2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Approximated Collision Detection for Contact-Rich Dexterous Manipulation With Nonnegative Least Squares

Weibing Li[1], Jiajun Luo[1], Lei Yang[1], Yehui Li[1] and Kai Huang[1]

***Abstract*— Collision detection between robotic hands and manipulated objects is crucial to model predictive control (MPC) for contact-rich dexterous manipulation. Based on the Gilbert-Johnson-Keerthi (GJK) algorithm and the expanding polytope algorithm (EPA), the GJK-EPA method has achieved success while requiring iterative optimizations. Recently, a signed distance function (SDF) based collision detection (C-SDF) method is used to estimate the contact information, which avoids iterations at the cost of matrix derivative operations. Inspired by this, in this paper, a simplified nonnegative least squares (NNLS) based quadratic programming (QP) algorithm is used to construct an approximated solution to the QP formulation of collision detection, for estimating collision points. Then, contact distances and Jacobians are calculated via physics computations and differentiable kinematics. Consequently, a C-NNLS method is proposed, which uses NNLS formulation to approximate the collision detection routine in the MPC while avoiding iterative optimizations and matrix derivatives. The C-NNLS method is applied to extensive simulative tasks, achieving lower average error while consuming 45.59% less time on average compared with the C-SDF method. Furthermore, the C-NNLS method is deployed on a real Allegro hand for on-palm reorientation. Results show that the C-NNLS method reduces average task time by 30.33% compared with the C-SDF method while maintaining high-quality dexterous manipulation.**

I. Introduction

Collision detection routine is one of the core modules in dexterous manipulation, which is mainly responsible for computing the contact distance and contact Jacobian. With multiple contact states frequently switching between the manipulated object and the robot hand (i.e., attachment and detachment), an accurate and efficient collision detection method becomes essential.

Numerous algorithms have been proposed to handle collision detection and distance computation for two convex shapes, such as the Gilbert-Johnson-Keerthi algorithm (GJK) [1], the V-Clip algorithm [2] and the Lin-Canny algorithm [3]. Among these methods, the GJK algorithm is the most classic. Many algorithms have been extended based on the GJK algorithm, e.g., the expanding polytope algorithm (EPA), which is proposed to compute the penetration depth [4]. Based on the GJK algorithm and EPA, the GJK-EPA method has been proven to be efficient in MuJoCo [5]. However, the iteration number increases significantly when dealing with smooth strictly convex objects due to its discrete nature, posing challenges for dexterous manipulation [6]. Recently, a signed distance function (SDF) based collision detection (C-SDF) method is used to approximate the collision detection routine in the model predictive control (MPC) for dexterous manipulation [7]. By approximating distances and employing a gradient-descent-like technique on predefined query points, the C-SDF method estimates collision points, which eliminates iterative optimizations but requires matrix derivatives.

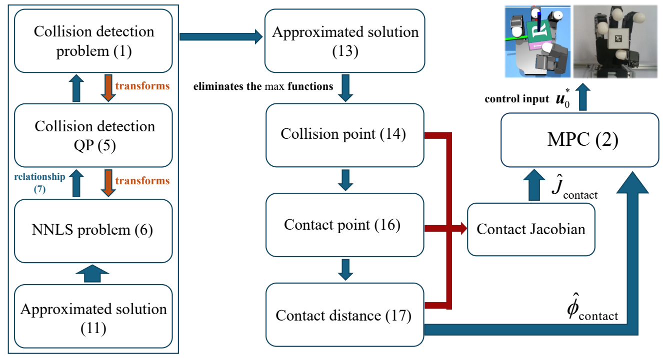


Fig. 1: The flow chart of the C-NNLS method.

This paper aims to propose a novel method to approximate the collision detection routine in contact-rich dexterous manipulation. The proposed method seeks to bridge the performance gap between the approximated method and the classic GJK-EPA method, and even surpass the GJK-EPA method in some tasks. Specifically, this paper proposes a C-NNLS method to approximate the collision detection routine by utilizing a simplified nonnegative least squares (NNLS) based quadratic programming (QP) algorithm [8]. In particular, the NNLS method is introduced to obtain an approximated solution to the QP of the collision detection optimization. With hyperparameters and projection techniques used, an explicit approximated solution to the NNLS formulation is constructed. By leveraging the relationship between the solution to the NNLS formulation and the QP, the approximated solution to the target QP is transformed into an explicit form. Then, the resultant explicit form is used to estimate the collision points, which can be used to compute the contact information. After that, the proposed C-NNLS method is integrated into an MPC framework for dexterous manipulation. For better understanding, the flow chart of the C-NNLS method is

*This work was supported in part by the Guangxi Key Research and Development Program under Grant GuikeAB25069495, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2025A1515011485, and in part by the Guangzhou Basic and Applied Basic Research Foundation under Grant 2025A04J5280. (Corresponding author: Weibing Li.)

[1]Weibing Li, Jiajun Luo, Lei Yang, Yehui Li and Kai Huang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China (e-mail: liwb53@mail.sysu.edu.cn; luojj68@mail2.sysu.edu.cn; yanglei39@mail.sysu.edu.cn; liyh699@mail.sysu.edu.cn; huangk36@mail.sysu.edu.cn)

presented in Fig. 1. The MPC framework with the C-NNLS method is evaluated through extensive real-time dexterous manipulation tasks. Moreover, the C-NNLS method is applied to a real Allegro hand for the cube reorientation tasks in the real world, demonstrating the effectiveness and feasibility of the C-NNLS method. The contributions can be summarized as follows.

1) A C-NNLS method is proposed to approximate the collision detection routine in the MPC framework for dexterous manipulation.
2) The C-NNLS method avoids iterative optimizations and matrix derivatives compared to existing methods.
3) The C-NNLS method is developed for dexterous manipulation, achieving advanced results: (i) for simulated tasks, achieving lower average error while consuming 45.59% less time on average, and (ii) for real-world experiments, reducing average task time by 30.33% while maintaining high-quality dexterous manipulation.

II. Related Work

In this section, collision detection methods including the GJK-EPA method and the C-SDF method are first introduced. Then, details of the NNLS method are described.

