDING TO THE FOUR COUPLES $[\epsilon_1, \epsilon_2]$ FOR THE POSE $[-0.2406, -0.1188, 0.5603, 2.6204, 1.1236, 0.4276]$ AS A FUNCTION OF q_4

.

To obtain the last four equations to calculate q_5 and q_6, we use the orientation equations of the rotation matrix ${}^4\mathbf{R}_6$. The r_{13} and r_{33} components determine q_5:

$$q_5 = \text{atan2}(\text{S}_5, \text{C}_5) \ , \tag{13}$$

where:

$$\begin{aligned} S_5 &= S_4(r_{13}S_1 - r_{23}C_1) - C_4(r_{33}S_{23} + C_{23}(r_{13}C_1 + r_{23}S_1)) \ , \\ C_5 &= r_{33}C_{23} - S_{23}(r_{13}C_1 + r_{23}S_1) \ , \end{aligned}$$

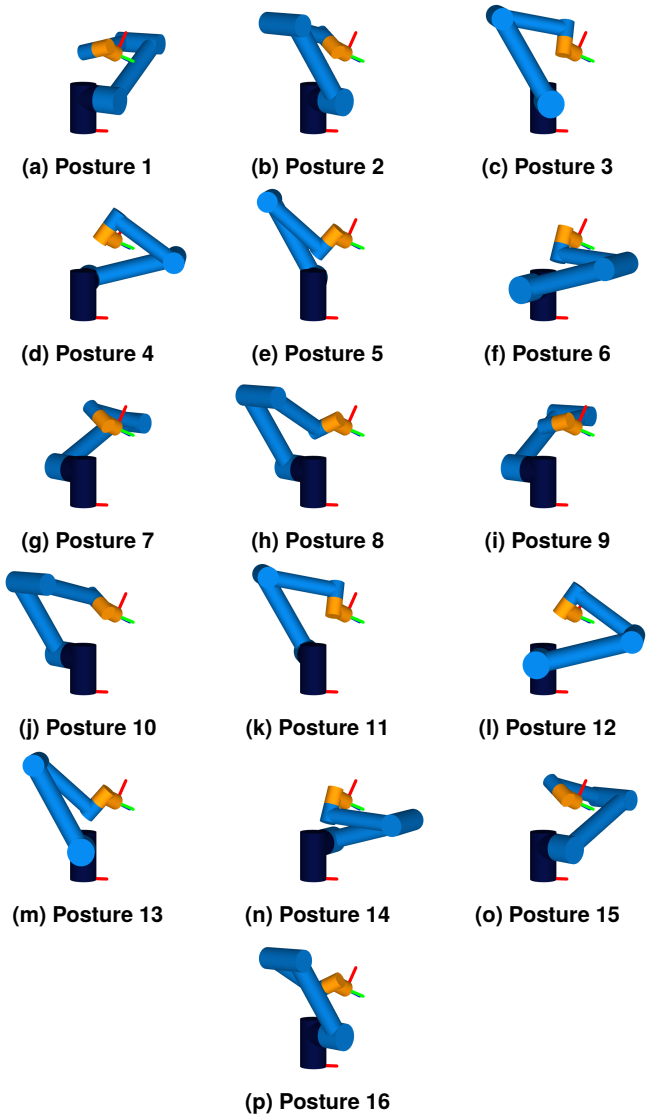
and the r_{21} and r_{22} components determine q_6:

$$q_6 = \text{atan2}(\text{S}_6, \text{C}_6) \ , \tag{14}$$

where:

$$\begin{aligned} S_6 &= C_4(r_{21}C_1 - r_{11}S_1) - r_{31}S_{23}S_4 \\ &- C_{23}S_4(r_{11}C_1 + r_{21}S_1) \ , \text{and} \\ C_6 &= C_4(r_{22}C_1 - r_{12}S_1) - r_{32}S_{23}S_4 \\ &- C_{23}S_4(r_{12}C_1 + r_{22}S_1) \ . \end{aligned}$$

The solutions for the previous example are given in Table 3, and Figure 2 illustrates the corresponding robot postures.



(a) Posture 1 (b) Posture 2 (c) Posture 3

(d) Posture 4 (e) Posture 5 (f) Posture 6

(g) Posture 7 (h) Posture 8 (i) Posture 9

(j) Posture 10 (k) Posture 11 (l) Posture 12

(m) Posture 13 (n) Posture 14 (o) Posture 15

(p) Posture 16

FIGURE 2: VISUALIZATION OF THE 16 INVERSE KINEMATIC SOLUTIONS FOR THE POSE [−0.2406, −0.1188, 0.5603, 2.6204, 1.1236, 0.4276].

