Robotics and Autonomous Systems 200 (2026) 105394

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

# Real-time collision avoidance with robot distance fields in a task-priority framework

Andrea Govoni [a,*], Michela Cavuoto [a], Yiming Li [b], Sylvain Calinon [b], Gianluca Palli [a]

[a] Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna, Via Risorgimento 2, 40136 Bologna, Italy
[b] Idiap Research Institute, Rue Marconi 19, 1920 Martigny, Switzerland

ARTICLE INFO

*Keywords:*
Collision avoidance
Task-priority control
Signed distance fields
Singularity Avoidance
Joint Limit Avoidance

ABSTRACT

Safe and efficient collision avoidance is essential for robots operating in dynamic and cluttered environments. We present a task-priority control framework that embeds signed distance fields (SDFs) directly into the control loop, enabling smooth and reactive avoidance of both environmental and self-collisions. Robot links are represented with Bernstein polynomial-based distance fields, which provide continuous geometry models and closed-form gradients for defining repulsive actions. These avoidance behaviors are activated seamlessly within the task hierarchy and executed in real time through a GPU-accelerated implementation. The framework is validated on fixed-base and mobile manipulators exposed to dynamic obstacles sensed with depth cameras and laser scanners. Results show consistent improvements in responsiveness, computational efficiency and motion smoothness compared to conventional optimization-based approaches, demonstrating the effectiveness of integrating SDFs into task-priority control for robust robot motion in unstructured environments.

## 1. Introduction

In the context of modern robotics, motion planning plays a crucial role as it enables robots to perform complex tasks while ensuring the safety of both the surrounding environment and the robot itself. The primary goal of motion planning is to allow a robot to complete a given task while avoiding problematic situations that could cause damage. This includes obstacle management and safe interaction with the environment, aspects that become particularly relevant in advanced applications such as collaborative robotics or logistics.

A key challenge in motion planning is Collision Avoidance (CA), which refers to the robot's ability to avoid obstacles while executing an assigned path. Initially, CA was tackled through offline trajectory calculations, as [1,2], where the robot's paths were precomputed before task execution. While useful in static environments, these approaches proved to be limiting, as they relied on strong assumptions about the immobility of objects and the computation time for avoiding collisions increased as the task's dimensionality grew. As the demands of robotics have evolved, the need for real-time solutions has led to the development of online CA algorithms. These algorithms are able to operate in dynamic environments, ensuring the safety of the robot as well as the people with whom it interacts. They have become increasingly critical not only for preventing damage to the robot but also for ensuring human safety [3–5].

Motivated by the limitations of existing CA strategies in terms of continuity, modularity and real-time performance, we adopt the Robot Distance Function (RDF) as our core representation. RDF models each robot link independently using Bernstein polynomial (BP)-based Signed Distance Fields (SDFs), providing smooth surface representations and closed-form analytical gradients, interpretable as surface normals. These gradients support precise contact point identification and naturally define repulsive directions for reactive control. Building on this foundation, the main contribution of this article is the integration of RDF into a Task Priority (TP) control framework for redundant robots. The resulting CA behavior is formulated as a secondary inequality task, projected in the null space of the main objective and driven by both distance and gradient information. To broaden applicability, we extend RDF beyond the widely adopted Franka Emika Panda arm to a more diverse set of platforms, including mobile manipulators.

Furthermore, to support operation in dynamic and unstructured environments, we introduce a projection strategy that extends SDF evaluation beyond its original bounded domain, allowing robust distance computation across the full 3D space. In parallel, we address the underexplored problem of self-collision avoidance by developing a scalable, GPU-accelerated architecture that decouples self and environment evaluations—significantly improving computational efficiency and enabling real-time performance.

* Corresponding author.
*E-mail address:* andrea.govoni11@unibo.it (A. Govoni).

https://doi.org/10.1016/j.robot.2026.105394
Received 22 September 2025; Received in revised form 21 January 2026; Accepted 15 February 2026
Available online 20 February 2026
0921-8890/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

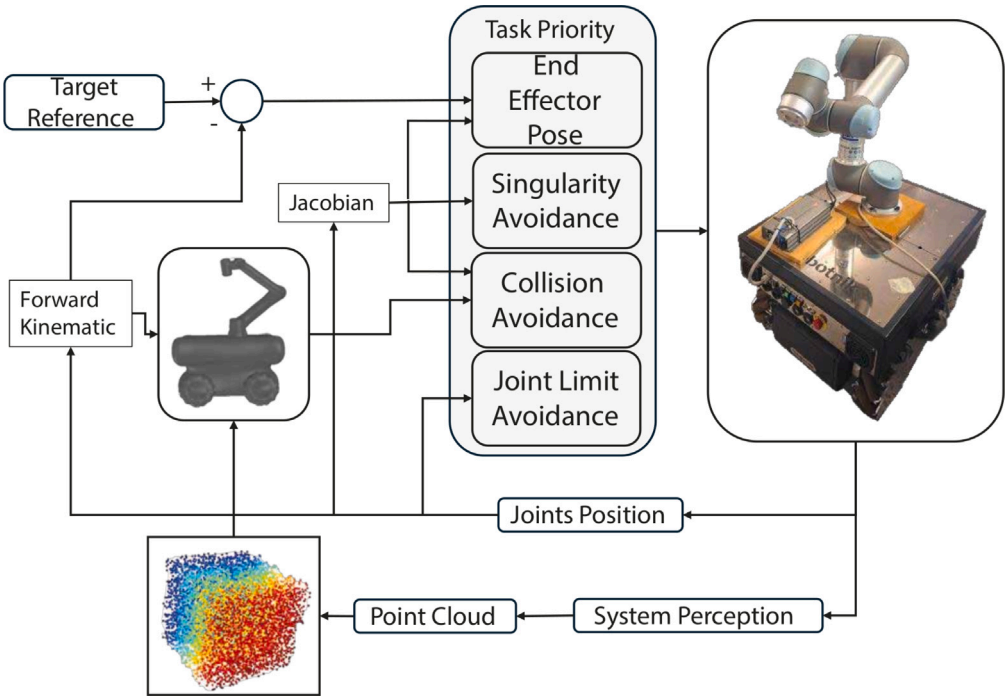



**Fig. 1.** Overview of the proposed Task-Priority framework integrating RDF for collision avoidance.

The article is structured as follows: Section 2 reports a literature review of related works; Section 3 expands RDF formulation and the method to construct and to calculate them; Section 4 introduces the TP framework and discusses how to integrate RDF into TP for obstacle avoidance; Section 5 presents the results obtained in various experiments; finally, Section 6 covers the conclusions.

## 2. Related works

Signed Distance Fields (SDFs) have gained traction in robotics and computer vision for their ability to efficiently represent complex geometries and support fast distance and gradient queries [6]. While traditionally used to model scenes and objects for tasks such as motion planning [7] and grasping [8], recent research has shifted toward representing robot geometry itself using SDFs. This shift enables configuration-space reasoning and facilitates integration with planning and control frameworks [9–11]. Unlike approaches that require precomputing SDFs for each scene or object, robot-centric SDF representations allow direct distance queries from raw sensor data, such as point clouds. Several methods have proposed learning SDFs over joint configurations using neural networks (e.g., Neural-JSDF [12], ReDSDF [3]), but these often neglect the robot's kinematic structure, limiting accuracy and consistency. Other methods (e.g., RDF [9], SDF-SC [13]) instead model each robot link individually and compose the full robot geometry using SE(3) transformations and Boolean operations, which improves accuracy and modularity. Additionally, point-based methods like RMMI [14] and GPU-accelerated SDF computation [15] offer scalability but suffer in dense or geometrically complex scenarios due to the lack of a compact and differentiable structure.

Other online techniques are based on potential fields [16,17] or approximate obstacle shapes [16,18,19], are employed. While computationally efficient, these approaches can be inaccurate in complex environments, potentially resulting in suboptimal paths or collisions. This is particularly problematic for robot self-collision, as the geometry is highly non-convex and configuration-dependent. Recently, Signed Distance Fields (SDFs) have gained popularity for their ability to represent complex shapes compactly and support efficient distance and gradient queries, making them well-suited for obstacle avoidance [20]. In an SDF, each point in the 3D space encodes the minimum distance to the nearest obstacle, with the sign indicating whether it lies inside or outside the object. Recent work has extended this representation to robot manipulators, enabling efficient distance queries between the robot and the environment using raw point cloud data [9]. Numerous algorithms leveraging SDFs have already been proposed, though they are predominantly integrated with quadratic programming techniques [10,12–14,21], but few of them are being explored using TP [4,22].

The goal of CA is to develop online approaches designed to handle scenarios where the environment is dynamic and requires real-time reaction capabilities. These methods, also known as reactive approaches, allow the robot to dynamically adjust its path to avoid collisions, even in the presence of moving obstacles. While many challenges in the literature have been addressed to improve calculations using point cloud and voxel representations, the challenges related to SDFs have mostly been explored using Optimization-Based Methods, particularly QP [9,12–14,23]. However, their integration with Null Space methods has been largely unexplored. The advantage of using the Null Space Method (NSM) is that it offers flexibility and modularity, at the cost of integrating partial Jacobians associated with managing multiple simultaneous control objectives for redundant robots [24]. These methods have been widely studied and applied in various contexts, not only for redundant robotic manipulators but also for mobile robots on land, air and water. However, their application in CA remains limited.

In the context of redundant robotic systems, TP plays a crucial role in managing multiple objectives simultaneously. Redundancy, i.e., the availability of more degrees of freedom than strictly needed for a primary task, enables the execution of secondary tasks without compromising the main objective. This is made possible through NSM, which allows secondary tasks to be projected into the null space of the primary task's Jacobian, ensuring non-interference as long as sufficient redundancy exists. This hierarchical structure is widely adopted in advanced control algorithms [25,26], where secondary objectives such as obstacle avoidance [18,19,27], joint limit or singularity avoidance [28,29] are handled without affecting the execution of high-priority tasks like reaching a target or following a trajectory. Studies such as [27] utilizes the null space of the Jacobian to regulate joint accelerations

and reduce velocity discontinuities during CA. However, their approach does not incorporate a dedicated mechanism for identifying precise collision points on the robot's surface. Accurately localizing such contact points remains a challenging and computationally demanding task. To address this, some methods approximate the robot's geometry using simplified shapes such as spheres or voxels, which ease computation but often sacrifice accuracy. For example, [18] employs large spheres to represent the robot's structure, resulting in poor geometric fidelity. Similarly, [19,30] combines spheres and voxels or points to better approximate surface geometry, but this leads to significantly increased computational cost as the resolution of the representation grows.

SDF-CPF [4] combines SDF with TP but operates without an activation matrix, which introduces certain disadvantages. The lack of an activation matrix within the TP architecture can reduce continuity and smoothness in managing CA, leading to less seamless transitions between tasks, especially in complex environments. Instead of using an activation matrix, they trigger the evasion velocity based on a distance condition, which can cause discontinuities in the velocities applied for evasion.

## 3. Robotic distance functions

This section expands upon the RDF tool introduced in [9], integrating it within the TP framework and providing an overview of its structure.

### 3.1. SDF and RDF

Given a closed surface $\mathcal{S}$, e.g. representing the surface of an object, the SDF provides the signed Euclidean distance of a generic point $p$ to $\mathcal{S}$. The sign indicates whether p lies inside (negative), outside (positive), or exactly on (zero) the surface.

$$f(p) = \begin{cases} d(p,\mathcal{S}) & \text{if } p \text{ is outside of } \mathcal{S}, \\ 0 & \text{if } p \text{ is on the surface } \mathcal{S}, \\ -d(p,\mathcal{S}) & \text{if } p \text{ is inside of } \mathcal{S}, \end{cases} \tag{1}$$

The SDF allows for the description of complex geometric shapes in a continuous and compact manner, but constructing an SDF can be challenging and time-consuming, especially for intricate shapes. In the following, how to create an SDF for each link of the robot in a way that optimizes computation speed will be discussed.

In robotics each link SDF is conveniently represented in its own local frame. Let $T_l^w(q) \in SE(3)$ denote the homogeneous transform of link frame $l$ in the world frame. Then, distance query point $p$ is obtained by transforming the query point from world frame to the link frame:

$$f_l(p,q) = f_l\left((T_l^w(q))^{-1}\, p\right), \tag{2}$$

where $f_l(\cdot)$ denotes the SDF of link $l$ expressed in the link local frame. Henceforth, we omit the explicit dependence on $q$ and $l$ when clear from the context.