A. Collision Detection Methods

For collision detection between two convex objects, the GJK algorithm is a classic approach based on the Minkowski operations of two polytopes [1], [9]. It can compute collision and distance queries for numerous different convex polyhedra including cylinders, ellipsoids and spheres [10]. Due to its versatility and efficiency, the GJK algorithm has become one of the most popular collision detection algorithms, and several improvements have been proposed and implemented based on the GJK algorithm. On one hand, its sub-operations have been enhanced by replacing the original Johnson algorithm and backup procedure with a faster distance subalgorithm [11]. On the other hand, the number of iterations in the GJK algorithm has been decreased to accelerate the process [12], [13]. As an extension of the GJK algorithm, the EPA method is proposed to calculate the depth of penetration between two convex objects [4], [14]. Most of these algorithms are implemented in the widely used Flexible Collision Library (FCL) [15], and are applied in many physics engines. For example, the GJK-EPA method serves as the collision detection core and performs well in MuJoCo [5]. However, its iteration number increases significantly when dealing with smooth and convex shapes [6]. Recently, the C-SDF approximated method is proposed, which avoids iterative optimizations [7]. But this method needs matrix derivatives while computing the collision points by using a gradient-descent-like technique.

B. Nonnegative Least Squares Method

The NNLS method has wide applications including data classification and genomic analysis [16], [17]. In particular, an NNLS-based QP algorithm is proposed [8]. In this NNLS method, a strictly convex QP is transformed into an NNLS form, and then the explicit relationship between the solution to the NNLS form and the target QP is leveraged. By solving the NNLS problem, the solution to its QP is obtained. In this paper, the NNLS method is used to obtain an approximated solution to the QP for collision detection. Specifically, hyperparameters and projection techniques are employed to construct an approximated solution to the NNLS form, thereby deriving an approximated alternative to collision detection. Thus, the C-NNLS method is proposed. The C-NNLS method first obtains approximated collision points and then directly calculates the distance between the collision points and the query points. Unlike the C-SDF method [7], the C-NNLS method does not require matrix derivatives.

III. Preliminaries

In this section, a collision detection problem is first introduced, and then an MPC for dexterous manipulation is detailed.

A. Collision Detection

First, let us consider the collision detection problem under the following assumptions:

1) The manipulated object is rigid.
2) The geometry of the object is convex, which can be represented by a set of supporting planes $\mathcal{A} = \left\{\boldsymbol{x} \mid \boldsymbol{n}_i^\mathsf{T}\boldsymbol{x} + b_i \leq 0,\ i = 1, 2, ..., \mathrm{N}\right\}$. Each supporting plane is parameterized by its unit normal vector $\boldsymbol{n}_i \in \mathbb{R}^3$ and offset $b_i \in \mathbb{R}$.
3) The geometry of the object is given, which can be obtained from depth data integration [18], or learned from raw sensor data [19].

For convenience, several notations are defined. The matrix $\tilde{\boldsymbol{\mathcal{N}}} := [\boldsymbol{n}_1 \quad \boldsymbol{n}_2 \quad \ldots \quad \boldsymbol{n}_\mathrm{N}] \in \mathbb{R}^{3\times\mathrm{N}}$ denotes the horizontal concatenation of vectors $\boldsymbol{n}_i$ and vector $\tilde{\boldsymbol{b}} := [b_1; b_2; \ldots; b_\mathrm{N}] \in \mathbb{R}^\mathrm{N}$ indicates the vertical stacking of scalars b_i.

For collision detection, an object-centric perspective is usually applied to a dexterous manipulation system [7]. For each contact point, the collision detection routine is to compute the contact distance and the contact Jacobian between a query point $\boldsymbol{x}_\text{query} \in \mathbb{R}^3$ and the manipulated object $\mathcal{A}$, i.e.:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \left\|\boldsymbol{x} - \boldsymbol{x}_\text{query}\right\|_2 \\ \text{s.t.} \quad & \tilde{\boldsymbol{\mathcal{N}}}^\mathsf{T}\boldsymbol{x} + \tilde{\boldsymbol{b}} \leq \boldsymbol{0}, \end{aligned} \tag{1}$$

where the query point can be obtained from sampling the surfaces of a robotic hand or an environment.

B. MPC for Dexterous Manipulation

In an MPC for dexterous manipulation, the MPC can be formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{u}_{0:T\text{-}1} \in [\mathbf{u}_\text{lb}, \mathbf{u}_\text{ub}]} \quad & \sum\nolimits_{t=0}^{T-1} C_t\left(\boldsymbol{q}_t, \boldsymbol{u}_t\right) + C_T\left(\boldsymbol{q}_T\right) \\ \text{subject to} \quad & \boldsymbol{q}_{t+1} = \boldsymbol{q}_t \oplus h\boldsymbol{v}_t, \\ & \boldsymbol{v}_t = f\left(\tilde{\boldsymbol{\phi}}_0, \tilde{\boldsymbol{J}}_0, \boldsymbol{u}_t, \boldsymbol{\theta}\right), \\ & \tilde{\boldsymbol{\phi}}_0, \tilde{\boldsymbol{J}}_0 \text{ are obtained by solving (1)}, \\ & \text{given } \boldsymbol{q}_0, \quad t = 0, 1, \ldots, T \end{aligned} \tag{2}$$

where h and $\boldsymbol{u}$ denote the timestep and the control input, respectively. The first constraint updates the system state $\boldsymbol{q}_t$ with the system velocity $\boldsymbol{v}_t$, which is the time-stepping prediction routine. The functioin $f(\cdot)$ is a contact model used to calculate the system velocity $\boldsymbol{v}_t$ with contact distance $\tilde{\boldsymbol{\phi}}_0$, contact Jacobian $\tilde{\boldsymbol{J}}_0$ and parameter $\boldsymbol{\theta}$. Note that the contact information $\{\tilde{\boldsymbol{\phi}}_0, \tilde{\boldsymbol{J}}_0\}$ is fixed in the horizon T for reducing the computational complexity [7], [20].

