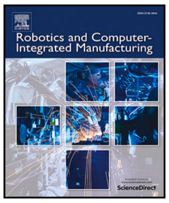
Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

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



Full length article

Online dual robot–human collaboration trajectory generation by convex optimization

Lai Wei [a,d], Yanzhe Wang [c,e], Yibo Hu [c,e], Tin Lun Lam [b,d,*], Yanding Wei [c,e,**]

[a] *School of Data Science, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China*
[b] *School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China*
[c] *The State Key Laboratory of Fluid Power and Mechatronic Systems, School of Mechanical Engineering, Zhejiang University, Hangzhou 310027, China*
[d] *Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, 518129, China*
[e] *Key Laboratory of Advanced Manufacturing Technology of Zhejiang Province, School of Mechanical Engineering, Zhejiang University, Hangzhou 310027, China*

ARTICLE INFO

Keywords:
Human-dual robot collaboration
Convex based path planning
Disciplined parameterized programming
Online mutual collision avoidance

ABSTRACT

For dynamic collision-free trajectory planning in dual-robot and human collaborative tasks, this paper develops an online dual-robot Mutual Collision Avoidance (MCA) scheme based on convex optimization. A novel convex optimization formulation model, named Disciplined Convex programming by Shifting reference paths (DCS), is proposed for solving the single-robot trajectory optimization problem. Furthermore, a new dual-robot trajectory convex optimization algorithm is presented for online adjustment of the dual-robot trajectories according to the collaborative task priority. The overall pipeline, named DCS-MCA, generates collision-free and time-optimal dual-robot trajectories, while prioritizing the task accessibility of the high-priority robot. Simulation experiments demonstrate that DCS exhibits comparable performance to the current state-of-the-art single-robot motion planner, while the DCS-MCA outperforms common algorithms by up to 30% in time optimality for dual-robot collaborative tasks. The feasibility and dynamic performance of the proposed approach are further validated in a real collaborative cell, illustrating its suitability for collaborative dual-robot tasks in moderately dynamic environments.

1. Introduction

With the application of collaborative robots in industry goes deeper, the field of human–robot collaboration has made great strides. The early single-robot and human cooperation is no longer sufficient for the task in certain complex industrial scenarios, where multiple robots and humans need to work together [1]. Collaborative dual robots are deployed to assist humans in completing tasks such as assembly [2] and material handling [3]. This new requirement places higher demands on the flexibility and responsiveness of the robot, as multiple robots and humans share the same workspace where human collaborators and other robots performing different tasks may create more uncertainty for the robot. Since the hybrid environment is complex and dynamic, how each robot can orderly plan its path to avoid other robots, human operators, and obstacles in the environments while satisfying the overall task execution efficiency becomes the primary challenge in this field today. In the context of joint collaboration between dual robots and humans, this paper proposes an innovative solution for online collision-free trajectory planning of dual-robot while considering both dynamic and static robot workspaces, task priority constraints, and human collaboration safety conditions.

The classical single-robot motion planning algorithms are divided into the following categories: sampling-based planning algorithms, neural network (NN)-based algorithms, and optimization-based planning algorithms.

Sampling-based planning algorithms include PRM [4,5] with its variants and RRT [6–8] with its variants, among which Rapidly exploring Random Trees (RRT) and especially RRT* are the most popular sampling-based motion planning algorithms. RRT provides probabilistic completeness and RRT* achieves asymptotic convergence for solution. Kaltsoukalas et al. [9] proposed an algorithm based on intelligent sampling on a high-dimension joint space grids. Wang et al. [10] proposed an improved RRT* algorithm based on the sampling pool and restricted nearest node strategy. Recently, Merckaert et al. [11] introduced a human–robot collaboration planning and control framework that combines the RRT planner with an explicit reference governor. Sampling-based planning algorithms necessitate a delicate balance

* Corresponding author at: School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China.
** Corresponding author.
E-mail addresses: tllam@cuhk.edu.cn (T.L. Lam), weiyd@zju.edu.cn (Y. Wei).

https://doi.org/10.1016/j.rcim.2024.102850
Received 11 March 2023; Received in revised form 23 July 2024; Accepted 2 August 2024

between planning efficiency and path quality. Particularly in high-dimensional and intricate planning spaces, achieving both asymptotic optimality and search efficiency poses a formidable challenge. The dynamic performance of planners typically comes at the expense of compromising the quality of the generated paths.

In contrast, the NN-based methods are superior to sampling-based methods in terms of dynamic performance. One famous NN-based planner is MPNet [12], which used two sub-networks to generate a series of critical waypoints, with which a collision-free optimized path was generated by the post-processing algorithms. Recently, another NN approach [13] based on the distribution of robot empirical configurations has been proposed for human–robot collaboration. The neural network is used to produce discrete path points in the configuration space, and then the trajectories are generated by improved STOMP optimization. Duguleana et al. [14] proposed an redundant manipulator collision avoidance motion planning algorithm by reinforcement learning. However, these methods seldom consider kinetic constraints, and they usually require additional steps to optimize and smooth the final solution path. In addition, NN-based methods need large scale of training data and are lack of interpretability.