2.2. Validation of a robot trajectory

For each initial posture, we will execute the trajectory and evaluate, at each point, the proximity to the robot's singularities and joint limits. To move the robot, we use the kinematic Jacobian matrix $\mathbf{J}^{-1}$, which relates the kinematic twist ${}^0\mathbb{V}_E$ (of the end effector expressed in the frame attached to the base) to the joint velocities vector $\dot{\mathbf{q}}$ (see [14]):

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\,{}^0\mathbb{V}_E \,. \tag{15}$$

The i^{th} column of this Jacobian matrix for a revolute joint can be computed from the Direct Kinematic Model (DKM), with ${}^0\mathbf{P}_i$ being the position vector of frame i in the robot's base frame and

i	q_1	q_2	q_3	q_4	q_5	q_6
1	1.108	2.597	2.062	-2.896	2.690	0.498
2	0.900	0.953	0.858	-2.601	-0.169	-0.334
3	0.312	1.050	0.713	-1.941	-0.679	-1.167
4	3.129	0.259	0.688	-1.546	2.241	-1.836
5	-3.050	2.077	1.912	-1.436	0.824	1.165
6	-0.393	2.850	1.922	-1.186	-1.762	1.817
7	-2.230	0.461	0.861	-0.526	2.983	-0.318
8	-2.226	2.191	2.003	-0.521	0.150	0.721
9	-2.035	0.543	1.080	0.248	2.691	0.500
10	-2.240	2.188	2.283	0.538	-0.168	-0.332
11	-2.827	2.090	2.428	1.198	-0.677	-1.165
12	-0.013	2.881	2.452	1.593	2.240	-1.837
13	0.089	1.063	1.229	1.703	0.826	1.165
14	2.749	0.290	1.219	1.953	-1.764	1.817
15	0.909	2.680	2.280	2.613	2.983	-0.320
16	0.913	0.949	1.138	2.618	0.151	0.722

TABLE 3: 16 INVERSE KINEMATIC SOLUTIONS FOR THE POSE [−0.2406, −0.1188, 0.5603, 2.6204, 1.1236, 0.4276] GIVEN IN RADIANS.

the local vector z_i expressed in the robot's base frame[1]:

$${}^0\mathbf{J}_{E_i} = \begin{bmatrix} {}^0\hat{z}_i({}^0\mathbf{P}_n - {}^0\mathbf{P}_i) \\ {}^0z_i \end{bmatrix} . \tag{16}$$

In our application, we are not concerned with the joint velocities but rather with their successive positions. We proceed iteratively by approximating the integral using the explicit Euler method. At each iteration, we move by a small pose $\mathbf{dk}$ (which is a change in position and orientation of the end effector), i.e:

$$\mathbf{q}_{j+1} \simeq \mathbf{q}_j + \mathbf{J}^{-1}(\mathbf{q}_j)\mathbf{dk} \,. \tag{17}$$

The trajectory is represented by a set of points connected by segments. Along these segments, change in position and orientation are assumed to be uniform. The position increment step dp is chosen to guarantee an acceptable accuracy of motion [15]. This implies $|\mathbf{dk}_P| = dp$, where $\mathbf{dk}_P$ denotes the position component of $\mathbf{dk}$. The intensity of the orientation change $\mathbf{dk}_\omega$ is defined as performing the orientation change over the same number of increments as the position along the segment. For a given segment, we can introduce $\mathbf{p}_{p_{i+1}} - \mathbf{p}_{p_i}$ the change of position between pose p_i and p_{i+1} and $\mathbf{R} = \mathbf{R}_{p_{i+1}}\mathbf{R}_{p_i}^T$ the change of rotation. The number of simulation steps N and the variations in $\mathbf{dk}$ are given by:

$$\mathbf{N} = \frac{|\mathbf{p}_{p_{i+1}} - \mathbf{p}_{p_i}|}{dp} \,, \mathbf{dk}_P = \frac{\mathbf{p}_{p_{i+1}} - \mathbf{p}_{p_i}}{N} \,, \text{ and } \mathbf{dk}_\omega = \frac{\theta \mathbf{u}}{N} \,,$$

where θ and $\mathbf{u}$ are the representation of the rotation matrix $\mathbf{R}$ in the form of a rotation angle and a rotation axis. It can be computed using Rodrigues formula [16]:

$$\theta = \text{acos}\left(\frac{\text{trace}(\mathbf{R}) - 1}{2}\right), \text{ and } \mathbf{u} = \frac{1}{2S_\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} . \tag{18}$$

[1]The 'hat' notation is (3×3) asymmetric tensor associated to (3×1) mentioned vector. For any vectors U and V of $\mathbb{R}^3$, $\hat{U}$ is defined such that $\hat{U}V = U \times V$.

Along the trajectory, we will assess its feasibility by ensuring that none of the robot's joints exceeds the joint limits given in Table 2, that the determinant of the Jacobian matrix does not become singular, which would result in the loss of mobility and very high joint speeds [17], and that the robot does not self-collide (see the next section). Algorithm 1 summarizes the procedure to simulate and plan a trajectory.