In the dexterous manipulation system, the path and cost functions in the MPC can be defined as

$$\begin{aligned} C_t\left(\boldsymbol{q}_t, \boldsymbol{u}_t\right) &:= a_{\mathrm{c}} C_{\text{contact}}(\boldsymbol{q}) + a_{\mathrm{u}} \left\|\boldsymbol{u}\right\|^2, \\ C_T\left(\boldsymbol{q}_T\right) &:= a_{\mathrm{p}} \left\|\mathbf{p}^{\text{obj}} - \mathbf{p}_{\text{goal}}\right\|^2 + a_{\mathrm{q}}\left(1 - (\mathbf{q}_{\text{goal}}^{\mathsf{T}} \mathbf{q}^{\text{obj}})^2\right) \end{aligned} \tag{3}$$

with

$$C_{\text{contact}}(\boldsymbol{q}) := \sum_{i=1}^{n_{\text{fg}}} \left\|\mathbf{p}^{\text{obj}} - \mathbf{p}^{\text{fg}_i}\right\|^2, \tag{4}$$

where $(\mathbf{p}^{\text{obj}}, \mathbf{q}^{\text{obj}})$ and $(\mathbf{p}^{\text{goal}}, \mathbf{q}^{\text{goal}})$ denote the actual pose and the goal pose of the object, respectively. Meanwhile, $\mathbf{p}^{\text{fg}_i}$ represents the fingertip position of the finger i in the robotic hand. $\{a_{\mathrm{c}}, a_{\mathrm{u}}, a_{\mathrm{p}}, a_{\mathrm{q}}, n_{\text{fg}}\} \in \mathbb{R}$ are specific parameters where n_{fg} is the amount of fingers and others denote the weights.

Therefore, the final cost C_T represents the distance between the object pose and the goal pose. The path cost C_t consists of contact cost C_{contact} and control cost $\|\boldsymbol{u}\|^2$, where contact cost C_{contact} encourages the fingertips keeping contact with the object. Additionally, in this paper, the MPC (2) is solved using CasADi [21] with the IPOPT solver [22].

IV. NNLS-Based Collision Detection

In this section, the C-NNLS method is proposed. Then, details of the NNLS method for approximating the collision detection in dexterous manipulation are introduced.

A. NNLS-Based Solution for Collision Detection

The QP depicted in (1) for collision detection can be transformed into the following QP:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{x} - \boldsymbol{x}_{\text{query}}^{\mathsf{T}}\boldsymbol{x} \\ \text{s.t.} \quad & \tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{x} + \tilde{\boldsymbol{b}} \leq \mathbf{0}. \end{aligned} \tag{5}$$

Both the objective function of (1) and (5) have the same unique global minimizer, thus, it suffices to solve (5).

According to [8], the solution to QP (5) can be obtained by solving an NNLS problem, which is formulated as follows:

$$\min_{\boldsymbol{y} \geq \mathbf{0}} \quad \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{p}\right\|_2^2, \tag{6}$$

with $\boldsymbol{A} := \begin{bmatrix} -\boldsymbol{M}^{\mathsf{T}} \\ -\boldsymbol{d}^{\mathsf{T}} \end{bmatrix} \in \mathbb{R}^{4\times \mathrm{N}}$ and $\boldsymbol{p} := \begin{bmatrix} \mathbf{0} \\ \gamma \end{bmatrix} \in \mathbb{R}^4$ where $\boldsymbol{M} := \tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{L}^{-1} \in \mathbb{R}^{\mathrm{N}\times 3}, \boldsymbol{d} := -\tilde{\boldsymbol{b}} - \tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{x}_{\text{query}} \in \mathbb{R}^{\mathrm{N}}$, $\gamma \in \mathbb{R}^+$ and $\boldsymbol{L}$ is a Cholesky factor of the Hessian matrix. Additionally, the optimal solution to (6) is represented by $\boldsymbol{y}^*$. Then, let $\boldsymbol{r}^* := \boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{p}$, when $\boldsymbol{r}^* \neq \mathbf{0}$, it follows Theorem 1 in [8] that the optimal solution $\boldsymbol{x}^*$ to (5) can be recovered by

$$\boldsymbol{x}^* = \boldsymbol{x}_{\text{query}} - \frac{\tilde{\boldsymbol{\mathcal{N}}}\boldsymbol{y}^*}{\gamma + \boldsymbol{d}^{\mathsf{T}}\boldsymbol{y}^*}. \tag{7}$$

From the above relation, it can be found that problem (5) can be solved by solving problem (6). However, solving problem (6) exactly is still computationally demanding. In order to better meet the real-time requirements, the traditional optimization algorithm is not used to directly solve (6) to obtain $\boldsymbol{y}^*$. Instead, a hyperparameterized solution is adopted to obtain $\hat{\boldsymbol{y}}^*$ as an approximatd alternative of $\boldsymbol{y}^*$.

B. Approximated Solution for NNLS Reformulation

Considering that matrix $\boldsymbol{A}$ in (6) is not necessarily of full column rank, to obtain an approximated solution, let us add a regularization term to the objective function:

$$\boldsymbol{\mathcal{F}}\left(\boldsymbol{y}\right) := \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{p}\right\|_2^2 + \frac{\kappa}{2}\left\|\boldsymbol{y}\right\|_2^2, \tag{8}$$

where parameter κ is a small constant.

From the first-order optimality criteria of a quadratic objective, the solution to $\min\{\boldsymbol{\mathcal{F}}\left(\boldsymbol{y}\right)\}$ can be given by

$$\hat{\boldsymbol{y}}^* = (\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A} + \kappa\boldsymbol{I})^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{p}. \tag{9}$$

Moreover, to simplify the computation, it is assumed that the inverse of the matrix $(\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A} + \kappa\boldsymbol{I})^{-1}$ in (9) can be approximated by a matrix $\boldsymbol{K}_{\text{cd}} := c\boldsymbol{I}$ with $c \in \mathbb{R}^+$, i.e.,

$$\hat{\boldsymbol{y}}^* = \boldsymbol{K}_{\text{cd}}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{p}. \tag{10}$$

Recalling that $\boldsymbol{y} \geq \mathbf{0}$ in (6), the approximation $\hat{\boldsymbol{y}}^*$ in (10) does not necessarily satisfy this condition. Thus, let us perform a simple projection:

$$\hat{\boldsymbol{y}}^* = \max\left(\boldsymbol{K}_{\text{cd}}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{p}, \mathbf{0}\right). \tag{11}$$

Remark 1: The approach in (11) involves using a hyperparameterized $\hat{\boldsymbol{y}}^*$ to approximate the optimal solution $\boldsymbol{y}^*$, followed by performing a simple projection to satisfy the non-negativity constraint. This method does not yield the optimal solution $\boldsymbol{y}^*$ and is relative to the choice of $\boldsymbol{K}_{\text{cd}}$.