On the other hand, classical optimization-based planning algorithms do not rely on any prior data and the kinetic constraints can be considered. Algorithms such as CHOMP [15] and STOMP [16] start from an initial trajectory connecting the initial and target configurations, and then optimize iteratively to satisfy collision-free constraints and kinetic constraints while minimizing the length cost. Therefore, the trajectories produced by optimization-based algorithms are generally smooth and locally optimal. However, these algorithms rely on gradient information, which requires differentiable cost function, and the solution trajectory quality may be significantly influenced by the initial trajectory. While the reference database, path selection and modification steps of DCS-MCA algorithm provide good initialization to avoid the impact of poor random initial trajectories. Convex optimization is another approach, but many constraints in motion planning such as obstacle avoidance are naturally non-convex, which is difficult to be formulated as convex optimization problems. Various methods have been developed in existing research to deal with nonconvexities. Açikmese et al. [17] introduced lossless convexification by expanding the space. Liu et al. [18] applied successive linear approximations to remove nonconvex constraints. TrajOpt [19] attempted to use sequential convex optimization for planning. GCS [20] innovatively combined convex optimization with graphs, to significantly reduce the complexity of the problem. Most of these algorithms are faced with the trade-off between the solving efficiency and the preservation of the search domain.

To further address the above problems, this paper proposes an empirical path as the initial trajectory of the optimization in convex optimization algorithm for single-robot obstacle avoidance. With the initial trajectory and the given dynamic and static obstacles, a convex objective function is designed and a series of techniques to transform the nonconvex constraints into a convex one are applied. In addition, the problem is transformed as a DPP problem, and the computational efficiency is greatly improved. In the process of approximating the non-convex conditions, DCS tries to ensure that the optimality of the solution is not affected. With the help of off-the-shelf convex optimization solver, the proposed optimization method can solve the problem fast and optimally, thus can be applied to online collision avoidance trajectory planning tasks.

Based on the above single-robot trajectory planning methods, various studies have proposed different schemes to solve the collaborative path planning problem of dual robots. Zhang et al. [21] presented a recursive neural network (RNN)-based mutual collision avoidance algorithm to tackle the dual-robot motion planning problem. Wong et al. [22] offered a soft agent-critical (SAC) based motion planning method to enable the robot to effectively avoid self-collision, with two neural networks controlling two robot arms motion direction respectively. Choi et al. [23] presented a B-spline trajectory planning algorithm for a two-armed robot that can produce collision-free and smooth trajectories. The above approaches only focus on the obstacle avoidance relationship between the two robots in the spatial dimension, i.e., they require the trajectory path points of the two robots do not intersect at all. However, those dual-robot trajectory planning models are not suitable for some practical industrial applications. For example, one robot is performing an urgent task that it must cross the other robot's trajectory, or multiple robots need to pass through a certain intersection. Reasonable solutions in the temporal dimension need to be considered as well. To the authors' knowledge, there is no optimization-based collaborative trajectory planning method for two robots that can plan obstacle avoidance trajectories considering both spatial and temporal dimensions under task priority constraints.

This paper adopts a convex optimization approach to obtain the dual robot collision free motion trajectory. Based on the results from the previous step of the single-robot path generation, the timestamps of the two robots passing through the critical sections are adjusted. The solution pipeline can get optimal trajectories fast. Simulations and real experiments verify the effectiveness of the algorithm under various testing scenarios.

Major contributions of this paper are: (1) Utilize a novel pattern to transform the trajectory generation problem into a batch of convex optimization problems while preserving the nature of the original problem. (2) Boost the solving speed by converting the formulated problem into a DPP problem. (3) Propose a novel pipeline in optimizing dual robots trajectories to avoid collision with each other. (4) Apply some techniques in improving trajectory quality and computation efficiency.

2. Problem construction and algorithm overview

2.1. Problem construction

The DCS-MCA method proposed in this paper is applicable to solve the dual robot–human dynamic collision avoidance problem. Two robots workspaces are $S_{R1}, S_{R2} \subset \mathbb{R}^3$ respectively and the human workspace is $S_{human} \subset \mathbb{R}^3$. The robots-human shared workspace is $S_{share} = (S_{R1} \cap S_{R2}) \cup (S_{R1} \cap S_{human}) \cup (S_{R2} \cap S_{human})$. The method plans two robots motion trajectory online while (1) finishing the tasks based on the priority. (2) not colliding with the static obstacles in S_{R1}, S_{R2}. (3) not colliding with other robot and human in S_{share}.

To simplify the algorithm design works, all obstacles in the workspace are approximated as spheres in different radius, and the static obstacles spheres in the system can be input in advance. Dynamic human arms can be modeled as a sequence of balls along the arm, and arm localization can be obtained by sensors like cameras, IMUs, lasers and magnetic trackers.