As explained in [9], a trade-off between precision and computational efficiency, the SDF can be approximated as a linear combination of $N$ basis functions:

$$f(p) = \Psi^T(p) w \tag{3}$$

where $\Psi(p)$ is the vector of basis functions and $w$ is the weight vector. Generally, we can express the basis functions $\Psi(p)$ as a product of univariate Bernstein bases along each of the three coordinates $x, y, z$ of the point $p$. Specifically:

$$\Psi(p) = \Phi(x) \otimes \Phi(y) \otimes \Phi(z) \tag{4}$$

where $\otimes$ denotes the Kronecker product, $\Phi(\cdot) = [\phi_0(\cdot)\ \cdots\ \phi_{N-1}(\cdot)]$ is the vector of Bernstein polynomials $\phi_n(t) = \binom{N-1}{n} t^n (1-t)^{N-1-n}$ and $0 \le t \le 1$ is the normalized coordinate.

The proposed SDF representation allows adopting variable resolution by changing the number of basis functions. This can be exploited in practical applications involving multiple robot links to have different approximations across the links. On the other side, to enable consistent batch computation on GPU, it is convenient to use always the same number of basis functions. To satisfy both these requirements, the *padding* of the basis functions can be used, consisting in replacing the basis functions over a certain order with zeros up to the maximum number of basis functions required in the problem.

### 3.2. Geometric expansion using ellipsoid

SDF weights $w_i$ in Eq. (3) are generally optimized over a spatially bounded dataset. Specifically, in our setting, all training samples are confined within a sphere of constant radius, see Fig. 2(a). This bounded nature of the dataset directly constrains the learned SDF to operate reliably only within that limited domain. This limitation significantly reduces its applicability in robotic frameworks, where it is often necessary to evaluate large portions of the workspace, including regions outside the training bounds. Filtered the dataset to keep only the object points, i.e., points for which the SDF satisfies SDF<0, we fit an ellipsoid that approximates the spatial extent of the data. Given a point p with coordinates $x, y, z$ in the original coordinate system, its representation in the ellipsoid's local coordinate system is

$$p_e = \Lambda^\top (p - \boldsymbol{\mu}) \tag{5}$$

where $\boldsymbol{\mu} \in \mathbb{R}^3$ is the centroid of the point cloud representing the link, computed as $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} p_i$, where $p_i$ are the points in the dataset. The matrix $\Lambda \in \mathbb{R}^{3\times3}$ contains in its columns the right singular eigenvectors of the point covariance, defining the orientation of the ellipsoid aligned with the principal directions. The semi-axis lengths $a, b, c$ of the ellipsoid are estimated as the maximum absolute values along each local axis: $a = \max|x|, \quad b = \max|y|, \quad c = \max|z|$

Given the ellipsoid implicitly defined by the equation

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1$$

to efficiently project points on the ellipsoid surface, the method introduced by Eberly [31,32] is adopted, which reformulates the problem into minimizing the following function parametrized by the Lagrange multiplier $\lambda$

$$F(\lambda) = \left(\frac{ax}{a^2+\lambda}\right)^2 + \left(\frac{by}{b^2+\lambda}\right)^2 + \left(\frac{cz}{c^2+\lambda}\right)^2 - 1$$

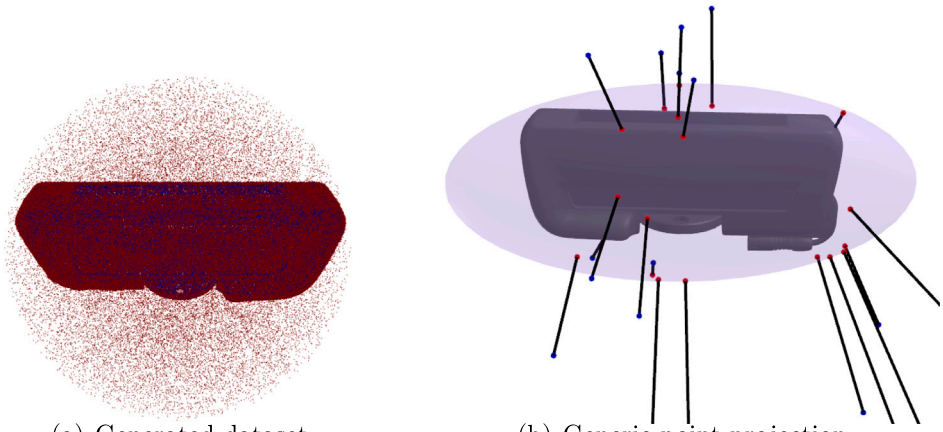
The minimization problem is solved by using the Newton–Raphson method:

$$\lambda_{k+1} = \lambda_k - \frac{F(\lambda_k)}{F'(\lambda_k)}.$$

The algorithm is stopped when $|F(\lambda_k)| < \varepsilon$. The corresponding value of $\lambda_k = \lambda^*$ is used to project the point $p$ on the ellipsoid surface. This projection, namely $p^*$, is expressed in global coordinates as

$$p^* = \Lambda \begin{bmatrix} \frac{a^2 x}{a^2+\lambda^*} \\ \frac{b^2 y}{b^2+\lambda^*} \\ \frac{c^2 z}{c^2+\lambda^*} \end{bmatrix} + \mu$$

The distance between the original point and the point on the ellipsoid surface is computed as $d^* = \|p^* - p\|$. The SDF value at the original point $p$ is then computed by adding this distance to the SDF evaluated at the projected point $p^*$. Hence, in case the point is out of the ellipsoid, the distance function is redefined as $f(p) = f(p^*) + d^*$. The computation is implemented on the GPU to enable parallel processing of large batches of query points.



(a) Generated dataset. (b) Generic point projection.

**Fig. 2.** Dataset sampled around the panda hand and the sphere used for point projection inside dataset training area.

### 3.3. Surface analysis via Bernstein gradients

The distance gradient plays a crucial role in surface analysis, as it indicates the direction of maximum change in distance relative to the surface. In particular, the gradient $\nabla f(\cdot) \in \mathbb{R}^3$ provides a unit normal vector to the object's surface, oriented outward, representing a unit normal vector to the surface of the object. This property makes the gradient not only a descriptor of the local surface geometry, but also an essential tool to determine the direction of potential collisions. Indeed, these vectors can be interpreted as repulsive forces, guiding the robot away from obstacles, whether in case of external objects or self-collisions. A key advantage of the SDF lies in its differentiability, which ensures that its derivatives are continuous and smooth. This smoothness guarantees that distance variations are predictable and free from abrupt changes, enabling robust and reliable gradient-based control strategies.

To compute the gradient of the SDF, we can leverage the derivatives of the basis functions

$$\nabla f(p) = \nabla \Psi(p) w \tag{6}$$

The gradient of the basis function $\nabla\Psi$ can be computed separately with respect to each spatial variable and exploiting Eq. (4) as

$$\begin{aligned} \nabla_x \Psi &= \Phi'(x) \otimes \Phi(y) \otimes \Phi(z), \\ \nabla_y \Psi &= \Phi(x) \otimes \Phi'(y) \otimes \Phi(z), \\ \nabla_z \Psi &= \Phi(x) \otimes \Phi(y) \otimes \Phi'(z) \end{aligned} \tag{7}$$

where the $n$th element of $\Phi'(t)$ is the derivative of the $n$th Bernstein polynomial of order $n$, i.e.

$$\begin{aligned} \nabla_t \phi_n(t) = \binom{N-1}{n} [n\, t^{n-1}(1-t)^{N-n-1} \\ - (N-n-1)\, t^n (1-t)^{N-n-2}] \end{aligned}$$

### 3.4. Projection of a point onto the surface

A key property of the SDF is its ability to determine, for an arbitrary point in space $p$, its orthogonal projection $\hat{p}$ onto the surface implicitly defined by the zero-level set of the SDF, i.e., the set of points satisfying $f(\cdot) = 0$. The projection algorithm is initialized at step 0 with a $p_0 = p$. The projection is then computed iteratively using the SDF gradient $\nabla f(p)$ by using normalized gradient descent:

$$p_{n+1} = p_n - \alpha_n \frac{\nabla f(p_n)}{\|\nabla f(p_n)\|} \tag{8}$$

where $\alpha_n$ is a step size that can be chosen as constant or adaptive. We adopt an adaptive step size $\alpha_n = |f(p_n)|$. In practice, as shown in Fig. 3, exploiting the surface along the normal direction leads to rapid convergence. The algorithm terminates when the absolute value of the distance function falls below a given threshold $\varepsilon > 0$, i.e. $|f(p_n)| < \varepsilon$. Therefore, $\hat{p} = p_n$ is assumed.

## 4. Task priority control

The TP framework is a widely adopted control technique for robotic systems, particularly for complex and redundant platforms. Its core principle is the management of multiple tasks through a hierarchical structure, where each task is assigned a specified priority level. This allows the system to handle concurrent objectives while ensuring that lower-priority tasks do not interfere with higher-priority ones. Moreover, TP allows tasks to be dynamically activated or deactivated to ensure the priority of higher-priority tasks over those already active and ranked lower in the hierarchy. Therefore, in the TP framework, it is possible to distinguish between tasks that are always active, the Equality Tasks (ET), usually representing low-priority tasks and Inequality Tasks (IT), usually representing the high-priority tasks used to ensure robot limits and safety are preserved.

In this section, we focus on four tasks shown in Fig. 1, from lower to higher priority:

- End-effector control (ET) to guide the manipulator gripper toward the desired pose;
- Singularity avoidance (IT) to keep the manipulator away from singular configurations;
- Collision avoidance (IT) from both external and self collisions;
- Joint limit avoidance (IT) to not overcome the robot physical limits.

Note that the only ET is End-effector control, the lowest priority. This task is always active to guide the robot toward its goal, but it can be temporarily overridden when higher-priority tasks become active. In such cases, the end-effector task can still contribute to the system's motion through the available redundancy, as long as it does not conflict with the execution of higher-priority objectives.

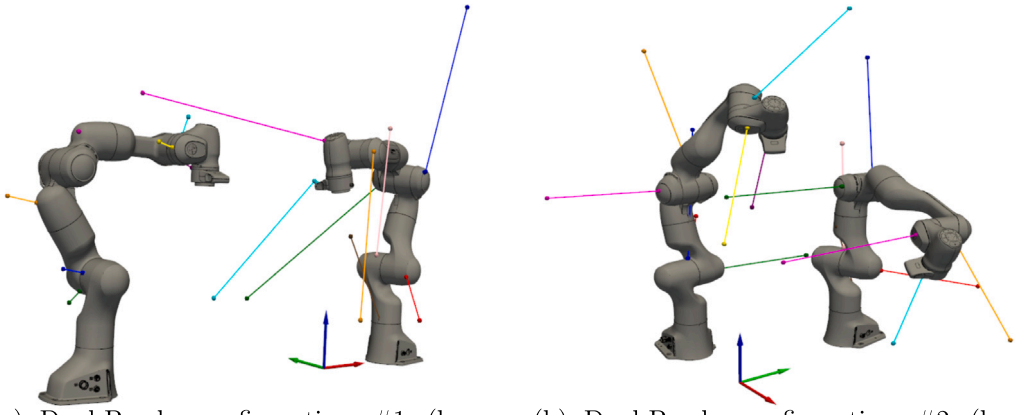
### 4.1. Overview

Let $n$ be the number of degrees of freedom (DoF) of the manipulator and let $m$ be the dimension of the task space, such that $m \leq n$. Let $\mathbf{q} = [q_1, q_2, \dots, q_n]^\top \in \mathbb{R}^n$ be the joint configuration vector of the robotic arm and let $\mathbf{x} \in \mathbb{R}^m$ be the task space vector. Let $\mathbf{J} \in \mathbb{R}^{m \times n}$ be the Jacobian that describes the relationship between the robot's operational space and the space of its actuators. The task differential kinematics is given by:

$$\dot{\mathbf{x}}(\mathbf{t}) = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}(t) \tag{9}$$

The task objective can be followed by generating a control velocity vector $\dot{\mathbf{q}}$, given a reference velocity in task space $\dot{\mathbf{x}}$, through a minimization problem:

$$\min_q \|\dot{\mathbf{x}}(\mathbf{t}) - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}(t)\|^2 \rightarrow \dot{\mathbf{q}}(t) = \mathbf{J}^\dagger(\mathbf{q})\dot{\mathbf{x}} \tag{10}$$



(a) Dual-Panda configuration #1 (base placement and joint configuration).