By substituting $\boldsymbol{A}^{\mathsf{T}}$ and $\boldsymbol{p}$, an approximated solution to NNLS formulation (6) can be obtained:

$$\hat{\boldsymbol{y}}^* = \max\left(-\gamma\boldsymbol{K}_{\text{cd}}\boldsymbol{d}, \mathbf{0}\right). \tag{12}$$

Then, the relation (7) is utilized to construct the approximated solution to (5):

$$\hat{\boldsymbol{x}}^* = \boldsymbol{x}_{\text{query}} - \frac{\tilde{\boldsymbol{\mathcal{N}}}\max\left(-\gamma\boldsymbol{K}_{\text{cd}}\boldsymbol{d}, \mathbf{0}\right)}{\gamma + \boldsymbol{d}^{\mathsf{T}}\max\left(-\gamma\boldsymbol{K}_{\text{cd}}\boldsymbol{d}, \mathbf{0}\right)}. \tag{13}$$

In fact, the non-zero terms in the $\max$ functions of (13) can be proven to be greater than zero. Noting that $\gamma > 0$ and matrix $\boldsymbol{K}_{\text{cd}}$ is defined as a positive multiple of the unit matrix, the sign of the vector $\boldsymbol{d}$ needs considering only. If $\boldsymbol{d} < 0$ is proved, it directly follows that the non-zero terms are greater than zero in (13). Obviously, the vector $\boldsymbol{d}$ is defined as $-\tilde{\boldsymbol{b}} - \tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{x}_{\text{query}}$, which is equivalent to $-(\tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{x}_{\text{query}} + \tilde{\boldsymbol{b}})$. Since the query point $\boldsymbol{x}_{\text{query}}$ does not belong to the manipulated object's point cloud, $(\tilde{\boldsymbol{\mathcal{N}}}^{\mathsf{T}}\boldsymbol{x}_{\text{query}} + \tilde{\boldsymbol{b}}) > 0$ holds, which implies $\boldsymbol{d} < 0$. Therefore, it can be assumed that the non-zero term is greater than zero, leading to the elimination of the $\max$ functions in (13):

$$\hat{\boldsymbol{x}}^* = \boldsymbol{x}_{\text{query}} + \frac{\tilde{\boldsymbol{\mathcal{N}}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}}{1 - \boldsymbol{d}^{\mathsf{T}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}}. \tag{14}$$

Remark 2: The approximated collision point is computed directly in (14). Compared to the C-SDF method [7], the C-NNLS method avoids matrix derivatives and effectively cuts down computational costs, which can be demonstrated in the subsequent results.

Note that (14) provides an approximated solution to collision detection (5). With the query point $\boldsymbol{x}_{\text{query}}$ on the robot subtracted by the collision point $\hat{\boldsymbol{x}}^*$ on the object, the distance between them can be obtained:

$$\text{dist}(\hat{\boldsymbol{x}}^*, \boldsymbol{x}_{\text{query}}) = \left\| \frac{\tilde{\boldsymbol{\mathcal{N}}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}}{1 - \boldsymbol{d}^{\mathsf{T}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}} \right\|_2. \tag{15}$$

Generally, the approximated contact point $\hat{\boldsymbol{x}}_{\text{contact}}$ is defined as the midpoint between the collision point $\hat{\boldsymbol{x}}^*$ and the query point $\boldsymbol{x}_{\text{query}}$:

$$\hat{\boldsymbol{x}}_{\text{contact}} = \boldsymbol{x}_{\text{query}} + \frac{\tilde{\boldsymbol{\mathcal{N}}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}}{2\left(1 - \boldsymbol{d}^{\mathsf{T}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}\right)}. \tag{16}$$

Then, the approximated contact distance $\hat{\phi}_{\text{contact}}$ is defined as half of the distance $\text{dist}(\hat{\boldsymbol{x}}^*, \boldsymbol{x}_{\text{query}})$, that is:

$$\hat{\phi}_{\text{contact}} = \frac{1}{2} \left\| \frac{\tilde{\boldsymbol{\mathcal{N}}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}}{1 - \boldsymbol{d}^{\mathsf{T}} \boldsymbol{K}_{\text{cd}} \boldsymbol{d}} \right\|_2. \tag{17}$$

With the above approximated contact point $\hat{\boldsymbol{x}}_{\text{contact}}$ and its contact distance $\hat{\phi}_{\text{contact}}$, the contact Jacobian $\hat{\boldsymbol{J}}_{\text{contact}}$ required in (2) can be computed using differential kinematics [23].

V. Simulative Validation

In this section, the C-NNLS method is applied to on-palm z-axis reorientation tasks for evaluating its performance.

A. Simulative Setup

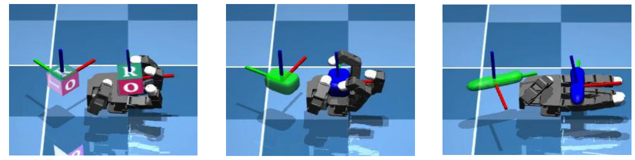
1) Simulation Implementation: The simulative robot involves an Allegro hand with 16 degrees-of-freedom (DoFs), which are actuated by PD controllers with proportional gain $k_{\text{p}} = 1$ and damping gain $k_{\text{d}} = 0.05$. As shown in Fig. 2, the task is to rotate the object on the palm of the Allegro hand from an initial orientation to a target orientation, including three different objects from the ContactDB dataset [24]. The target orientation $\mathbf{q}_{\text{goal}}$ is given in TABLE I, which consists of clockwise (CW) rotations and counter-clockwise (CCW) rotations.

TABLE I: Goals in regular z-axis rotation

Cube	Foambrick	Stick
$\{\pm\frac{\pi}{4}, \pm\frac{\pi}{3}, \pm\frac{\pi}{2}\}$	$\{\pm\frac{\pi}{6}, \pm\frac{\pi}{3}, \pm\frac{\pi}{2}\}$	$\{\pm 0.2\pi, \pm 0.25\pi, \pm 0.4\pi\}$

For three different objects, each has six regular target orientations, resulting in a total of eighteen regular trials.

For the MPC setup, the relevant parameters of the simulative validation are listed in TABLE II. In addition, the prediction horizon $T = 4$, and the control bound $\{\mathbf{u}_{\text{lb}}, \mathbf{u}_{\text{ub}}\}$ is set to $\{-0.15, 0.15\}$ for cube object while it is set to $\{-0.1, 0.1\}$ for foambrick and stick objects.



(a) Cube (b) Foambrick (c) Stick

Fig. 2: Examples of the initial scene of the regular z-axis rotation task, where the left object shows the goal orientation.