Algorithm 1 SIMULATE A TRAJECTORY

Require: ${}^{W}\mathbf{T}_{p}, {}^{W}\mathbf{T}_{0}, {}^{6}\mathbf{T}_{E}$
  for $i \in N_{points}$ **do**
    ${}^{0}\mathbf{T}_{6}^{(i)} = {}^{W}\mathbf{T}_{0}^{-1}.{}^{W}\mathbf{T}_{p}^{(i)}.{}^{6}\mathbf{T}_{E}^{-1}$
  end for
  postures = IKM(${}^{0}\mathbf{T}_{6}^{(0)}$)
  for $\mathbf{q}_0 \in$ postures **do**
    $\mathbf{q}_j = \mathbf{q}_0$
    for $i \in [0, N_{points} - 1]$ **do**
      $[N_{incr}, dk] = f({}^{0}\mathbf{T}_{6}^{(p_i)}, {}^{0}\mathbf{T}_{6}^{(p_{i+1})}, dp)$
      for $\_ \in [0, N_{incr}]$ **do**
        $\mathbf{q}_{j+1} = \mathbf{q}_j + \mathbf{J}^{-1}(\mathbf{q}_j)\mathbf{dk}$
        $d_{sing} = \det(\mathbf{J}(\mathbf{q}_{j+1}))$
        $d_{lim} = \min(\text{dist}(\mathbf{q}_{j+1}, \mathbf{q}_{limit}))$
        coll = robot_self_collision($\mathbf{q}_{j+1}$)
      end for
    end for
  end for

2.3. Detection of robot self-collisions

Along the trajectory, the robot may self-collide even if it remains within its joint limits, e.g. when the elbow is bent towards the base. It is important to determine these configurations, which will prevent the trajectory from being executed on the physical robot. Moreover, these calculations must be kept as simple as possible, to avoid a significant increase in computation times.

We adopted a method based on the computation of the minimum distance between two segments. To do this, we implemented an algorithm proposed by [18], designed to reduce the number of calculations required. This algorithm proceeds as follows:

1. It begins by extending the segments into straight lines and identifying the points that minimize the distance between them. This is done using the parametric representation of each line to compute the distance. The partial derivative is zero for the minimum distance.

2. If these points do not lie on the segments, the first point is moved to the nearest extremity, and the nearest corresponding point on the other line is then computed.

3. If the second point is not on the segment, it is shifted to its nearest extremity, and a new corresponding point is determined on the first line.

Remark, in the case of parallel lines, select one end of the first segment and go directly to step 3.

The developed method is based on the successive computation of the minimum distances between robot segments. First, the coordinates of the segment extremities are obtained using the DKM, which provides the positions of the joint centres and the end effector. The algorithm then proceeds iteratively: for each segment, it computes the minimum distance to other segments that are not successive (and have not yet been evaluated). The robot's segments are modeled as capsules, and the radius of each capsule is subtracted to obtain the final minimum distance. If one of these distances is smaller than or equal to zero, the robot is in collision. The whole process is summarized in the algorithm 2.

Algorithm 2 ROBOT SELF-COLLISIONS

Require: $\mathbf{q}$
  $\mathbf{P}$ = DKM($\mathbf{q}$)
  for $i \in [0, N-2]$ **do**
    for $j \in [i+2, N]$ **do**
      $\text{Seg}_i = (\mathbf{P}_i, \mathbf{P}_{i+1})$
      $\text{Seg}_j = (\mathbf{P}_j, \mathbf{P}_{j+1})$
      dist_{min} = min_distance_segments($\text{Seg}_i, \text{Seg}_j$)
      if $\text{dist}_{min} \leq R_{\text{caps}_i} + R_{\text{caps}_j}$ **then**
        return True
      end if
    end for
  end for
  return False

3. OPTIMIZATION OF THE ROBOT'S POSITION

Using the developed simulator, we can assess the feasibility of a trajectory for a given robot configuration, defined by the position and orientation of its base and TCP. This allows us to explore various configurations and give them a score. In our application, the robot base is mounted on a mobile support, controlled by the user. Our goal is to assist him in selecting the optimal placement. As the positioning accuracy is approximate, we prioritize an approach that is robust to these inaccuracies rather than focusing solely on singularities and joint limits. Specifically, our aim is to identify the centre of the largest inscribed circle within the trajectory's feasibility region.

3.1. Exploring configuration spaces with a PSO algorithm

The search space is potentially vast, it is not conceivable to exhaustively evaluate all possible configurations in a production environment as the simulation time of a configuration is too high. To overcome this constraint, we have used a PSO algorithm[19] to efficiently explore the search space. The aim is first to investigate the whole space, then to concentrate on areas of interest, i.e. those where the trajectory is feasible. Particle swarm optimization is inspired by the collective behavior of birds in flight and schools of fish. This algorithm relies on a population of particles exploring the search space to identify an optimal solution. Each particle represents a candidate solution and is characterized by:

1. An $\mathbf{x_i}$ position: corresponds to a potential solution to the problem. Note that the orientation angle ψ (see (2) is chosen so that the robot faces the workpiece.