(b) Dual-Panda configuration #2 (base placement and joint configuration).

**Fig. 3.** Visualization of the projection step used to map sampled points to the SDF zero level set in a dual-Panda setup. For each link, points are initialized randomly in the workspace and updated via gradient descent; colored segments depict the link-local descent direction of a random sampled point. Using the SDF value as an adaptive step size is sufficient for fast convergence in 1–2 iterations.

where $\mathbf{J}^{\dagger}$ denotes the Moore–Penrose pseudo-inverse of the task. In order to fulfill the task control problem, i.e. to steer the task state $\mathbf{x}$ to the desired value $\mathbf{x}^*$, let us define $\widetilde{\mathbf{x}} = \mathbf{x}^* - \mathbf{x}$ as the task error and let $\mathbf{K} \in R^m$ be a positive gain matrix. Therefore, Eq. (10) can be rewritten as:

$$\dot{\mathbf{q}}^* = \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{K}\widetilde{\mathbf{x}} \tag{11}$$

In general, Eq. (11) provides the minimum-norm solution (in terms of $\dot{\mathbf{q}}$). In case $m < n$, infinite solutions are possible and the space of joint velocities can be expressed as:

$$\dot{\mathbf{q}}^* = \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{K}\widetilde{\mathbf{x}} + \mathbf{N}\mathbf{z} \tag{12}$$

where $\mathbf{N} = \mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}$ denotes the null space projector of $\mathbf{J}$ and $\mathbf{z}$ is a generic vector of proper dimension (the null space dimension) that can be used to steer the system to accomplish secondary tasks, i.e. $\mathbf{z} = \dot{\mathbf{q}}_s^*$ can be a secondary task control output obtained from Eq. (11) considering a secondary task objective $\widetilde{\mathbf{x}}_s$.

The core idea behind TP control is to use this recursive definition to create a prioritized task list, in which every task is projected in the null space of the previous higher-priority ones. As mentioned earlier, IT are activated only when the corresponding task state reaches the task domain boundary, to ensure the state remains in its own domain, thus avoiding task domain violation. To this end, Eq. (10) becomes $\min_q \|\mathbf{A}(\dot{\mathbf{x}}(\mathbf{t}) - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}(t))\|^2$ whose solution yields the joint velocity

$$\dot{\mathbf{q}}^* = (\mathbf{A}\mathbf{J}(\mathbf{q}))^{\dagger}\mathbf{A}\mathbf{K}\widetilde{\mathbf{x}} + \mathbf{N}\mathbf{z} \tag{13}$$

where $\mathbf{N} = \mathbf{I} - (\mathbf{A}\mathbf{J})^{\dagger}\mathbf{A}\mathbf{J}$. To ensure smoothness and continuity during task activation, the approach introduced in [26] is adopted. For a smooth task activation and deactivation, avoiding discontinuity of the control input, each $i$th IT is provided with a diagonal activation matrix $\mathbf{A}_i \in \mathbb{R}^{m_i \times m_i}$ whose diagonal entries are defined as a sigmoid function

$$s(x; x_{\text{th}}^-, x_{\text{th}}^+) = \begin{cases} 1, & x \le x_{\text{th}}^-, \\ \frac{1}{2}\left(1 + \cos\left(\pi \frac{x - x_{\text{th}}^-}{x_{\text{th}}^+ - x_{\text{th}}^-}\right)\right), & x_{\text{th}}^- < x < x_{\text{th}}^+, \\ 0, & x \ge x_{\text{th}}^+. \end{cases} \tag{14}$$

where $x_{\text{th}}^-$, $x_{\text{th}}^+$ are suitable threshold parameters chosen to ensure smooth task activation. The switching function in (14) enables a smooth transition, preventing discontinuities in the task weights and the control action. Fig. 4 illustrates the smooth cosine transition, with different value of $x_{\text{th}}^-$ and $x_{\text{th}}^+$. As a trivial generalization, the activation matrix of ET can be considered an identity matrix of proper dimension.

### 4.2. End-effector control

The End-Effector (EE) control is designed to drive the manipulator's end-effector towards a desired pose in Cartesian space, which can be either a fixed target or a time-varying trajectory. It is encoded as an ET and typically assigned lower priority in the task hierarchy to allow redundancy exploitation when higher-priority tasks are active. Given a robotic system, the current end-effector pose $\mathbf{x} \in SO(3)$ (with position $\mathbf{p}_{\mathbf{EE}} \in \mathbb{R}^3$ and orientation $\mathbf{r}_{\mathbf{EE}} \in \mathbb{H}$) is computed via forward kinematics. The desired position $\mathbf{p}^*(t) \in \mathbb{R}^3$ and orientation $\mathbf{r}^*(t) \in \mathbb{H}$ are tracked by computing their respective errors. Starting from the position error, the linear correction velocity is computed as a proportional function of the difference between the desired and current positions: $\mathbf{e}_{\mathbf{p}} = \mathbf{K}_{\mathbf{p}}(\mathbf{p}^*(t) - \mathbf{p}_{\mathbf{EE}(q)})$. Similarly, from the quaternion orientation error, the orientation error is approximated as $\mathbf{e}_{\mathbf{r}} = \frac{2}{\Delta t}\mathbf{E}(\mathbf{r}^*)\mathbf{r}_{\mathbf{EE}}$, where $\Delta t$ is the controller sample time and $\mathbf{E}(\mathbf{r}) \in \mathbb{R}^{3\times 4}$ is a matrix constructed from the unit quaternion stored as a vector $\mathbf{r} = [r_0\ r_1\ r_2\ r_3]^{\top}$ ($r_0$ is the scalar quantity) as

$$\mathbf{E}(\mathbf{r}) = \begin{bmatrix} -r_1 & r_0 & -r_3 & r_2 \\ -r_2 & r_3 & r_0 & -r_1 \\ -r_3 & -r_2 & r_1 & r_0 \end{bmatrix}.$$

In case of tracking a desired trajectory, the velocity reference must also account for the feedforward component $\dot{\mathbf{x}}_{\text{EE}}$ i.e., the desired Cartesian velocity of the trajectory. According to the task-priority approach, the EE control law can be written as:

$$\widetilde{\mathbf{x}}_{\text{EE}} = \dot{\mathbf{x}}_{\text{EE}} + \mathbf{K}_{\text{EE}}\widetilde{\mathbf{e}} \tag{15}$$

with $\widetilde{\mathbf{e}} = [\mathbf{e}_{\mathbf{p}}^{\top}\ \mathbf{e}_{\mathbf{r}}^{\top}]^{\top}$ and $\mathbf{K}_{\text{EE}}$ positive definite gain matrices.

### 4.3. Singularity avoidance

The Singularity Avoidance (SA) task is modeled as an IT aimed at avoiding kinematic singularities. Given the robot EE position Jacobian $\mathbf{J}(\mathbf{q})$, it is possible to decompose it via Singular Value Decomposition (SVD) as $\mathbf{J}(\mathbf{q}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$, where $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values $\sigma_1 > \cdots > \sigma_n$. These values quantify the manipulability of the system along different directions in joint space. The SA task monitors the smallest singular value, which tends to zero near singular configurations, indicating loss of controllability. The associated joint-space direction is the right singular vector corresponding to the smallest singular value, i.e., the last column of $\mathbf{V}$. Accordingly, the SA task Jacobian $\mathbf{J}_{\text{SA}} \in \mathbb{R}^{1\times n}$ is defined as the transpose of this vector.

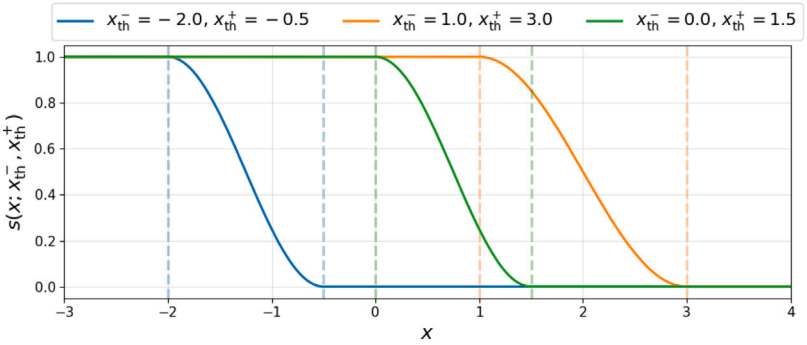



**Fig. 4.** Raised-cosine activation function $s(x; x_{\text{th}}^-, x_{\text{th}}^+)$ used for smooth task activation-deactivation.

To modulate this task near singular configurations, the corresponding activation matrix $\mathbf{A}_{\text{SA}} \in \mathbb{R}^{1\times 1}$ is built with the general formulation in Eq. (14). The activation is driven by the smallest singular value $\sigma_{\text{n}}$, which quantifies the proximity to a singularity. The scalar sigmoid function

$$a_{\text{SA}_{j,j}} = s\left(\sigma_n, \sigma^{\min}, \sigma^{\max}\right) \tag{16}$$

ensures a smooth transition from full activation (when near the singularity) to complete deactivation (when far from it), thereby avoiding discontinuities in the control input.

To regulate the system, a proportional control law drives the singularity toward a desired threshold $\mathbf{x}^*$, defining the task error as:

$$\tilde{x}_{\text{SA}} = K_{\text{SA}}(\mathbf{x}^* - \sigma_n) \tag{17}$$

### 4.4. Joint limits avoidance

The Joint Limit Avoidance (LA) task prevents joints from approaching their mechanical limits by applying a repulsive action as they near predefined thresholds. This is managed through a diagonal activation matrix $\mathbf{A}_{\text{LA}} \in \mathbb{R}^{n\times n}$, where each diagonal entry $a_{j,j}$ represents the activation level for a specific joint j. It is computed as the maximum between two sigmoid-based functions

$$\begin{aligned} a_{LA_{j,j}} &= \max\left(s_j^+, s_j^-\right) \\ s_j^+ &= 1 - s\left(q_j, q_j^+ - \Delta q_j^+, q_j^+\right) \\ s_j^- &= s\left(q_j, q_j^-, q_j^- + \Delta q_j^-\right) \end{aligned} \tag{18}$$

where $q_j^+$ and $q_j^-$ denote the joint's upper and lower limits, respectively $\Delta q_j^+$ and $\Delta q_j^-$ define smooth activation margins near these bounds. Within these regions, the activation value increases gradually, ensuring smooth task engagement and avoiding abrupt control responses.

The task Jacobian is the identity matrix $\mathbf{J}_{\text{LA}} = \mathbf{I}_n$ as each joint is regulated independently.

A proportional control law steers each joint away from its limits. Let $\mathbf{K}_{\text{LA}} \in \mathbb{R}^n$ denote a vector of proportional gains, the task error is computed as:

$$\tilde{\mathbf{x}}_{LA_j} = \begin{cases} K_j\left(q_j^+ - \Delta q_j^+ - q_j\right), & \text{if } q_j > q_j^+ - \Delta q_j^+ \\ K_j\left(q_j^- + \Delta q_j^- - q_j\right), & \text{if } q_j < q_j^- + \Delta q_j^- \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

### 4.5. Obstacle avoidance through robotic distance function

Given a point cloud $\mathcal{P}$ representing both environmental and robot link obstacles in the world coordinate frame, collision avoidance (CA) is formulated as an IT to ensure a minimum safety distance from potential obstacles. To prevent collisions, the aim of the CA task is to generate a repulsive action between the point with the highest collision potential and the corresponding point on the robot link that should be moved away.

To enable computation of the SDFs the point cloud must first be transformed from the world coordinate frame into the local coordinate frame of each robot link. To optimize the process, the point cloud $\mathcal{P}$ is divided in two subsets the environment point cloud $\mathcal{P}^{\text{env}}$, shared among all links and the robot link point cloud $\mathcal{P}^{\text{self}}$, filtered per link using a precomputed collision map. Since the environment point cloud has a fixed size across all links, it supports efficient batched and parallel processing. In contrast, the variable size of the robot link point cloud, determined by the collision map, prevents batching and requires each link to process its points individually.