TABLE II: MPC setting for simulative validation

Object	C_t			C_T		H
	n_{fg}	a_{c}	a_{u}	a_{p}	a_{q}	
Cube	4	0.5	0.1	1000	10000	150
Foambrick	4	1	0.2	500	5000	200
Stick	4	0.5	0.2	1000	10000	500

MPC setting for each object includes parameters of cost function $\{n_{\text{fg}}, a_{\text{c}}, a_{\text{u}}, a_{\text{p}}, a_{\text{q}}\}$ and the maximum rollout length H.

2) Metric: To evaluate the performance of the collision detection method, several metrics are considered in the simulative validation, including I) the success rate S_{rate}; II) **for successful trials**, average orientation error E_{ori} within the last 18 rollout steps; III) the task duration D_{task}. Further, a trial succeeds if the success condition $E_{\text{ori}} \leq 0.04$ is satisfied consecutively for 18 rollout steps within the maximum rollout length H, otherwise fails. Follows [20], the orientation error E_{ori} is defined as

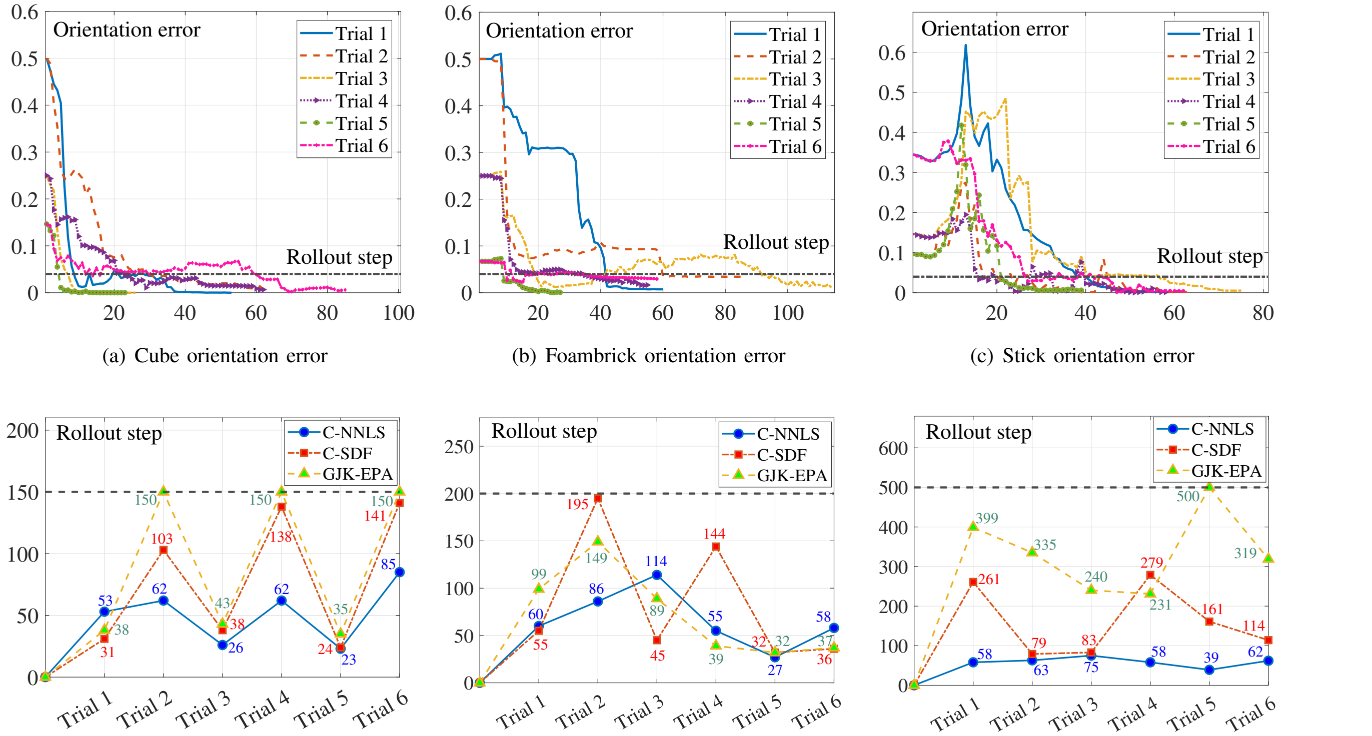
$$E_{\text{ori}} := 1 - (\mathbf{q}_{\text{goal}}^{\mathsf{T}} \mathbf{q}^{\text{obj}})^2, \tag{18}$$

where $\mathbf{q}_{\text{goal}}$ and $\mathbf{q}^{\text{obj}}$ denote the goal orientation of the object and current orientation of the object, respectively. Then, the task duration D_{task} is represented by the number of rollout steps in all trials. Because the number of steps is not influenced by the hardware state but depends only on the algorithm, and fewer rollout steps usually mean a shorter task duration.

3) Baseline: To better analyze the performance of the C-NNLS method, the C-SDF method [7] and the collision detection algorithm of the MuJoCo [5] are chosen to be the baseline approaches. Note that the collision detection in MuJoCo employs the GJK-EPA method.

B. Results and Analysis

Fig. 3 (a)-(c) show the orientation errors for three objects using the C-NNLS method and (d)-(f) list the detailed task duration for each trial using different collision detection methods. TABLE III shows the simulative results of the regular z-axis rotation task. Both the C-NNLS and C-SDF methods complete all tasks successfully, while the GJE-EPA method fails in some trials. For successful trials, the C-NNLS method achieves the lowest average error, reducing it by



(a) Cube orientation error (b) Foambrick orientation error (c) Stick orientation error

(d) Cube task duration (e) Foambrick task duration (f) Stick task duration

Fig. 3: Simulative results of the regular z-axis rotation task, where (a)-(c) show orientation errors using the C-NNLS method, and (d)-(f) present task durations for each object's six trials using different methods.