2. A velocity $\mathbf{v_i}$: controlling the particle's displacement in the search space.

3. A personal history $\mathbf{p}_{best_i}$: the best position reached by the particle.

4. A local history $\mathbf{l}_{best_i}$: the best position found by particles in its neighbourhood at iteration i.

At each iteration, the position and velocity of each particle are updated according to the following equations[2]:

$$\begin{aligned} \mathbf{v}_{i+1} &= w\mathbf{v}_i + c_1\mathbf{r}_{i1} \circ (\mathbf{p}_{best_i} - \mathbf{x}_i) \\ &+ c_2\mathbf{r}_{i2} \circ (\mathbf{l}_{best_i} - \mathbf{x}_i) + e\mathbf{r}_{ie} \text{ , and} \end{aligned} \tag{19}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1} \text{ ,} \tag{20}$$

where w, c_1, c_2 and e are inertia, cognition, social, and exploration coefficients respectively, while $\mathbf{r}_{i1}, \mathbf{r}_{i2}$ and $\mathbf{r}_{ie}$ are random vectors of $\mathbb{R}^n$ uniformly distributed in $[0, 1]^n$. The subset of particles forming a neighborhood is defined by a structure called topology.

Topology defines the way particles communicate with each other to share information about the best solutions found. It has a significant influence on the convergence rate and exploration ability of the algorithm. Static topologies are not suitable for all optimization problems [21]. Thus, we used a dynamic topology called DCluster [20], which combines two existing approaches: Four-clusters and Fitness. The DCluster topology is based on a dynamic segmentation of the swarm into several sub-swarms of equal size, called clusters. At each iteration, the topology is built in four steps:

1. Evaluate and sort particles according to their score for the current position, in ascending order.

2. Split the sorted swarm into several clusters of equal size, where the last cluster contains the best particles.

3. Fully connect particles in the same cluster, to create strongly connected subgraphs.

4. Create a central cluster, composed of the worst-performing particles, and connect each of them to the worst particle in a higher cluster.

This topology implies a specific swarm size given by the following equation:

$$S = N \times (N + 1) \text{ ,} \tag{21}$$

where N is the number of particles per cluster and S the total number of particles. This structure promotes efficient exploration of the search space by slowing down the propagation of information between clusters, thus reducing the risk of premature convergence towards a local optimum.

In our specific problem, the function to minimise corresponds to the percentage of the reference trajectory that could not be completed. This percentage is evaluated in terms of the number of segments covered. In contrast to conventional PSO applications, the minimum value is known, and the optimal solution is not necessarily reduced to a single point, but rather to a set of surfaces. Consequently, the objective is not to converge on a single solution, but to explore the whole space before focusing on the boundaries of the feasible zones. To encourage this exploration, we have introduced an exploratory factor e, see (19). Moreover, because we know in advance the space to be explored, we have adopted a different approach to conventional PSO algorithms. The particles are initially distributed uniformly along the boundaries of the space, forming a circle. The initial velocity is directed towards the center of the trajectory. This strategy aims to maximize coverage of the search space and to avoid missing any zones of interest. The algorithm 3 synthesizes the PSO method used to explore the parameter space.

Algorithm 3 PARTICLE SWARM OPTIMISATION ALGORITHM

Initialize particle positions $\mathbf{x}_0$ uniformly on the search space boundary
Set corresponding velocities $\mathbf{v}_0$ towards the trajectory center.
Evaluate $f(\mathbf{x}_0)$
for each iteration i **do**
  Sort particles by fitness $f(\mathbf{x}_i)$ in ascending order and divide them into N clusters of equal size
  Fully connect particles within each cluster
  Connect each particle from the worst cluster to the worst particle of a superior cluster
  for each particle **do**
    $\mathbf{l}_{best_i} = \mathbf{p}_{best_{i,k}}$ where $P_k = \text{argmin}\left(f(\mathbf{p}_{best_{i,j}}), P_j \in \text{neighborhood}\right)$
    $\mathbf{v}_{i+1} = w\mathbf{v}_i + c_1\mathbf{r}_{i1} \circ (\mathbf{p}_{best_i} - \mathbf{x}_i) + c_2\mathbf{r}_{i2} \circ (\mathbf{l}_{best_i} - x_i) + e\mathbf{r}_{ie}$
    $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1}$
  end for
  for each particle **do**
    Evaluate $f(\mathbf{x}_i)$
    if $f(\mathbf{x}_i) \leq f(\mathbf{p}_{best_i})$ **then**
      $\mathbf{p}_{best_i} = \mathbf{x}_i$
    end if
  end for
end for

3.2. Clustering and determination of the largest inscribed circle

The PSO algorithm allows us to efficiently explore the search space and identify areas of trajectory feasibility. We can then analyse these areas to determine the center of the largest inscribed circle. After filtering the point cloud to retain only the relevant points, we draw the polytopes delimiting these regions. To do this, we use the α-shape algorithm [22], which extracts the envelope of a 2D point cloud. Unlike the classic convex hull method, α-shape offers more flexibility by avoiding convex over-approximations and by adapting better to the real geometry of the data. This can be done by controlling the level of concavity with a parameter α. The method also identifies several clusters of points, representing different regions accessible to the robot, as well as any holes in the point cloud, which may represent under-explored areas or real regions of infeasibility.