This separation enables flexible handling of safety margins: Self-Collision Avoidance (SCA) and Environment Collision Avoidance (ECA) can be assigned different safety distances that will be denoted as $d^{\text{self}}$ and $d^{\text{env}}$ respectively. After introducing safety margins for each link, using the definition of SDF, we need to define the most critical collision point $\bar{p}$ to be considered by redefining two sets:

$$\mathcal{P}_{\text{crit}}^{\text{self}} = \{\, p \in \mathcal{P}^{\text{self}} \mid f(p) < d_{\min}^{\text{self}} \,\}, \tag{20a}$$

$$\mathcal{P}_{\text{crit}}^{\text{env}} = \{\, p \in \mathcal{P}^{\text{env}} \mid f(p) < d_{\min}^{\text{env}} \,\}. \tag{20b}$$

The overall set of critical points is then defined as $\mathcal{P}_{\text{crit}} = \mathcal{P}_{\text{crit}}^{\text{self}} \cup \mathcal{P}_{\text{crit}}^{\text{env}}$.

The most critical point can be identified as the point in $\mathcal{P}_{\text{crit}}$ minimizing the distance function:

$$\bar{p} = \arg\min_{p\in\mathcal{P}_{\text{crit}}} f(p) \tag{21}$$

After determining the most critical point $\bar{p}$ the point should be stored together with the associated distance value $f(\bar{p})$ and the surface gradient $\nabla f(\bar{p})$.

These data will then be used in the CA algorithm. Specifically, the critical point is first projected onto the link surface using Eq. (8). Then the two points are transformed back to the world coordinates yielding $p_{\text{robot}}^{\text{world}}$ from the robot side and $p_{\text{coll}}^{\text{world}}$ from the collision object side. Since $\nabla f(\cdot)$ is obtained in the link local frame, a rotation into the world frame is applied using the appropriate link transformation.

The Jacobian at robot critical point is defined by :

$$\mathbf{J}_j(\mathbf{q}) = -\nabla f^{\top}(p_{\text{coll}}^{\text{world}})\mathbf{J}_j^{\text{lin}}(\mathbf{q}) \tag{22}$$

where $\mathbf{J}^{\text{lin}}$ is the linear part of the Jacobian at the collision point. The Jacobian $\mathbf{J}_{\text{CA}}$ of the task will be given by:

$$\mathbf{J}_{\text{CA}}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_1(\mathbf{q}) \\ \vdots \\ \mathbf{J}_j(\mathbf{q}) \\ \vdots \\ \mathbf{J}_n(\mathbf{q}) \end{bmatrix} \tag{23}$$

To avoid discontinuities and to separately assess self- and environment-collision risks for each link, each activation matrix entries

**Algorithm 1** RDF-based collision avoidance pipeline (one control cycle)

**Require:** Link poses $\{T_l^w(q)\}_{l=1..n_L}$; environment cloud $\mathcal{P}_{\text{env}}$; self-collision map $\{\mathcal{P}_{\text{self}}\}$ (per-link); per-link RDF models $\{\text{RDF}_l\}$; CA params $\{d_{\text{env}}^{\min,\max}, d_{\text{self}}^{\min,\max}, d_{\text{safe}}\}$; $K_{\text{CA}}$

**Ensure:** $\widetilde{x}_{\text{CA}}$, $A_{\text{CA}}$, $J_{\text{CA}}$

1: Acquire and build point cloud of environment $\mathcal{P}_{\text{env}}$
2: Build per-link self sets $\{\mathcal{P}_{\text{self}}\}$ from collision map
3: Transform the collision points in link frame: $\mathcal{P}_{\text{env},l} \leftarrow \{(T_l^w)^{-1}\mathcal{P}_{\text{env}}\}_{l=1..n_L}$ and $\mathcal{P}_{\text{self},l} \leftarrow \{(T_l^w)^{-1}\mathcal{P}_{\text{self}}\}_{l=1..n_L}$
4: Apply ellipsoid expansion for out-of-domain query points, if needed
5: Using $\mathcal{P}_{\text{env},l}$ and $\mathcal{P}_{\text{self},l}$ compute RDF
6: Select the most critical point $\bar{p}$
7: Project $\bar{p}$ to robot surface $\hat{p}$ and build collision Jacobian $J_{CA}(q)$
8: Compute CA activation $A_{CA}$
9: Compute CA control law $\widetilde{x}_{\text{CA}}$
10: Solve the hierarchical task-priority controller

are defined as:

$$\begin{aligned} a_{\text{CA}_{j,j}} &= \max\left(s_{j_{\text{env}}}, s_{j_{\text{self}}}\right) \\ s_{j_{\text{env}}} &= s\left(f(\bar{p}_{\text{env}}), d_{j_{\text{env}}}^{\min}, d_{j_{\text{env}}}^{\max}\right) \\ s_{j_{\text{self}}} &= s\left(f_j(\bar{p}_{\text{self}}), d_{j_{\text{self}}}^{\min}, d_{j_{\text{self}}}^{\max}\right) \end{aligned} \tag{24}$$

where $d_{j_{\text{env}}}^{\min}, d_{j_{\text{env}}}^{\max}$ and $d_{j_{\text{self}}}^{\min}, d_{j_{\text{self}}}^{\max}$ are tunable threshold distance parameters.

To prevent the link from reaching a collision state, a barrier function control law is designed.

$$\widetilde{x}_{j_{\text{CA}}} = K_{j_{\text{CA}}} \frac{1}{f_j(p^*) - d_j^{\text{safe}}} \tag{25}$$

where for the $i$th link $d_i$ is the SDF, $d_i^{\text{safe}}$ is the minimum permissible distance.

Alg. 1 is the operational counterpart of the formulation presented in this section. It shows how, at each cycle, the data appearing in the formulation are assembled into the CA task by computing $\bar{p}$, projecting it to $\hat{p}$ and building $J_{\text{CA}}$ and $A_{\text{CA}}$ as defined in the equations, before passing the task to the TP solver.

#### 4.5.1. GPU padding analysis

*Padding of points.* Let $C_{\text{sdf}}(N_{\text{func}})$ denote the point cost of evaluating the distance function and its spatial gradient $f(\mathbf{p})$ from an RDF model with $N_{\text{func}}$ number of functions. Let $\mathcal{A}_{\bullet}$ and $\mathcal{M}_{\bullet}$ denote, respectively, the arithmetic complexity and the GPU-memory footprint.

Let $n_L$ be the number of robot links. As introduced in Eq. (20), we split $\mathcal{P}$ so that let $N_{\text{env}} := |\mathcal{P}_{\text{env}}|$ be the number of environment points provided by the sensor and $N_{\text{self}}^{(j)} := |\mathcal{P}_{\text{self}}^{(j)}|$ the number of self-collision points selected by the collision map of link $j$. Accordingly, for each link $j$ we evaluate $f(\cdot)$ on $N_{\text{env}}$ points and on $N_{\text{self}}^{(j)}$ link-dependent self-collision points.

When enforcing a fully-batched kernel on GPU, variable self-collision sets must be padded to a common size $N_{\max} := \max_j(N_{\text{self}}^{(j)})$, leading to a padded tensor $P_{\text{self}} \in \mathbb{R}^{n_L \times N_{\max} \times 3}$. Consequently, the arithmetic complexity scales as

$$\mathcal{A}_{\text{full-batch}} = \mathcal{O}\big(n_L\,(N_{\text{env}} + N_{\max})\,C_{\text{sdf}}(N_{\text{func}})\big), \tag{26}$$

while the dominant memory footprint scales as

$$\mathcal{M}_{\text{full-batch}} = \Theta\big(n_L\,(N_{\text{env}} + N_{\max})\big). \tag{27}$$

That is, both workload and memory are dictated by the *worst-case* link (largest collision map), even if most links have $N_{\text{self}}^{(j)} \ll N_{\max}$. This is precisely the condition under which padding introduces $(N_{\max} - N_{\text{self}}^{(j)})$ dummy points for each link $j$, increasing both computation and peak memory footprint and potentially triggering the Out Of Memory (OOM).

To mitigate this, we keep environment collisions batched (since $N_{\text{env}}$ is shared across links) and process self-collision serially per link, avoiding global padding:

$$\mathcal{A}_{\text{serial-self}} = \mathcal{O}\big(n_L\,N_{\text{env}}\,C_{\text{sdf}}(N_{\text{func}})\big) + \sum_{j=1}^{n_L} \mathcal{O}\Big(N_{\text{self}}^{(j)}\,C_{\text{sdf}}(N_{\text{func}})\Big), \tag{28}$$

$$\mathcal{M}_{\text{serial-self}} = \Theta\big(n_L\,N_{\text{env}}\big) + \Theta\Big(\max_j N_{\text{self}}^{(j)}\Big), \tag{29}$$

*Padding of weights.* Note that each robot link's SDF can be modeled with a basis of varying resolution, allowing the complexity of the representation to be adapted to the geometry of each link. To make this approach practical on parallel hardware, we assume that the number of basis functions per link does not vary excessively across the robot. Under this condition, the per-link evaluations can be efficiently batched without incurring significant padding overhead. When batching links with different basis sizes, we enforce a common coefficient dimensionality via zero-padding. Let link $i$ be represented with a separable basis of resolution $N_i$ per axis, yielding $N_i^3$ coefficients collected in a weight vector $w_i \in \mathbb{R}^{N_i^3}$. Define the maximum resolution across links as $N_{\text{highest}} := \max_i N_i$. We then embed each $w_i$ into a shared vector space of dimension $N_{\text{highest}}^3$ by appending zeros:

$$\tilde{w}_i := \begin{bmatrix} w_i \\ \mathbf{0}_{N_{\text{highest}}^3 - N_i^3} \end{bmatrix} \in \mathbb{R}^{N_{\text{highest}}^3}, \qquad \forall i, \tag{30}$$

where $\mathbf{0}_k$ denotes a $k$-dimensional zero vector. This padding preserves the original SDF evaluation since the zero-extended components do not affect the result: $\tilde{\Psi}(p)\tilde{w}_i = \Psi(p)w_i + \psi(p)\mathbf{0} = f(p)$ where $\psi(p) \in \mathbb{R}^{N_{\text{highest}}^3 - N^3}$ is the basis extension evaluated at the point of padding.

## 5. Results

The experiments and results are conducted on a system equipped with an Intel Core i7-12700H (12th generation, 2.7 GHz) processor and an NVIDIA GeForce RTX 4060 Ti GPU. It is worth mentioning that, to obtain reliable results, the quality of the input mesh plays a fundamental role. Notably, the dataset proposed in [33] does not robustly handle non-watertight meshes, often leading to poor sampling quality near topological defects such as holes or gaps. To address this issue, in this work all the meshes are preprocessed using the open-source library PyMeshFix [34], which automatically fills holes and repairs topological inconsistencies.

### 5.1. Computational performance

To show the performance of the proposed method, we use the Franka Emika Panda robot as a benchmark, along with a table used as collision object in pick and place actions.

Since [9,13] already provide a comprehensive comparison of real-time SDF-based methods, our evaluation focuses specifically on approaches closely aligned with our proposed TP framework, highlighting the computational benefits introduced by our CA design choices. To better understand the potential advantages of using SDF, we present a comparative analysis, in a static collision environment, of commonly adopted methods for robotic surface approximation and safety distance evaluation. In this evaluation, the robot is guided through multiple configurations and for each pose the minimum distance point is stored and statistic computed. All methods are compared against a reference surface representation, defined as the highest resolution point to point model obtained from high resolution point clouds of the scene. In Table 1 this is explicitly marked as *Ref*. For completeness, we also report lower resolution point to point variants to complete the ablation study, to quantify the accuracy of the proposed method. In Table 1, the column Robot/Table surface approx reports the surface approximation

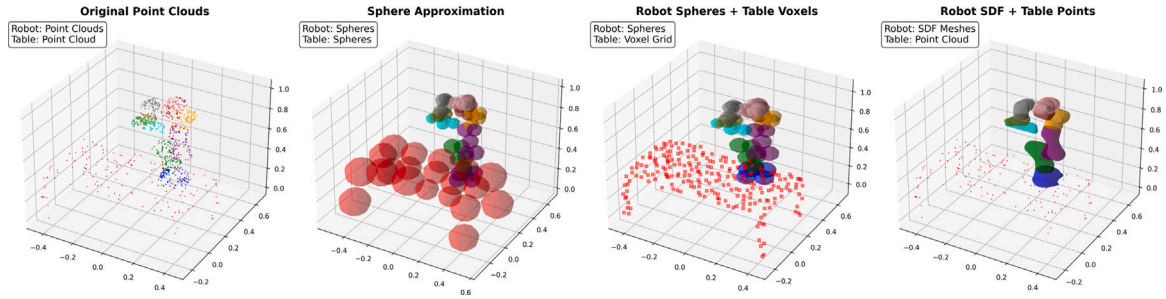
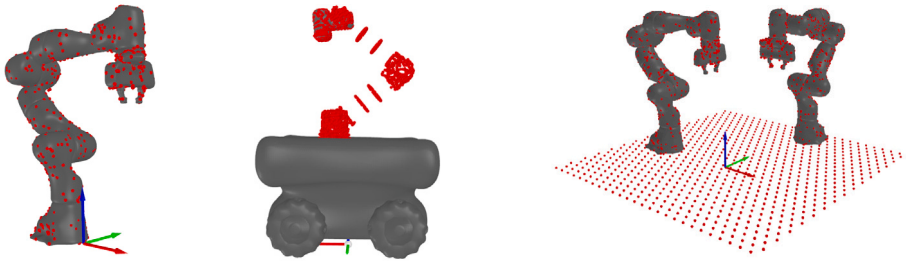



**Fig. 5.** Comparison of surface approximation techniques for collision avoidance: Point-to-Point, Sphere-to-Sphere, Sphere-to-Voxel and RDF-to-Point representations.