TABLE III: Comparison in regular z-axis rotation

Object	**Success rate S_{rate} [%]**			**Orientation error E_{ori} [$\times 10^{-3}$]**			**Task duration D_{task} [steps]**		
	C-NNLS	C-SDF [7]	GJK-EPA [5]	C-NNLS	C-SDF [7]	GJK-EPA [5]	C-NNLS	C-SDF [7]	GJK-EPA [5]
Cube	100	100	50	3.15	2.38	1.57	311	475	566
Foambrick	100	100	100	16.5	14.5	12.2	400	507	445
Stick	100	100	83.3	4.66	8.62	20.4	355	977	2024
Average	**100**	**100**	77.8	**8.10**	8.50	11.4	**355.3**	653	1011.7

The orientation error E_{ori} is computed at the last 18 steps of MPC rollout in success trials. For each failed trial, the task duration is the maximum length H.

approximately 28.95% and 4.71% compared to the GJK-EPA and C-SDF methods, respectively. For task duration D_{task}, TABLE III records the total steps for each object across six different target orientations. Given that the GJK-EPA method has failed trials, the focus is on comparing the average steps used by other methods. It shows that the C-NNLS method uses 297.7 fewer steps on average than the C-SDF method, reducing task time by about 45.59%. Specifically, for the stick, which tends to roll during manipulation, the advantages of the C-NNLS method are more prominent. Compared to the C-SDF method, the orientation error is reduced by about 45.94%, and compared to the GJK-EPA method, by about 77.16%. Fig. 4 illustrates screenshots of the stick rotation task using different methods across six trials, where the current time is shown in the top-right corner, and the right screenshots depict the last step of each trial. It shows that both the C-NNLS method and the C-SDF method can complete all stick reorientations, while the GJK-EPA method fails in Trial 5. Moreover, the total time consumption of the C-NNLS method reduces approximately 69.32% compared with the C-SDF method in the stick rotation task. Because the C-SDF method needs matrix derivatives to compute the collision point, while the C-NNLS method can avoid the problem. Therefore, the C-NNLS method not only completes all the tasks but also achieves a lower average orientation error in less time.

C. Discussion on Hyperparameter $\boldsymbol{K}_{cd}$

Computing the inverse of a Hessian matrix in high-dimensional space usually requires significant time cost, which can affect the real-time performance of the entire control system. Inspired by [20], to reduce the time cost, the Hessian matrix inverse in (9) is replaced with $\boldsymbol{K}_{\text{cd}}$, where

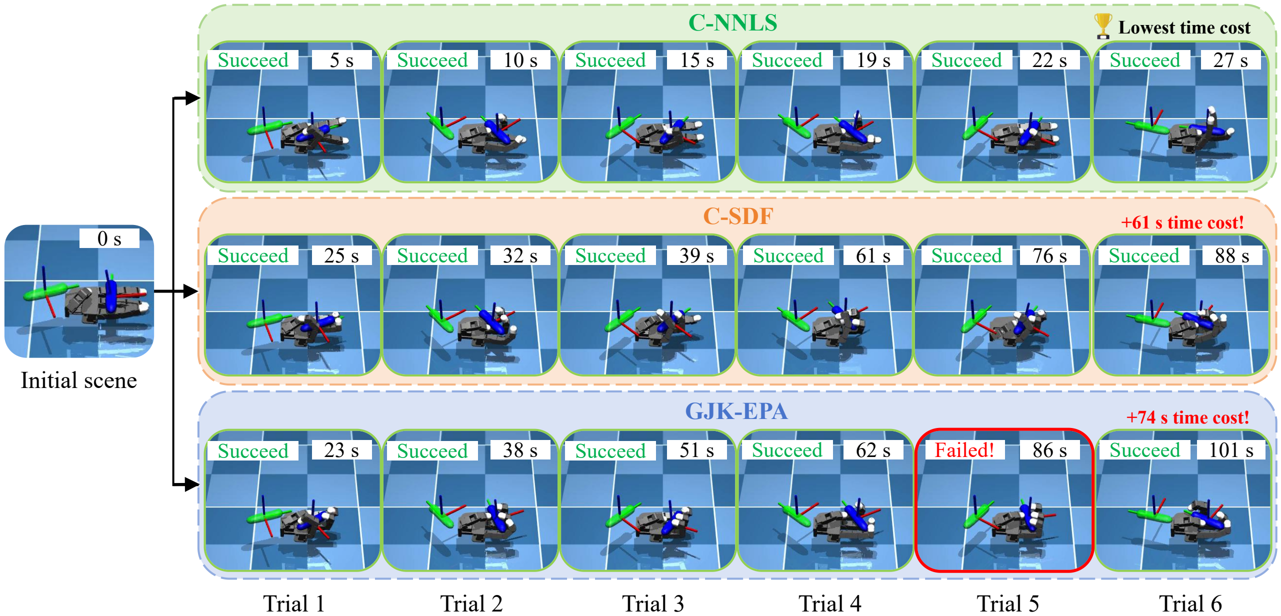


Fig. 4: Screenshots of the stick rotation task using different methods, where the right snapshots show the scenarios at the last step of each trial, with the current time displayed in the top-right corner.

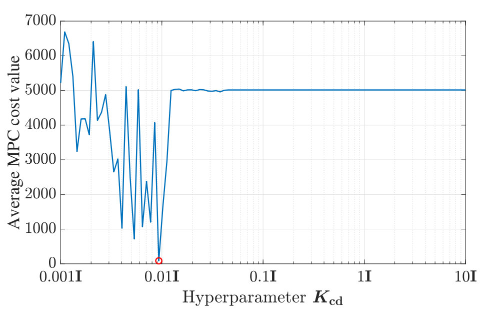


Fig. 5: MPC performance with different $\boldsymbol{K}_{\rm cd}$ in the cube rotation task, where the average MPC cost value is computed by two trials at the last step.

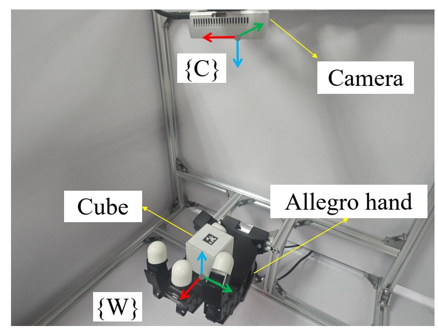


Fig. 6: Experimental setup.

$\boldsymbol{K}_{\rm cd}$ is set as a positive multiple of the identity matrix. In the above implementation, $\boldsymbol{K}_{\rm cd}$ is set with different values in the rotation task for different object. For example, in the cube rotation task, two trials are run for each $\boldsymbol{K}_{\rm cd}$ sampled from $10^{-3}\boldsymbol{I}$ to $10\boldsymbol{I}$. The average MPC cost value of two trials at the last step is used to evaluate the choice of $\boldsymbol{K}_{\rm cd}$ as shown in Fig. 5, indicating that the performance is influenced by the hyperparameter $\boldsymbol{K}_{\rm cd}$. Then, hyerparameter $\boldsymbol{K}_{\rm cd}$ with the lowest average MPC cost value is chosen, which is the red mark in Fig. 5. After that, the hyperparameter $\boldsymbol{K}_{\rm cd}$ is fine tuned to achieve the desired behaviour for the robotic hand.