The algorithm proposed in this paper is based on the following principle: a sphere of radius α explores the space containing

[2]The $\circ$ notation is the Hadamard product i.e. element-wise product of vectors or matrices [20].

the cloud points. Two points are connected by an edge in the α-shape if there is a disk of radius α containing them on its boundary without including any other points inside. The set of edges then forms the polytope delimiting the point cloud. A method based on Delaunay triangulation [23] can be:

1. Construct the Delaunay triangulation for all points S.
2. For each simplex, calculate its circumscribed radius r.
3. Include the simplex in the α-complex if $r \leq \alpha$.
4. The resulting α-complex boundary forms the α-shape.

Once the polytopes have been determined for each cluster, we can then proceed to determine the largest circle inscribed in each polytope. To do this, we use an approach based on Voronoi diagrams [24]:

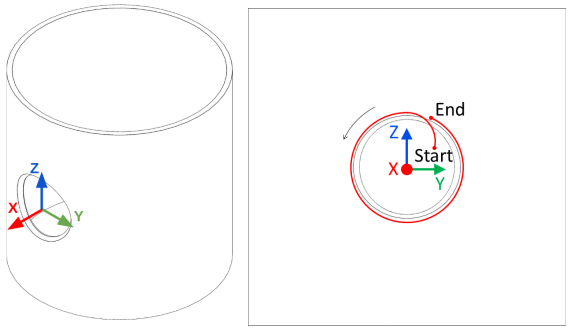
1. Calculate the Voronoi diagram of the polygon.
2. Identify diagram nodes inside the polygon.
3. For each internal node, measure the minimum distance to the polygon's edges.
4. Select the node with the maximum distance, defining the center of the largest inscribed circle, with this distance corresponding to the radius.

In order to obtain a Voronoi diagram representing the whole polygon and not just its vertices, we discretize the polygon edges. Note that this method can also be applied to non-convex polygons with holes. Clusters with a maximum radius of less than 0.05 m are not taken into account, as the margins are too small to be placed by a user.

4. EXPERIMENTATIONS

This section presents the results obtained for an industrial plasma cutting application. Specifically, it focuses on a trajectory resulting from the intersection of two cylinders. This application can be found, for example, in the fields of shipbuilding, boiler making, or metal infrastructures, for the assembly of piping networks or tubular structures. The objective is to cut the larger diameter tube so that the second tube can be fitted into it, ensuring a precise fit for subsequent welding operations. The trajectory is shaped like a horse saddle and will be referred to as "saddle" in the remainder of this document. In addition to this base trajectory, a first lead-in is used to allow time to pierce the thickness of the tube and reach the circle tangent, thus ensuring a clean cut and avoiding geometrical imperfections related to the cutting process. The path positions depend on the diameters of the two tubes and the angle of inclination between them. The path orientation, on the other hand, depend on a α bevel angle (determined according to the requirements of the welding process) and the presence or absence of a camera in the cutting axis.

In this section, we consider an example where the diameter of the larger tube is 1 m and the smaller tube is 0.3m, the larger tube is vertical and the smaller is perpendicular. The TCP frame ${}^{6}\mathbf{T}_{E}$ used corresponds to the pose [0.038, 0, 0.409, π/4,0,-π/2].



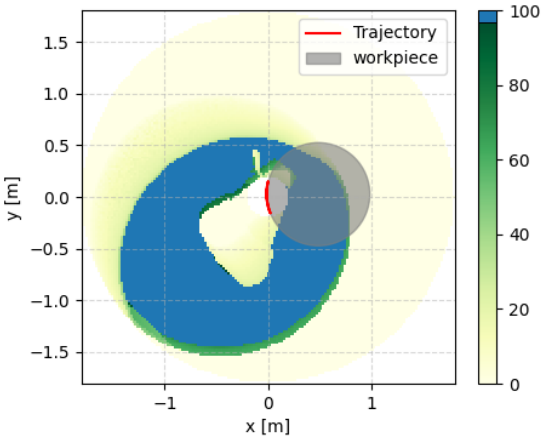
(a) Workpiece **(b) Trajectory**

FIGURE 3: WORKPIECE AND ITS ASSOCIATED SADDLE-SHAPED TRAJECTORY.