**Table 1**
Performance comparison across three resolution levels (Low/Medium/High). "Robot/Table" reports the number of primitives used for robot and table surface approximation. MAE and Std are computed w.r.t. the Point-to-Point baseline with 22500 for the Robot and 1500 for the Table.

| Method | Robot/Table | MAE [mm] | Std [mm] | Time [ms] |
|---|---|---|---|---|
| Point-to-Point | L: 2700/500 (Pts/Pts) | 23 | 15 | 0.9 |
| | M: 9000/1000 (Pts/Pts) | 11 | 12 | 1.8 |
| | H: 22500/1500 (Pts/Pts) | Ref. | Ref. | 10.9 |
| Sphere-to-Point [30] | L: 9/500 (Sph/Pts) | 47 | 30 | 0.4 |
| | M: 45/1000 (Sph/Pts) | 10 | 7 | 1.8 |
| | H: 90/1500 (Sph/Pts) | 6 | 5 | 5.2 |
| Sphere-to-Sphere [18] | L: 9/18 (Sph/Sph) | 105 | 32 | 0.1 |
| | M: 45/36 (Sph/Sph) | 34 | 11 | 1.1 |
| | H: 90/72 (Sph/Sph) | 21 | 1 | 2.9 |
| Sphere-to-Voxel [19] | L: 9/61 (Sph/Vox) | 131 | 39 | 0.3 |
| | M: 45/2284 (Sph/Vox) | 22 | 10 | 2.8 |
| | H: 90/19355 (Sph/Vox) | 19 | 17 | 32.6 |
| Segment-to-Point [23] | L: 9/500 (Seg/Pts) | 57 | 50 | 31.1 |
| | M: 9/1000 (Seg/Pts) | 52 | 43 | 32.0 |
| | H: 9/1500 (Seg/Pts) | 50 | 51 | 32.6 |
| RDF-to-Point [9] | L: 8/500 (NFunc/Pts) | 8 | 2 | 0.5 |
| | M: 12/1000 (NFunc/Pts) | 6 | 1 | 1.1 |
| | H: 16/1500 (NFunc/Pts) | 4 | 1 | 1.9 |



(a) Panda SDF and point cloud representation used in camera scenario.

(b) Mobile base and UR5 SDF and point cloud representation.

(c) Dual Panda arm SDF and point cloud representation and ground plane representation for environment collision.

**Fig. 6.** Three different kinds of systems scenario represented using iso-surface SDF and point cloud. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 2**
Root Mean Square Error (RMSE) and computation time with and without projection, for different numbers of Bernstein Polynomial (BP) basis functions and input points.

| #Func | NumPts | Time [ms] | | RMS [mm] | |
|---|---|---|---|---|---|
| | | Proj | NoProj | Proj | NoProj |
| 8 | 1000 | 3.73 | 0.7 | 7.10 | 8.31 |
| 12 | 1000 | 4.90 | 1.2 | 5.64 | 6.24 |
| 16 | 1000 | 4.73 | 1.79 | 3.3 | 4.9 |

the number of surface *primitives* used to approximate the robot and the table surfaces, respectively. Depending on the method, these primitives are points, voxels, spheres, or segments, as shown in Fig. 5. The results highlight a clear trade off between computational cost and accuracy for discrete methods, such as point-, voxel- and sphere-based representations. These approaches require increasingly fine discretizations to maintain accuracy, which significantly raises computational load. Although accurate at high resolution, such methods scale poorly, limiting their applicability in real-time settings. By contrast, the RDF based approach leverages a continuous implicit surface model, providing stable and accurate distance estimates already at relatively low discretization, resulting in a more favorable accuracy and runtime balance.

#### 5.1.1. Projection on ellipsoid

Table 2 compares the proposed projection method (Section 3.2) against a no-projection baseline. In the *NoProj* case, out of domain queries are handled by forcing the point inside the training domain. This choice keeps the evaluation cheap, but it distorts both the SDF value and its gradient whenever the query lies outside the domain used to fit the weights. In contrast, the proposed *Proj* step maps out of domain queries onto the ellipsoidal boundary, providing a consistent extension of the evaluation to the entire $\mathbb{R}^3$. Quantitatively, projection reduces the RMSE by about 9.6%–32.7% (Table 2) at an additional 2.9–3.7 ms per cycle in this experiment. The additional computation time is modest and remains compatible with real time control provided by sensor; the projection is not primarily meant to reduce RMSE, but to avoid distortions and to extend the method to $\mathbb{R}^3$ without requiring the training dataset to grow unboundedly.

#### 5.1.2. Point-padding

In GPU based collision checking, per link collision maps typically have different cardinalities. As discussed in Section 4.5.1, fully batched GPU kernels often enforce a uniform tensor shape by padding all links to the maximum set size. Consequently, runtime and peak GPU memory are dictated by the worst case link. This padding increases redundant computation and memory traffic and may ultimately trigger OOM failures, if the padding increases a lot. We next quantify this padding overhead and analyze its impact on execution time and memory footprint on a simple example using a UR manipulator mounted on a mobile base (Fig. 6(b)). Note that the wrist links may collide with the four wheels, whereas the four ur links (e.g., `base_link`, → `elbow`) will never interact with them. This construct a simple collision map. Now, let us consider a simple example scenario presented in Table 3 where the environment is set of $N_{\text{env}} = 500$ points and is shared among all links and each wheel is represented by two sampling $N_{\text{wheel}} \in \{50, 500\}$ points and these samples are included only in the self collision maps of the wrist links. In a fully batched implementation, all links are padded to the worst case self-set size $N_{\max} = 4N_{\text{wheel}}$, hence even non interacting links process $4N_{\text{wheel}}$ dummy points. Table 3 reports execution time and peak GPU memory for the *full padding* strategy as in Eqs. (26)–(27) and the *no-padding* strategy as in Eqs. (28)–(29). The results show a clear crossover: for small padding ($N_{\text{wheel}} = 50$), fully batching all links is faster (0.9 ms vs 1.3 ms) because the GPU benefits of a better utilization and reduced per kernel overhead. This experiment also supports the method proposed in Eq. (30), where zero padding enables a single batched evaluation across links instead a per-link evaluation. When the padded cardinality remains close to the per-link sizes, batching is more efficient than iterating over links; for larger padding, the benefit can vanish as dummy point processing dominates. Indeed when $N_{\text{wheel}} = 500$, the additional dummy points inflate both arithmetic work, so the padding overhead dominates and the no-pad strategy becomes faster.

### 5.2. Dynamic collision avoidance with depth camera

This experiment focuses on the application of CA for a 7-DoF Panda robot from Franka Emika, represented in Fig. 7, operating in a dynamic environment. The scene is continuously monitored by a ZED 2i depth camera, which provides real-time 3D perception of both the robot and its surroundings. The camera is mounted in an elevated viewpoint and oriented to maximize the coverage of the robot movement, ensuring continuous visibility of the interaction region throughout the experiment. The camera produces a raw point cloud representing the entire workspace. Since the robot enters in the camera field of view, points belonging to the robot must be removed. We address this by applying a filtering pipeline that removes points corresponding to the robot's structure from the raw point cloud $\mathcal{P}$, using the link meshes $\mathcal{D}$ and their current world transformations $\mathcal{T}_l^w$. The mesh models are first converted into point clouds and transformed into the world frame to reconstruct the robot's shape at each time step. A bounding box is then computed around the resulting cloud to define a region of interest and a *CropBox* filter is applied to remove all scene points within it. To further refine the data, a combination of Statistical and Radius Outlier Removal (SROR) filters is applied to suppress sparse or inconsistent noise.

During testing, we observed that residual artifacts can still appear: robot motion may induce lighting changes and specular reflections that generate spurious points, often projected near or even onto the robot surface. To mitigate this effect, we leverage the robot SDF: points that lie on (or sufficiently close to) the robot's zero level set are classified as self-artifacts and removed. This prevents them from being mistakenly interpreted as external obstacles. As shown in Fig. 9, reflections can produce point-cloud clusters that resemble obstacles. In this example, the operator's hand is correctly detected and triggers collision avoidance at the gripper, whereas the reflected points are suppressed by the iso-surface filtering, avoiding false avoidance behaviors. As a result, only the right link is activated and the robot safely avoids the hand while remaining robust to reflective noise.

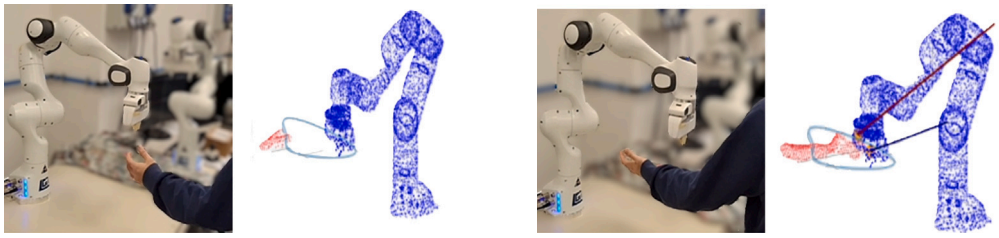
The CA parameters were empirically tuned and in Table 4 we report the tuning parameters used in the tests. To explicitly assess the controller's ability to exploit redundancy under joint limit constraints, we tightened the lower limit of joint 3 by setting $q_3^- = -2.4$ rad and $\Delta q_3^- = 0.3$ rad. The other joints remain constrained within their natural limits, all set with margins $\Delta q^{\pm} = \pm 0.1$ rad.

Fig. 7 illustrates the robot executing the reference trajectory while reacting to a human arm entering the camera field of view. In Fig. 7(a), the operator's hand represented by red points, appears in the scene and is moving to the planned end-effector path, shown as the closed blue curve. At this stage, the CA task is not activated because the hand remains sufficiently far from the robot surface and the corresponding activation matrix stays inactive. As the hand approaches the robot, CA is triggered (Fig. 7(b)): the repulsion gradients become active and drive the robot upward, as indicated by the gradient arrows pointing upward and away from the operator's arm. As shown in Fig. 7(c), the end-effector temporarily deviates from the nominal trajectory, moving upward to increase clearance from the operator's hand under the action of the active repulsion gradients. Finally, in Fig. 7(d), no collision points are detected and the CA task deactivates, so that the end-effector resumes tracking the nominal trajectory.

Similarly, Fig. 8 reports a time sequence of the same scenario, where obstacles enter and leave the camera field of view multiple times, repeatedly triggering CA; the corresponding activation intervals are

**Table 3**
Measured performance for the UR + mobile base example with four wheels. $N_{\text{wheel}}$ is the number of sampled points per wheel (random sampling), hence $N_{\max} = 4N_{\text{wheel}}$ and $N_{\text{env}} = 500$. Time is computed as the average over multiple iterations; Peak Memory refers to the peak GPU memory *allocated* during the forward pass (PyTorch), which is an upper bound on kernel working-set memory.

| $N_{\text{env}}$ | $N_{\text{wheel}}$ | Time [ms] | | Peak memory [GB] | |
|---|---|---|---|---|---|
| | | Pad (Eq. (26)) | No Pad (Eq. (28)) | Pad (Eq. (27)) | No Pad (Eq. (29)) |
| 500 | 50 | 0.9 | 1.3 | 0.23 | 0.17 |
| 500 | 500 | 3.3 | 1.6 | 0.60 | 0.19 |



(a) Robot tracks the reference trajectory; the operator's hand appears in the depth camera; no activation triggered.