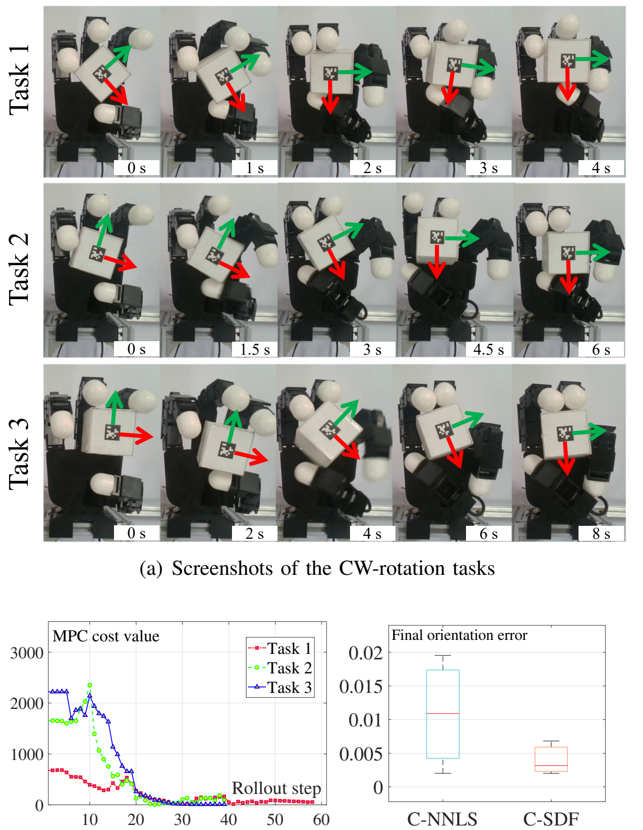
VI. Experimental Validation

In this section, the C-NNLS method is deployed on a real Allegro hand in the real-world dexterous manipulation to validate its feasibility and efficiency.

A. Experimental Setup

1) Hardware Implementation: The experimental setup is shown in Fig. 6. The Allegro hand has four 4-DoF fingers which are actuated by PD controllers. Because of the gap between simulation settings and the real world, the gain matrices in the simulation differ from those in reality. Note that excessive gain settings cause violent movement in the robotic hand, leading to system instability. In contrast, overly low gain settings resulted in sluggish or no movement of the robotic hand. Therefore, the gains $k_{\rm p}$ and $k_{\rm d}$ are tuned manually to achieve the desired behavior for each finger. The size of the cube object is $56 \times 56 \times 56$ mm, with the top face attached with an AprilTag for real-time pose estimation [25]. An RGB-D camera (Intel D435i RealSense) is responsible for collecting real-time RGB information used to calculate the pose of the object. Then, robot operating system [26] is used to establish communication between different modules.

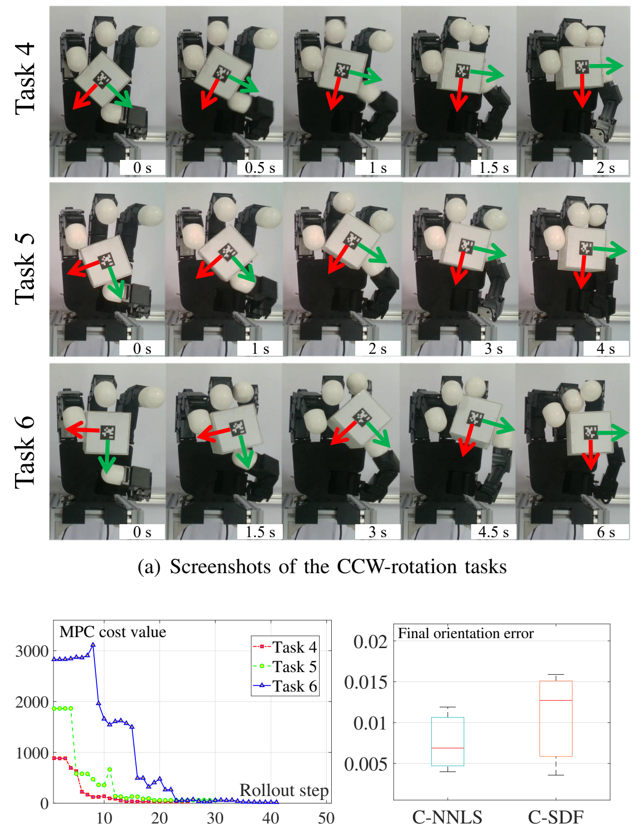
2) Task Setup: The task is to rotate the cube object from an initial orientation to the goal orientation along the z-axis, and the rotation goals are $\{\pm\pi/4, \pm\pi/3, \pm\pi/2\}$, which includes CW and CCW orientation. The initial pose settings of the hand are consistent with the simulation. Meanwhile, the query point setup and MPC setup follow [7], where



(a) Screenshots of the CW-rotation tasks

(b) MPC cost results (c) Final orientation errors

Fig. 7: Experimental results of the cube CW-rotation tasks. (a) and (b) shows the screenshots at different time and MPC cost values using the C-NNLS method, respectively. (c) Comparison of final orientation errors of the cube CW-rotation.



(a) Screenshots of the CCW-rotation tasks

(b) MPC cost results (c) Final orientation errors

Fig. 8: Experimental results of the cube CCW-rotation tasks. (a) and (b) shows the screenshots at different time and MPC cost values using the C-NNLS method, respectively. (c) Comparison of final orientation errors of the cube CCW-rotation.

cost weights $\{a_\text{c}, a_\text{u}, a_\text{p}, a_\text{q}\}$ are set to $\{1, 0.2, 500, 5000\}$, the control bound $\{\mathbf{u}_\text{lb}, \mathbf{u}_\text{ub}\}$ is set to $\{-0.15, 0.15\}$ and MPC horizon prediction $T = 3$.

3) Baseline: In evaluation, the proposed C-NNLS method is compared with the C-SDF method [7]. Both methods share the same time-stepping prediction routine and the same MPC cost function.