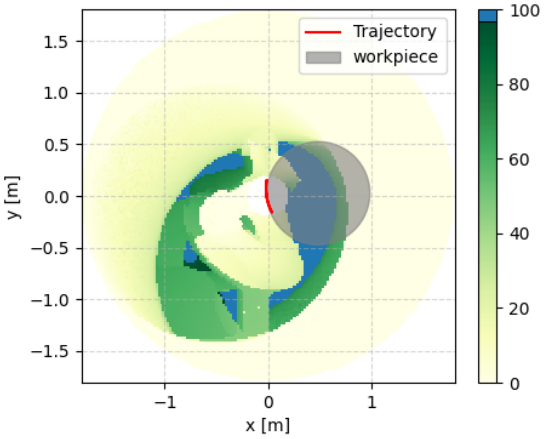
Figure 3 shows the final workpiece and a visualization of the trajectory to be performed. For a given tube configuration i.e., the radius of the cylinders and their relative orientation. The choice of angles can significantly influence the trajectory's feasibility zones. As an example, Figure 4 shows three different configurations of chamfer angles β and camera used: (a) 15° chamfer, without camera in axis; (b) 45° chamfer, without camera in axis, and (c) 15° chamfer, with camera in axis. This last property allows the user to control the execution of the trajectory and adapt it in terms of position and speed in real-time. Each point in Figure 4 represents the percentage of the trajectory that can be achieved before reaching a non-feasibility factor, such as a singularity, a joint limit, or a collision, for the best-associated posture. The blue dots represent the points where the entire trajectory is achievable. Figure 4 shows the different feasibility zones in the x-y plane for a height $z = -0.12$m from the center of the trajectory (center of the circle). The trajectory is projected onto the plane in red and the cylinder is represented by the grey circle. Note that the robot cannot be placed in this area. The results obtained highlight the difficulties the user may face when placing the robot. In fact, feasibility zones are highly dependent on the trajectory to be achieved, and a simple modification of a parameter can lead to a drastic change in their shape and size. The computation times required to exhaustively simulate all combinations are not compatible with online programming. The approach developed in this article, based on a PSO algorithm, enables us to efficiently explore the entire search space and concentrate on the boundaries of feasibility zones.

We have applied this algorithm to the examples presented above. Figure 5 illustrates the set of combinations explored by the PSO algorithm for previous examples. The blue dots represent configurations where the trajectory was entirely feasible, while a gradient from green to yellow indicates the score of the cost function in ascending order. In these examples, the parameters of the PSO algorithm used are: $N = 4$, $iter = 50$, $w = 0.8$, $c_1 = 0.35$, $c_2 = 0.15$, and $e = 0.2$. For comparison, simulation times are less than 3 minutes (1000 configurations tested), while they can reach 1 hour for exhaustive exploration ($\simeq 20,000$ configurations tested). In each simulation, the particles first explored the entire space before focusing on the areas of interest.

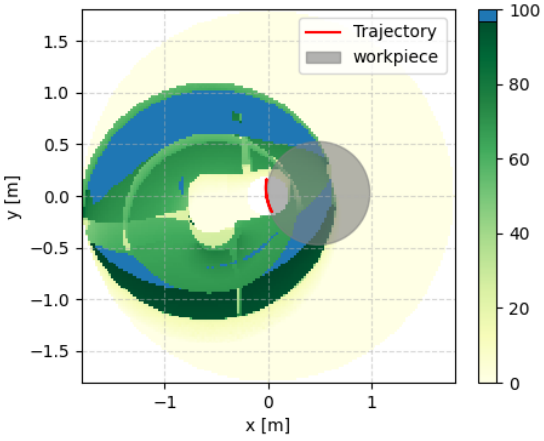
From the positions explored, retaining only those for which



(a) $\beta = 15^o$

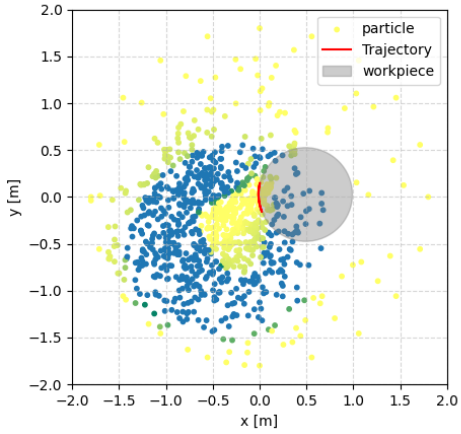


(b) $\beta = 45^o$

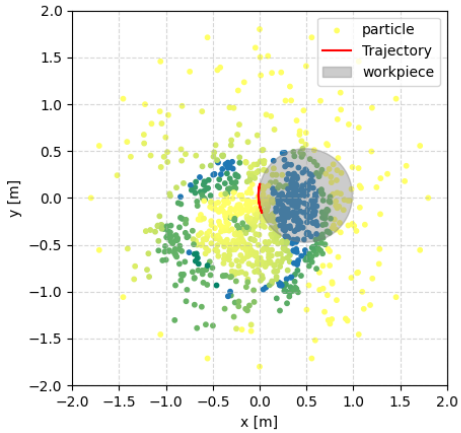


(c) $\beta = 15^o$ and camera in axis

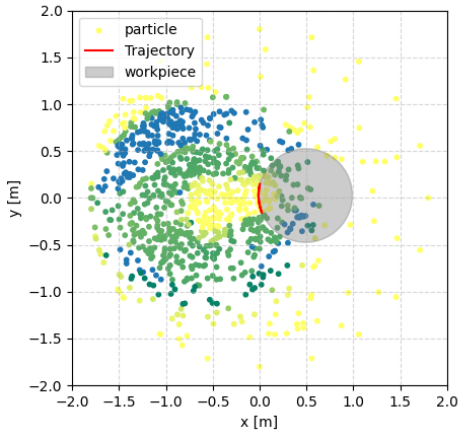
FIGURE 4: PERCENTAGE OF FEASIBLE TRAJECTORY FOR DIFFERENT SADDLE-SHAPED TRAJECTORIES.