(b) As the robot move close to the obstacle, the controller transitions from tracking to avoidance.

(c) Robot deviates from the trajectory moving above the arm to avoid the obstacle.

(d) Hand operator disappears from the scene and just the end effector task remains active.

**Fig. 7.** Collision avoidance scenarios with depth camera (left) and reconstructed point cloud (right). Blue points represent the robot, colored arrows the repulsion gradients, red points the collision object, and the blue line the reference trajectory. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 4**
Task control parameters for the depth-camera experiment.

| Prio | Task | Control params |
|---|---|---|
| | | TP frequency: $f = 300$ Hz, Camera frequency: $f = 20$ Hz |
| 1 | JL | $q^+ = [2.89, 1.76, 2.89, 0.069, 2.897, 3.75, 2.9]$ rad; $q^- = [-2.89, -1.76, -2.4, -2.4, -2.6, -0.017, -2.89]$ rad; joint 3: $\Delta q_3^- = 0.3$ rad; others: $\Delta q_j^{\pm} = 0.1$ rad; $K_{\text{LA}} = 2$ |
| 2 | CA | $d_{\min} = 200$ mm, $d_{\max} = 400$ mm, $d_{\text{safe}} = 50$ mm; $K_{\text{CA}} = 0.3$ Sampling: $N_{\text{camera}}$ = sensor-dependent (depth/ROI dependent) $N_{\text{link}} = 40$ pts/link |
| 3 | SA | $\sigma^{min} = 10^{-3}$, $\sigma^{max} = 10^{-1}$; $K_{\text{SA}} = 0.5$, $x^* = 0.12$ |
| 4 | EE | task always active; $K_{\text{EE}}^{pos} = 2.5$, $K_{\text{EE}}^{ori} = 1$ |

highlighted by gray-shaded regions. Within these intervals, the ECA task is activated as soon as a collision risk is detected, producing repulsive gradients on the distal links (L.6–L.8 in the first case and just L.8 in the second case, since L.6,7 are above the activation threshold). In both cases, the resulting corrective action causes the end effector to deviate smoothly from the nominal path. When the obstacle moves away, the activations decay to zero and the robot progressively returns to the reference trajectory, with the tracking error recovering accordingly.

We also analyze the interaction between the CA module and other secondary tasks. In the SA case, the robot tracks the desired trajectory without entering the singularity activation region; therefore, no noticeable interference with the avoidance behavior is observed. In contrast, the LA task enforces a constraint on joint 3, directly affecting redundancy resolution. In this case, the controller exploits kinematic redundancy to satisfy the joint limit requirement while simultaneously achieving the CA objective.

### 5.3. Omnidirectional mobile manipulator in dynamic environment

This section presents the application of the CA algorithm to a redundant robotic system composed of a UR5 manipulator mounted on a Robotnik Summit XLS omnidirectional mobile base, as shown in Fig. 6(b). The manipulator is constrained to follow a fifth-order polynomial trajectory whose final point lies close to the base, thus creating conditions in which both SCA and ECA can be evaluated. The parameters are set in Table 5. Notably in the joint limit task, the planar base motion is treated as an additional controlled joint vector, appended to the manipulator configuration for controller construction. External obstacles are detected by the base-mounted laser scanners, which generate a 3D point cloud of the surrounding environment, while self-collision risks are computed through forward kinematics by monitoring the relative distances between the manipulator and the

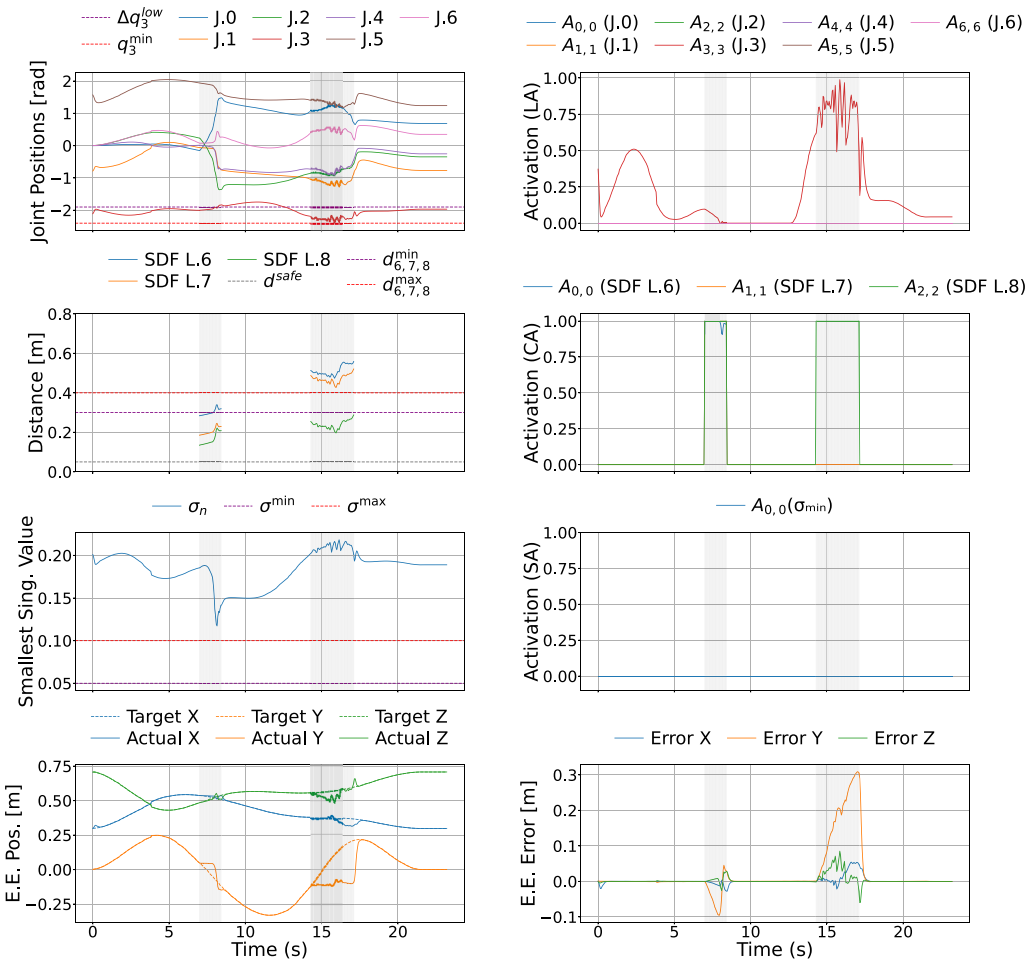
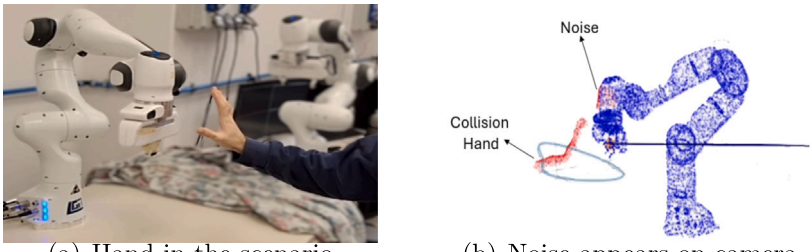



**Fig. 8.** Time evolution of the TP activations in camera task. From top to bottom: joint positions with the joint activations. Distances are computed and CA triggered when an object appears in the camera and exceed the activation limits. Singular value monitor and its activation. Trajectory position and the error given by activation of higher priority tasks.




(a) Hand in the scenario (b) Noise appears on camera

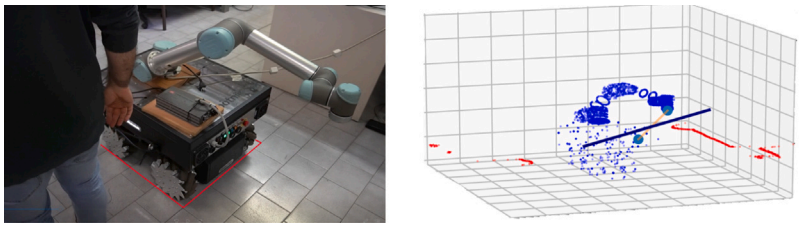
**Fig. 9.** Usage of the SDF for filtering spurious camera on robot surface.

base. We use the Summit XLS onboard laser scanners provided in the standard sensor suite, which continuously monitor the surrounding area around the base.

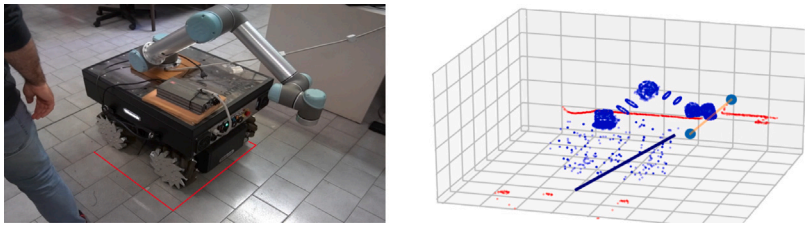
The robot's behavior is illustrated in three representative frames in Fig. 10, where the robot body is shown in blue, its executed trajectory in orange and red points denote obstacles detected by the onboard laser scanners. Colored arrows represent the repulsion gradients applied by the controller. In the initial phase (Fig. 10(a)), as the operator enters the workspace, ECA activates and the mobile base steps back to restore clearance. Later in the motion, the end-effector moves close to the base (Fig. 10(b)), triggering SCA. After the critical distance is restored, SCA disengages and the system resumes ECA-driven avoidance while tracking the reference trajectory, until the end-effector reaches the goal (Fig. 10(c)). In practice, when both hazards are present, task priorities are updated online, switching between ECA and SCA to maintain safe clearances. This behavior is illustrated in Fig. 11 in the distance row, where the task activations show the switching of dominance between ECA and SCA, keeping activation matrix always active to keep the motion in a safe zone.

### 5.4. Dual-arm cross pick-and-place

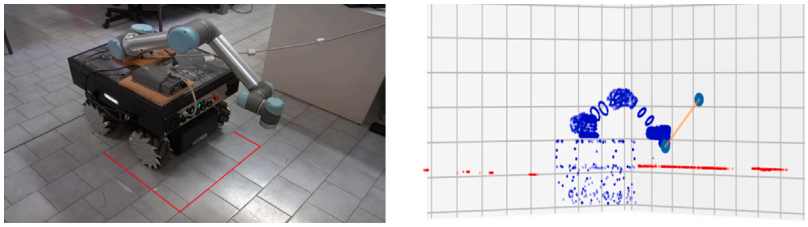
The last scenario involves a dual-arm setup composed of two Franka Emika Panda manipulators, denoted in the graphs of Fig. 13 as Robot A and Robot B (left and right represented as SDF iso-surface in Fig. 6(c), respectively). The task requires the two robots to perform a cross pick-and-place operation, such that each robot must reach into the other's workspace. This naturally creates the need for mutual collision



(a) The operator approaches the robot, triggering Collision Avoidance (CA); the mobile base starts moving.



(b) The operator is still close to the base, but priority shifts to the end-effector; the end-effector approaches the base and triggers SCA.



(c) The base has moved and brings the end-effector to the target without collisions.