4) Metric: Both collision detection methods are evaluated in the z-axis rotation task, where the terminal orientation error E_ori and the task duration D_task are metrics. Note that the terminal orientation error is computed at the last step. Once $E_\text{ori} \leq 0.04$ is satisfied consecutively for 18 rollout steps, the control node will be terminated.

B. Results and Analysis

Figs. 7 and 8 show experimental results of the CW and CCW rotation tasks, respectively. It shows that the C-NNLS method can accomplish both CW and CCW tasks successfully, with better performance in CCW tasks, while the C-SDF method is superior in CW tasks. TABLE IV summarizes the experimental results of cube rotation around the z-axis. Both methods successfully complete the tasks. The C-NNLS method's final orientation error E_ori, though slightly higher than that of the C-SDF method, remains on the same 10^{-3} order of magnitude. Meanwhile, the C-NNLS method reduces the average task duration D_task by 50.5 steps. This indicates that the C-NNLS method achieves high-quality reorientation results, while reducing D_task by approximately 30.33%. Thus, the feasibility and efficiency of the C-NNLS method are demonstrated in the real-world dexterous manipulation.

TABLE IV: Comparison in real-world z-axis rotation

Orientation	C-NNLS		C-SDF [7]	
	E_ori	D_task	E_ori	D_task
CW	1.08×10^{-2}	136	4.00×10^{-3}	126
CCW	7.59×10^{-3}	96	1.07×10^{-2}	207
Average	9.20×10^{-3}	**116**	$\mathbf{7.36 \times 10^{-3}}$	166.5

E_ori is the average of the final orientation errors and D_task is the total sum of steps from three different target rotation tasks.

VII. Conclusions and Future Work

In this paper, a C-NNLS method is proposed to approximate the collision detection in MPC for dexterous manipulation, which avoids iterative optimizations and matrix derivatives. Results show that the C-NNLS method achieves comparable-quality reorientation while costing significantly less time.

However, the C-NNLS method is currently applied only to the on-palm z-axis reorientation tasks. Therefore, reorientation tasks in other axes will be explored with more accurate object pose detection methods in the future. Moreover, the current hyperparameter settings are heuristic, which can effectively improve real-time performance but inevitably introduce computing errors. In the future, hyperparameters will be tuned through an adaptive framework to achieve a better balance between real-time performance and solution accuracy.

REFERENCES

[1] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Automat.*, vol. 4, no. 2, pp. 193–203, Aug. 2002.

[2] B. Mirtich, "V-Clip: Fast and robust polyhedral collision detection," *ACM Trans. Graph.*, vol. 17, no. 3, pp. 177–208, Jul. 1998.

[3] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 1991, pp. 9–12.

[4] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3D game objects," in *Game Developers Conference*, vol. 170, 2001, p. 209.

[5] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 5026–5033.

[6] S. An, S. Lee, J. Lee, S. Park, and D. Lee, "Collision detection between smooth convex bodies via riemannian optimization framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2024, pp. 12 464–12 471.

[7] W. Yang and W. Jin, "ContactSDF: Signed distance functions as multi-contact models for dexterous manipulation," *IEEE Robot. Autom. Lett.*, vol. 10, no. 5, pp. 4212–4219, May 2025.

[8] A. Bemporad, "A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control," *IEEE Trans. Automat. Control*, vol. 61, no. 4, pp. 1111–1116, Apr. 2016.

[9] S. Ruan, X. Wang, and G. S. Chirikjian, "Collision detection for unions of convex bodies with smooth boundaries using closed-form contact space parameterization," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 9485–9492, Jul. 2022.

[10] G. v. d. Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graph. Tools*, vol. 4, no. 2, pp. 7–25, Jul. 1999.

[11] M. Montanari, N. Petrinic, and E. Barbieri, "Improving the GJK algorithm for faster and more reliable distance queries between convex objects," *ACM Trans. Graph.*, vol. 36, no. 3, pp. 1–17, Jul. 2017.

[12] L. Montaut, Q. L. Lidec, V. Petrik, J. Sivic, and J. Carpentier, "Collision detection accelerated: An optimization perspective," in *Robot.: Sci. Syst. (RSS)*, 2022.

[13] L. Montaut, Q. Le Lidec, V. Petrik, J. Sivic, and J. Carpentier, "GJK++: Leveraging acceleration methods for faster collision detection," *IEEE Trans. Robot.*, vol. 40, pp. 2564–2581, Apr. 2024.

[14] K. Tracy, T. A. Howell, and Z. Manchester, "Differentiable collision detection for a set of convex primitives," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 3663–3670.

[15] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2012, pp. 3859–3866.

[16] Y. Li and A. Ngom, "Classification approach based on non-negative least squares," *Neurocomputing*, vol. 118, pp. 41–57, Oct. 2013.

[17] Z.-M. Zhang, X.-Q. Chen, H.-M. Lu, Y.-Z. Liang, W. Fan, D. Xu, J. Zhou, F. Ye, and Z.-Y. Yang, "Mixture analysis using reverse searching and non-negative least squares," *Chemometr. Intell. Lab.*, vol. 137, pp. 10–20, Oct. 2014.

[18] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Müller, A. Evans, D. Fox, J. Kautz, and S. Birchfield, "BundleSDF: Neural 6-DoF tracking and 3D reconstruction of unknown objects," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recog. (CVPR)*, Jun. 2023, pp. 606–617.

[19] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi, "CvxNet: Learnable convex decomposition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recog. (CVPR)*, Jun. 2020, pp. 31–44.

[20] W. Jin, "Complementarity-free multi-contact modeling and optimization for dexterous manipulation," in *Robot.: Sci. Syst. (RSS)*, 2025.

[21] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[22] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, pp. 25–57, 2006.

[23] M. W. Spong, "Robot modeling and control," *IEEE Control Syst.*, vol. 42, no. 1, pp. 126–128, Jan. 2022.

[24] S. Brahmbhatt, C. Ham, C. C. Kemp, and J. Hays, "ContactDB: Analyzing and predicting grasp contact via thermal imaging," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recog. (CVPR)*, Jun. 2019, pp. 8709–8719.

[25] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2011, pp. 3400–3407.

[26] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: An open-source robot operating system," in *Proc. IEEE ICRA Workshop Open. Source Softw.*, vol. 3, no. 3.2, 2009, p. 5.