(a) $\beta = 15^o$



(b) $\beta = 45^o$



(c) $\beta = 15^o$ and camera in axis

FIGURE 5: EXPLORATION WITH THE PSO ALGORITHM ON THE PREVIOUS EXAMPLES: GRADIENT FROM GREEN TO YELLOW INDICATES THE SCORE IN ASCENDING ORDER WHEN BLUE IS THE MINIMAL SCORE.

the trajectory is feasible, we can construct the α-shape and compute the largest inscribed circles. Figure 6 shows the results obtained for the previous configurations. For these examples, we

used a parameter α = 0.05m. In each case, the main cluster was correctly identified, and the largest inscribed circle corresponds closely to the one that would have been obtained using exhaustive discretization (see Figure 4). The center and radius of this circle provide valuable assistance to the user in the placement of the robot, indicating both the optimum position and the tolerance margin available. For example, in case (c), the algorithm returns the point x=-1.37m, y=-0.66m and with a margin of 0.18m. The positions returned by the algorithm were tested on the physical Fanuc robot. In each case, the trajectory was achieved without any problems. Note that for the second example, the second cluster was used because the first one was behind the workpiece.

5. CONCLUSIONS

In this study, we have developed an innovative approach to optimizing the placement of robotic arms, using the Fanuc CRX10iA/L robot as a case study. The trajectory simulation incorporates motion constraints, singularities, joint limits, and robot self-collision detection to ensure feasibility and efficiency. By integrating kinematic simulation, a particle swarm optimization algorithm, and feasibility zone identification using the α-shape algorithm, we have developed an efficient and robust method for optimizing robot placement.

Experimental results have convincingly demonstrated that our approach not only effectively identifies feasibility zones but also provides a robust criterion for handling initial placement uncertainties. Its industrial application to a plasma cutting task has confirmed the method's relevance, significantly reducing calibration time and improving the reproducibility of robot-executed trajectories. Moreover, cutting under real shop conditions was successfully executed using the Weez-U welding mobile base, as illustrated in Figure 7.

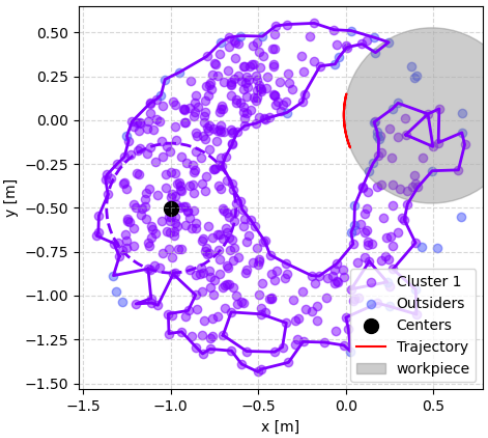
Future directions for this study include integrating the workspace into the simulation to account for complex parts a robot may need to manipulate. Additionally, extending this approach to other optimization parameters will assist robotic integrators in application design, particularly in selecting a robust TCP. Finally, optimizing computation times by translating the Python code into a lower-level language and parallelizing simulations would enable the exploration of more parameter dimensions without excessive computational costs. This would also allow fast optimal robot positioning, enhancing its usability in online programming.

ACKNOWLEDGEMENT

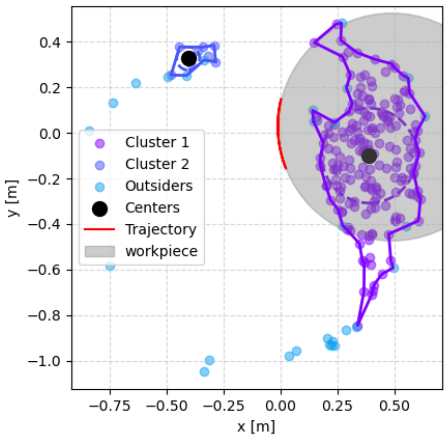
This research was supported by ANRT CIFRE grant n°2023 /1565 which funded the first author's doctoral studies.

REFERENCES

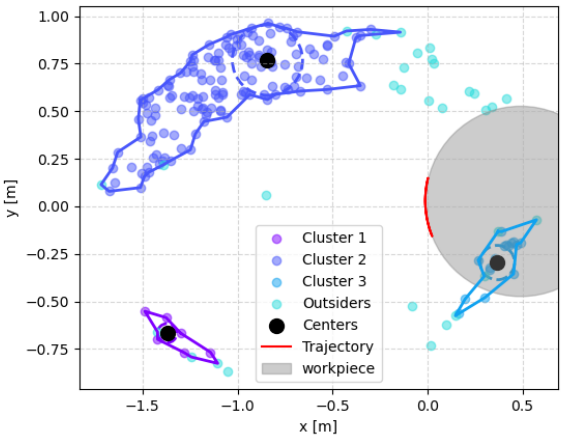
[1] Nicolas Gautier, Yves Guillermit, Yazid Sebsadji, Mathieu Porez, and Damien Chablat. Comparison of robot morphologies and base positioning for welding applications. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 88414, page V007T07A026. American Society of Mechanical Engineers, 2024.