**Fig. 10.** Mobile-base collision avoidance with laser scanners and reconstructed scene (right). Blue points denote the mobile robot, red points the laser-scanner returns, the yellow line the reference trajectory (with start and goal markers) and the dark-blue line the repulsion gradient. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 5**
Task control parameters for the UR5 + mobile-base experiment.

| Prio | Task | Control params for UR + Robotnik |
|---|---|---|
| | | TP frequency: $f = 300$ Hz, Laser Scanner frequency: $f = 50$ Hz |
| 1 | JL | UR5 joint limits (default): $q^+ = [3.14, 3.14, 3.14, 3.14, 3.14, 3.14]$ rad; $q^- = [-3.14, -3.14, -3.14, -3.14, -3.14, -3.14]$ rad; $\Delta q_j^{\pm} = 0.1$ rad (all joints); $K_{\text{LA}} = 2$ Mobile-base limits (planar): $\mathbf{q}_b = [x, y, \theta]$ with $x \in [-0.6, 0.5]$ m, $y \in [-0.6, 0.5]$ m, $\theta \in [-3.14, 3.14]$ rad; $\Delta q_b^{\pm} = [0.1, 0.1, 0.1]$; same $K_{\text{LA}} = 2$ |
| 2 | CA | ECA thresholds: $d_{\text{env}}^{\min} = 550$ mm, $d_{\text{env}}^{\max} = 750$ mm; SCA thresholds: $d_{\text{self}}^{\min} = 200$ mm, $d_{\text{self}}^{\max} = 350$ mm; $d_{\text{safe}} = 50$ mm; $K_{\text{CA}} = 0.3$ Sampling: $N_{\text{laser}}$ = laser dependent $N_{\text{link}} = 100$pts/link |
| 3 | SA | $\sigma^{min} = 10^{-3}$, $\sigma^{max} = 10^{-1}$; $K_{\text{SA}} = 0.5$, $x^* = 0.12$ |
| 4 | EE | task always active; $K_{\text{EE}}^{pos} = 1.5$, $K_{\text{EE}}^{ori} = 0.5$ |

**Table 6**
Task control parameters for the dual-Panda experiment.

| Prio | Task | Control params |
|---|---|---|
| | | TP frequency: $f = 400$ Hz |
| 1 | JL | Panda joint limits (default), applied to both robots: $q^+ = [2.89, 1.76, 2.89, 0.069, 2.897, 3.75, 2.9]$ rad; $q^- = [-2.89, -1.76, -2.89, -2.4, -2.6, -0.017, -2.89]$ rad; $\Delta q_j^{\pm} = 0.7$ rad (all joints); $K_{\text{LA}} = 2.5$ |
| 2 | CA | Thresholds (ECA & SCA): $d_{\min} = 0.075$ m, $d_{\max} = 0.11$ m, $d_{\text{safe}} = 0.001$ m; $K_{\text{CA}} = 0.5$ Sampling: $N_{\text{table}} = 300$ pts; $N_{\text{link}} = 60$ pts/link |
| 3 | SA | $\sigma^{min} = 10^{-3}$, $\sigma^{max} = 10^{-1}$; $K_{\text{SA}} = 0.5$, $x^* = 0.12$ |
| 4 | EE | task always active; $K_{\text{EE}}^{pos} = 2$, $K_{\text{EE}}^{ori} = 1$ |

avoidance, as both arms must dynamically adapt their trajectories to prevent interference while completing their pick-and-place actions. Table 6 reports the parameters of the task, to maintain joints as centered as possible, we forced margins to $\Delta q = 0.7$ rad for both upper and lower bounds and the activation distances are uniformed across all links for ECA and SCA both. The number of sampled points on each link surface is increased with respect to Table 4, to better capture close-proximity interactions, where a finer spatial resolution is beneficial.

The evaluation is carried out within a unified TP framework, in which the two manipulators are modeled as a single composite system rather than as independent robots. In this formulation, ECA is restricted to the ground plane represented as red points on the floor in Fig. 6(c), whereas SCA encompasses both intra-robot and inter-robot interactions. In practice, each manipulator link is aware of other links in the chain as a potential collision source but also perceives the geometry of the other manipulator. Particular attention is paid to the distal links, which are

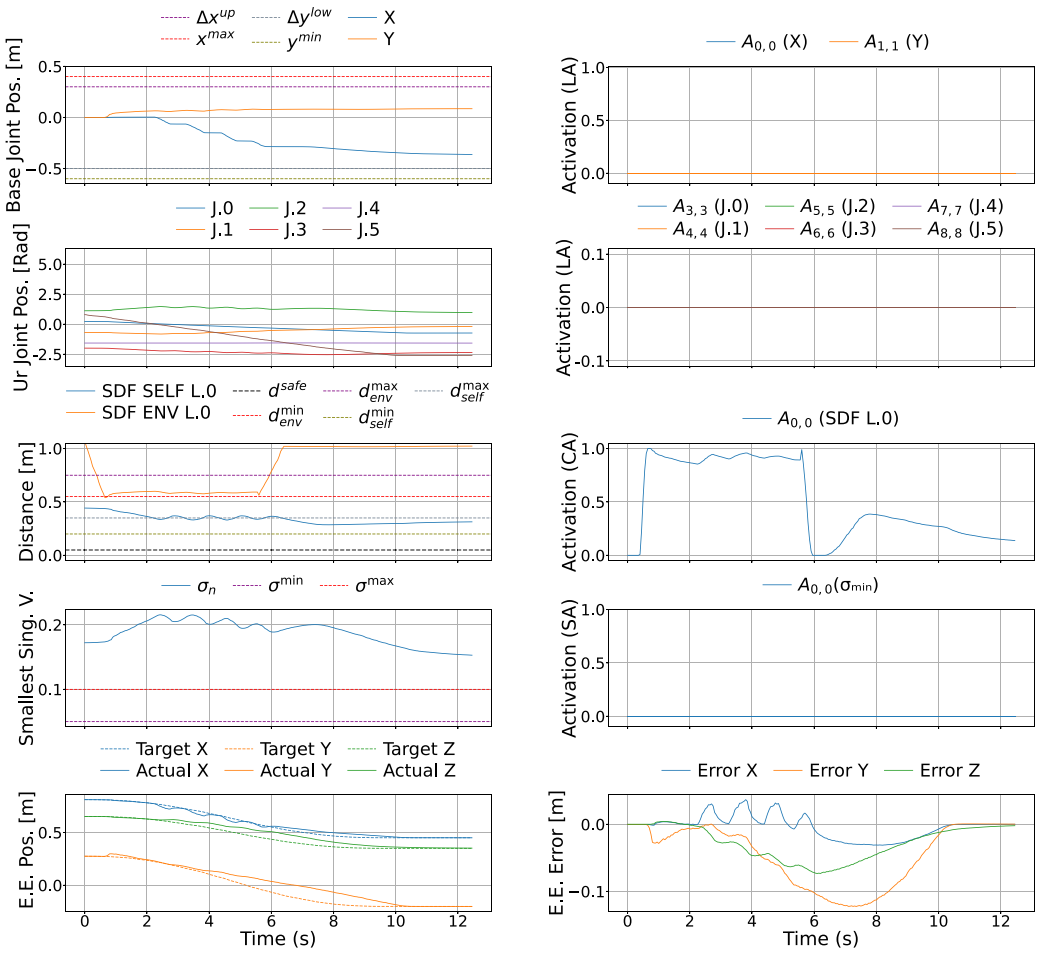



**Fig. 11.** Time evolution of the TP activations in mobile base task with laser scanners. From top to bottom: joint positions with the joint activations. Distances are computed and CA triggered when an object appears near the base. Lowest singular value monitor and its activation. Trajectory position and the error given by activation of higher priority tasks.

the most likely to interfere during the crossing motion. For clarity, in the activation diagrams of Fig. 6(c), link 5 corresponds to the elbow, link 6 to the forearm, link 7 to the wrist and link 8 to the gripper for both robots.

Fig. 12 reports representative frames of the dual-arm execution. We emphasize the first crossing phase: in Fig. 12(b), a potential contact between Robot A's gripper and Robot B's elbow activates CA, generating repulsive gradients that drive Robot B's elbow away from the interaction region. As a result, Robot B adapts its posture while preserving the ongoing motion. During the pick phase (Fig. 12(c)), the activation on Robot B's gripper remains low and the end-effector task stays dominant. In the place phase, only a mild interaction occurs at the grippers, whereas the elbow remains clear thanks to the previously adjusted posture.

Overall, the plots in Fig. 13 together with the execution frames in Fig. 12 confirm that the proposed unified TP framework can successfully manage multi-arm coordination under shared collision constraints. The results highlight not only the alternating dominance between SCA and ECA but also the ability of the system to cope with joint limit activations without compromising task feasibility.

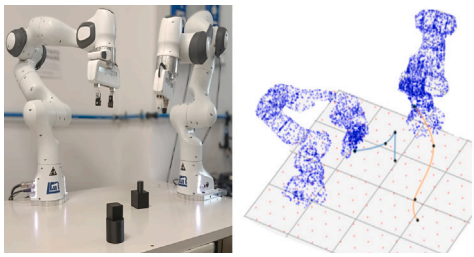
## 6. Conclusion

This work introduced a novel and efficient framework for real-time collision avoidance in redundant robotic systems by integrating RDF within a TP control architecture. The use of SDFs enabled high-fidelity geometric modeling of both the robot and its environment, allowing the system to identify critical contact points and compute smooth repulsive actions through analytical gradients.

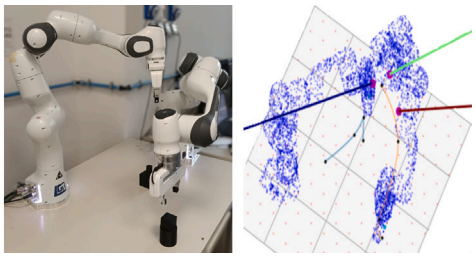
Experimental validation across diverse robotic platforms, ranging from fixed-base arms to mobile manipulators, demonstrated the effectiveness, modularity and computational efficiency of the RDF-TP framework in managing both environmental and self-collisions. The introduction of activation matrices and smooth task transitions proved essential in ensuring stable behavior even in the presence of dynamic obstacles and frequent task switching. Nevertheless, the method can benefit from additional point-cloud filtering and denoising strategies. In this work, we exploited the robot SDF as a preliminary filtering stage to suppress self-artifacts near the robot surface; however, this may be insufficient as the scene complexity increases and the sensing conditions degrade because of noise. Future work will investigate more robust perception pipelines, including multi-view sensing and sensor-fusion mechanisms, to mitigate depth-camera jitter and improve robustness under partial visibility and sensing degradation.

Furthermore, the GPU-accelerated implementation confirmed the method's suitability for high-frequency control applications, maintaining low latency while supporting dense point cloud processing. Compared to existing optimizations and voxel-grid approaches, the proposed method offers a superior trade-off between geometric accuracy, responsiveness and real-time performance.

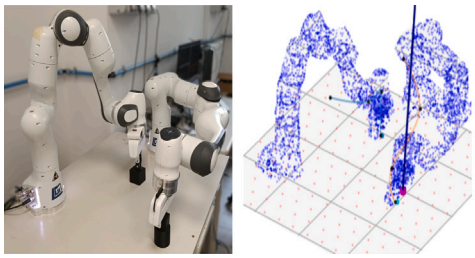
The modularity of the architecture makes it highly adaptable to various robotic platforms and compatible with existing motion planning pipelines. Future work will focus on extending the framework with data-driven strategies for automatic tuning of control parameters



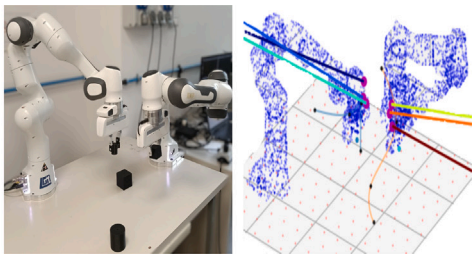
(a) Initial configuration.



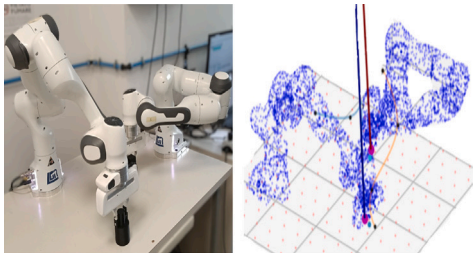
(b) Collision triggered between the left gripper and the right elbow; both arms are pushed apart.



(c) Because of elbow CA the right arm exploits redundancy and it reaches the grasp pose with a different elbow posture



(d) Trajectories cross at close distance; CA is activated, moving a little the end effector from nominal trajectory.



(e) Task completed.

**Fig. 12.** Dual-arm cross pick-and-place task highlights. Blue points denote the Robot A on the left and Robot B on the right. Red points denotes the environment collision point cloud. The colored line denoted the repulsion gradients. the repulsion gradient. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and on integrating predictive models to enhance responsiveness and anticipation in dynamic environments.

Despite the effectiveness of the proposed approach, the current controller still relies on manual tuning of key task parameters (e.g. activation thresholds). This limitation motivates future extensions toward AI-driven and sim-to-real strategies to automatically calibrate these parameters, improving robustness and transferability across tasks and deployment conditions.