(a) β = 15^o



(b) β = 45^o



(c) β = 15^o and camera in axis

FIGURE 6: CLUSTERS CALCULATED WITH THE α-SHAPE ALGORITHM AND THEIR LARGEST INSCRIBED CIRCLES ON THE PREVIOUS EXAMPLES.

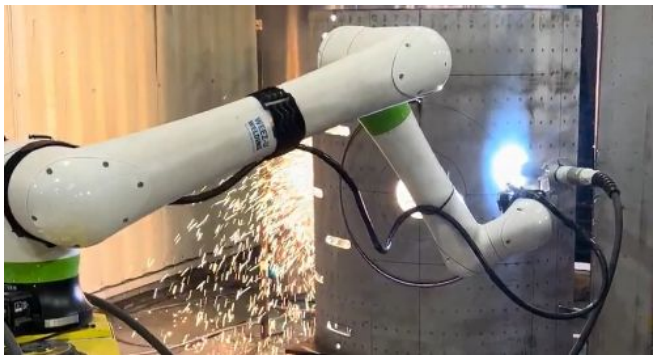


FIGURE 7: OPTIMIZING ROBOT POSITIONING FOR PLASMA CUTTING IS AN INDUSTRIAL APPLICATION.

[2] Domenico Spensieri, Johan S Carlson, Robert Bohlin, Jonas Kressin, and Jane Shi. Optimal robot placement for tasks execution. *Procedia Cirp*, 44:395–400, 2016.

[3] Khalil Zbiss, Amal Kacem, Mario Santillo, and Alireza Mohammadi. Automatic optimal robotic base placement for collaborative industrial robotic car painting. *Applied Sciences*, 14(19):8614, 2024.

[4] Saša Stradovnik and Aleš Hace. Workpiece placement optimization for robot machining based on the evaluation of feasible kinematic directional capabilities. *Applied Sciences*, 14(4):1531, 2024.

[5] George-Christopher Vosniakos and Elias Matsas. Improving feasibility of robotic milling through robot placement optimisation. *Robotics and Computer-Integrated Manufacturing*, 26(5):517–525, 2010.

[6] Huiwen Zhang, Kai Mi, and Zhijun Zhang. Base placement optimization for coverage mobile manipulation tasks. *arXiv preprint arXiv:2304.08246*, 2023.

[7] Manish Saini, Melvin Paul Jacob, Minh Nguyen, and Nico Hochgeschwender. Planning robot placement for object grasping. *arXiv preprint arXiv:2405.16692*, 2024.

[8] W Khalil and E Dombre. Modeling, identification and control of robots hermes. *Penton., Paris et Londre*, 2002.

[9] Wisama Khalil and J Kleinfinger. A new geometric notation for open and closed-loop robots. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1174–1179. IEEE, 1986.

[10] Philippe Wenger and Damien Chablat. A review of cuspidal serial and parallel manipulators. *Journal of Mechanisms and Robotics*, 15(4):040801, 2023.

[11] Durgesh Haribhau Salunkhe, Tobias Marauli, Andreas Müller, Damien Chablat, and Philippe Wenger. Kinematic issues in 6r cuspidal robots, guidelines for path planning and deciding cuspidality. *The International Journal of Robotics Research*, page 02783649241293481, 2024.

[12] Steffan Lloyd, Rishad A Irani, and Mojtaba Ahmadi. Fast and robust inverse kinematics of serial robots using halley's method. *IEEE Transactions on Robotics*, 38(5):2768–2780, 2022.

[13] Hamed Montazer Zohour, Bruno Belzile, and David St-Onge. Kinova gen3-lite manipulator inverse kinematics: optimal polynomial solution. *arXiv preprint arXiv:2102.01217*, 2021.

[14] Lorenzo Sciavicco, Luigi Villani, Bruno SICILIANO, and Giuseppe ORIOLO. *Robotics: modelling, planning and control*. Springer, 2010.

[15] Richard P Paul. *Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators*. Richard Paul, 1981.

[16] Kuo Kan Liang. Efficient conversion from rotating matrix to rotation axis and angle by extending rodrigues' formula. *arXiv preprint arXiv:1810.02999*, 2018.

[17] Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The international journal of Robotics Research*, 4(2):3–9, 1985.

[18] Vladimir J Lumelsky. On fast computation of distance between line segments. *Information Processing Letters*, 21(2):55–61, 1985.

[19] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, pages 39–43. Ieee, 1995.

[20] Abbas El Dor, David Lemoine, Maurice Clerc, Patrick Siarry, Laurent Deroussi, and Michel Gourgand. Dynamic cluster in particle swarm optimization algorithm. *Natural Computing*, 14:655–672, 2015.

[21] James Kennedy and Rui Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1671–1676. IEEE, 2002.

[22] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4):551–559, 1983.

[23] Boris Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin of the Russian Academy of Sciences. Mathematical Series*, (6):793–800, 1934.

[24] Burak Beyhan, Cüneyt Güler, and Hidayet Tağa. An algorithm for maximum inscribed circle based on voronoi diagrams and geometrical properties. *Journal of Geographical Systems*, 22(3):391–418, 2020.