## CRediT authorship contribution statement

**Andrea Govoni:** Writing – original draft, Validation, Software. **Michela Cavuoto:** Writing – original draft, Software. **Yiming Li:** Writing – review & editing, Conceptualization. **Sylvain Calinon:** Writing – review & editing, Supervision. **Gianluca Palli:** Writing – review & editing, Writing – original draft, Supervision.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Andrea Govoni reports equipment, drugs, or supplies was provided by Intelliman (EU Horizon Europe Project, grant number 101070136). Andrea Govoni reports a relationship with University of Bologna that includes: non-financial support. The authors declare that they have no other activities or relationships that could have influenced the submitted work. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This research did not receive specific funding. However, this work is supported by the Horizon Europe project IntelliMan – AI-Powered Manipulation System for Advanced Robotic Service, Manufacturing and Prosthetics [grant number 101070136].

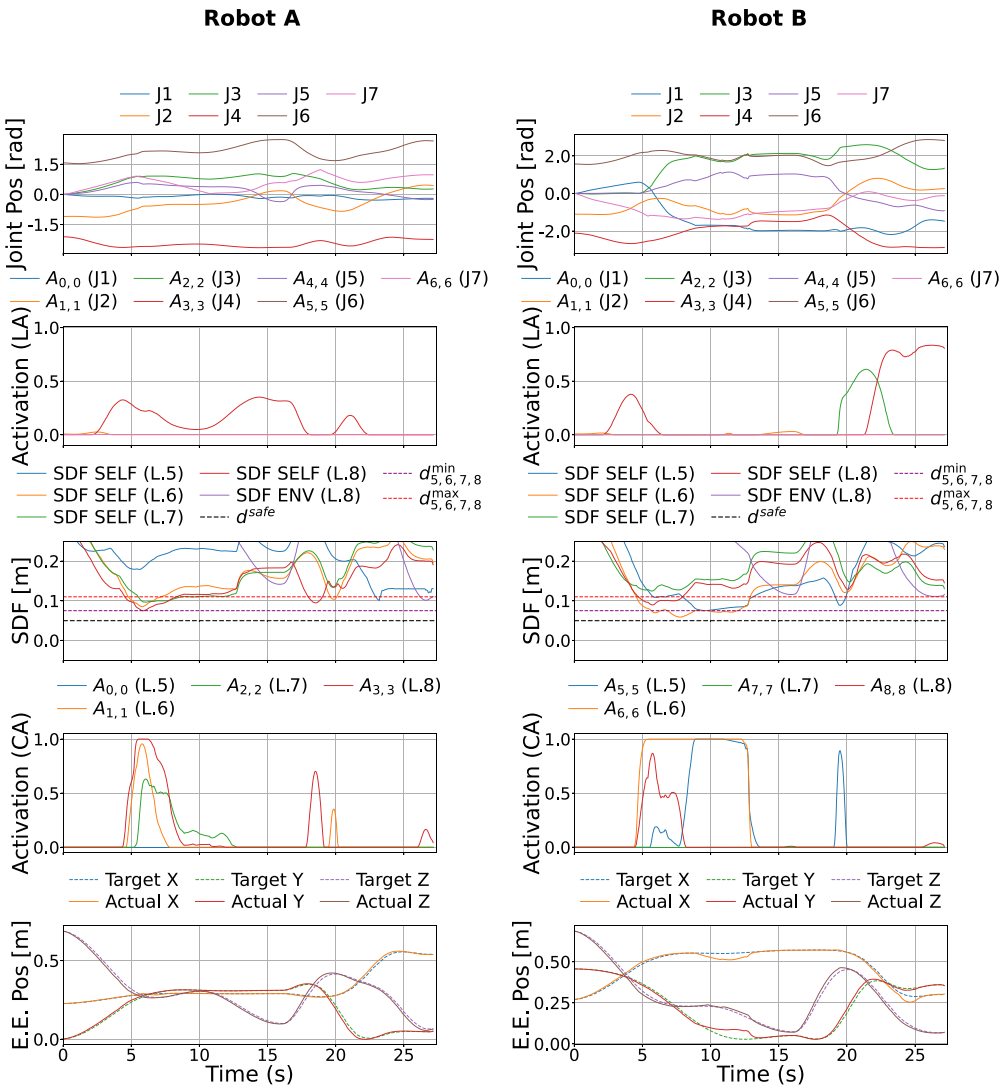



**Fig. 13.** Time evolution of TP activations during the dual-arm pick-and-place task. The left column reports Robot A, and the right column Robot B. From top to bottom: joint positions together with the corresponding activation levels; inter-robot/environment distances used to trigger CA when the robots approach each other or the surroundings; and the end-effector trajectory tracking (reference and executed) in the last row.

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2026.105394.

## Data availability

The data that has been used is confidential.

## References


[1] J. Kuffner, S. LaValle, Rrt-connect: An efficient approach to single-query path planning, in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), vol. 2, 2000, pp. 995–1001, http://dx.doi.org/10.1109/ROBOT.2000.844730.

[2] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, S. Teller, Anytime motion planning using the rrt*, in: 2011 IEEE International Conference on Robotics and Automation, 2011, http://dx.doi.org/10.1109/ICRA.2011.5980479.

[3] P. Liu, K. Zhang, D. Tateo, S. Jauhri, J. Peters, G. Chalvatzaki, Regularized deep signed distance fields for reactive motion generation, 2022, arXiv:2203.04739, URL https://arxiv.org/abs/2203.04739.

[4] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, in: Proceedings. 1985 IEEE International Conference on Robotics and Automation, 1985, http://dx.doi.org/10.1109/ROBOT.1985.1087247.

[5] G. Du, Y. Liang, G. Yao, C. Li, R.J. Murat, H. Yuan, Active collision avoidance for human-manipulator safety, IEEE Access (2022) 16518–16529, http://dx.doi.org/10.1109/ACCESS.2020.2979878.

[6] J.J. Park, P. Florence, J. Straub, R. Newcombe, S. Lovegrove, Deepsdf: Learning continuous signed distance functions for shape representation, 2019, pp. 165–174.

[7] D. Driess, J.-S. Ha, M. Toussaint, R. Tedrake, Learning models as functionals of signed-distance fields for manipulation planning, 2021, arXiv:2110.00792, URL https://arxiv.org/abs/2110.00792.

[8] T. Weng, D. Held, F. Meier, M. Mukadam, Neural grasp distance fields for robot manipulation, 2023, arXiv:2211.02647. URL https://arxiv.org/abs/2211.02647.

[9] Y. Li, Y. Zhang, A. Razmjoo, S. Calinon, Representing robot geometry as distance fields: Applications to whole-body manipulation, in: 2024 IEEE International Conference on Robotics and Automation, ICRA, 2024, pp. 15351–15357, http://dx.doi.org/10.1109/ICRA57147.2024.10611674.

[10] K. Long, H. Parwana, G. Fainekos, B. Hoxha, H. Okamoto, N. Atanasov, Neural configuration distance function for continuum robot control, 2025, arXiv:2409.13865. URL https://arxiv.org/abs/2409.13865.

[11] Y. Li, X. Chi, A. Razmjoo, S. Calinon, Configuration space distance fields for manipulation planning, in: Proc. Robotics: Science and Systems, RSS, 2024.

[12] M. Koptev, N. Figueroa, A. Billard, Neural joint space implicit signed distance functions for reactive robot manipulator control, IEEE Robot. Autom. Lett. (2022) http://dx.doi.org/10.1109/LRA.2022.3227860.

[13] X. Zhu, Y. Xin, S. Li, H. Liu, C. Xia, B. Liang, Efficient collision detection framework for enhancing collision-free robot motion, 2024, arXiv:2409.14955, URL https://arxiv.org/abs/2409.14955.

[14] N. Marticorena, T. Fischer, J. Haviland, N. Suenderhauf, Rmmi: Enhanced obstacle avoidance for reactive mobile manipulation using an implicit neural map, 2024, arXiv:2408.16206. URL https://arxiv.org/abs/2408.16206.

[15] V. Vasilopoulos, S. Garg, P. Piacenza, J. Huh, V. Isler, Ramp: Hierarchical reactive motion planning for manipulation tasks using implicit signed distance functions, in: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2023, pp. 10551–10558.

[16] K.B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, D.A. Anisi, Collision avoidance with potential fields based on parallel processing of 3d-point cloud data on the gpu, in: 2014 IEEE International Conference on Robotics and Automation, ICRA, 2014, pp. 3250–3257, http://dx.doi.org/10.1109/ICRA.2014.6907326.

[17] K. Katona, H.A. Neamah, P. Korondi, Obstacle avoidance and path planning methods for autonomous navigation of mobile robot, Sensors (2024) http://dx.doi.org/10.3390/s24113573, https://www.mdpi.com/1424-8220/24/11/3573.

[18] P. Cieślak, R. Simoni, P. Ridao Rodríguez, D. Youakim, Practical formulation of obstacle avoidance in the task-priority framework for use in robotic inspection and intervention scenarios, Robot. Auton. Syst. (2020) http://dx.doi.org/10.1016/j.robot.2019.103396, https://www.sciencedirect.com/science/article/pii/S0921889019306104.

[19] W.B. Bedada, G. Palli, Real time collision avoidance with gpu-computed distance maps, 2024, arXiv:2407.02363. URL https://arxiv.org/abs/2407.02363.

[20] N. Ratliff, M. Zucker, J.A. Bagnell, S. Srinivasa, Chomp: Gradient Optimization Techniques for Efficient Motion Planning, IEEE, 2009, pp. 489–494.

[21] Y. Li, X. Chi, A. Razmjoo, S. Calinon, Configuration space distance fields for manipulation planning, 2024, arXiv:2406.01137. URL https://arxiv.org/abs/2406.01137.

[22] S. Xu, G. Li, J. Liu, Obstacle avoidance for manipulator with arbitrary arm shape using signed distance function, in: 2018 IEEE International Conference on Robotics and Biomimetics, ROBIO, 2018, http://dx.doi.org/10.1109/ROBIO.2018.8665083.

[23] Y. Wu, X. Jia, T. Li, J. Liu, A real-time collision avoidance method for redundant dual-arm robots in an open operational environment, Robot. Comput.-Integr. Manuf. 92 (2025) 102894, http://dx.doi.org/10.1016/j.rcim.2024.102894, https://www.sciencedirect.com/science/article/pii/S0736584524001819.

[24] Y. Nakamura, Advanced robotics: Redundancy and optimization, in: Addison-Wesley series in electrical and computer engineering: Control engineering, Addison-Wesley Publishing Company, 1991, https://books.google.it/books?id=hp4QAQAAMAAJ.

[25] J. Cacace, F. Ruggiero, V. Lippiello, Hierarchical task-priority control for human–robot co-manipulation, 2020, pp. 125–138, http://dx.doi.org/10.1007/978-3-030-42026-0_10.

[26] E. Simetti, G. Casalino, A novel practical technique to integrate inequality control objectives and task transitions in priority based control, J. Intell. Robot. Syst. (2016) http://dx.doi.org/10.1007/s10846-016-0368-6.

[27] M. Khatib, K. Al Khudir, A. De Luca, Task priority matrix at the acceleration level: Collision avoidance under relaxed constraints, IEEE Robot. Autom. Lett. (2020) http://dx.doi.org/10.1109/LRA.2020.3004771.

[28] L. Huo, L. Baron, The joint-limits and singularity avoidance in robotic welding, Ind. Robot.: An Int. J. (2008) http://dx.doi.org/10.1108/01439910810893626.

[29] B.J. Nelson, P.K. Khosla, Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limit avoidance, Int. J. Robot. Res. (1995) 255–269, http://dx.doi.org/10.1177/027836499501400304.

[30] J. Alonso-Mora, T. Naegeli, R. Siegwart, P. Beardsley, Collision avoidance for aerial vehicles in multi-agent scenarios, Auton. Robots 39 (1) (2015) 101–121, http://dx.doi.org/10.1007/s10514-015-9429-0.

[31] D. Eberly, Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid, 2020, https://www.geometrictools.com/Documentation/DistancePointEllipseEllipsoidHyperellipsoid.pdf, geometric Tools, Version 1.0.

[32] D. Eberly, Perspective projection of an ellipsoid, 1999, https://www.geometrictools.com/Documentation/PerspectiveProjectionEllipsoid.pdf, last revised 12, 2013, URL https://www.geometrictools.com/Documentation/PerspectiveProjectionEllipsoid.pdf.

[33] J.J. Park, P. Florence, J. Straub, R. Newcombe, S. Lovegrove, Deepsdf: Learning continuous signed distance functions for shape representation, 2019, arXiv:1901.05103, URL https://arxiv.org/abs/1901.05103.

[34] M. Attene, A lightweight approach to repairing digitized polygon meshes, Vis. Comput. (2010) http://dx.doi.org/10.1007/s00371-010-0416-3.