2.2. Algorithm overview

A trajectory generation algorithm DCS-MCA that satisfies obstacle avoidance requirements by convex optimization for dual-robot end-effectors is proposed in this paper, as shown in Fig. 1. The main steps of the algorithm are briefly described as follows. First, two robots' obstacle collision-free trajectories are obtained separately by the preprocessing steps, step 1 and step 2. This routine is also known as DCS. So far, two robots are time optimal to the destination with obstacle collision-free property, but since their paths are optimized individually, the robots may collide with each other. Therefore, in the second step, some valid trajectory combination candidates are made from the previous step. If the candidate has robots spatial–temporal intersection, a secondary optimization procedure (step 3) is executed to optimize the low priority robot's time series to avoid collision to the high priority robot. Eventually, the solution with the shortest time among all feasible candidates is selected as the final solution. In this way, two robots are free of collision

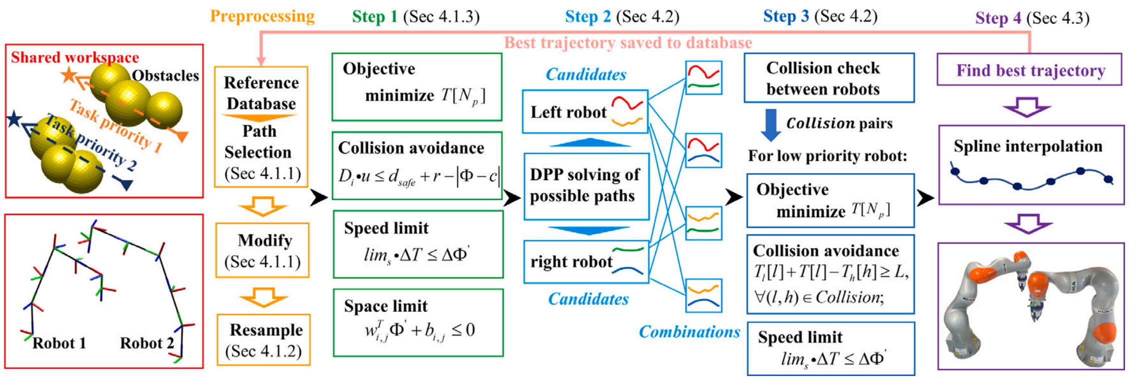

Preprocessing
Step 1 (Sec 4.1.3)
Step 2 (Sec 4.2)
Step 3 (Sec 4.2)
Step 4 (Sec 4.3)
Best trajectory saved to database
Shared workspace
Obstacles
Task priority 1
Task priority 2
Robot 1
Robot 2
Reference Database
Path Selection (Sec 4.1.1)
Modify (Sec 4.1.1)
Resample (Sec 4.1.2)
Objective minimize $T[N_p]$
Collision avoidance $D_i \cdot u \le d_{safe} + r - |\Phi - c|$
Speed limit $lim_s \cdot \Delta T \le \Delta\Phi'$
Space limit $w_{i,j}^T \Phi' + b_{i,j} \le 0$
Candidates
Left robot
DPP solving of possible paths
right robot
Candidates
Combinations
Collision check between robots
Collision pairs
For low priority robot:
Objective minimize $T[N_p]$
Collision avoidance $T_l[l]+T[l]-T_h[h] \ge L, \forall(l,h) \in Collision;$
Speed limit $lim_s \cdot \Delta T \le \Delta\Phi'$
Find best trajectory
Spline interpolation

Fig. 1. Overall structure of the DCS-MCA algorithm.

with each other while maintaining the optimality. The trajectories are finally generated through interpolation (step 4).

The general algorithmic of DCS-MCA steps are given in Algorithm 1. The details of the above steps are elaborated in the following sub-sections.

Algorithm 1: Overall algorithmic steps for the proposed DCS-MCA approach

```
Input: Left, right robot tasks with priority, and the obstacles
Output: Optimized left, right path with timestamp,
        $\{\Phi'_L, T_L\}, \{\Phi'_R, T_R\}$
1. Single-robot trajectory generation (two robots separately):
   a. Generate reference path $\Phi$ of the task from the database;
   b. Re-sample the reference path $\Phi$;
   c. Construct batch of disciplined parameterized programming
   problems;
   d. Solve the problems and get the solution paths $\{\Phi'_i, T_i\}$;
2. Make joint robot path combinations $\{\Phi'_{L,i}, T_{L,i}\} \times \{\Phi'_{R,j}, T_{R,j}\}$
   from step 1.
3. Optimize joint robot path combinations:
   a. Construct joint-trajectory optimization problems;
   b. Choose the best trajectory $\{\Phi'_L, T_L\}$, $\{\Phi'_R, T_R\}$ among all
   combinations;
4. Interpolate the final trajectory.
```

3. Algorithm design

3.1. Single robot trajectory generation

The aim of single robot trajectory generation in DCS is to obtain a single robot obstacle collision-free optimal path $\Phi' \in \mathbb{R}^{N_p \times 3}$ (where N_p denotes number of points on the path) with corresponding timestamp at each waypoints $T \in \mathbb{R}^{N_p}$ from the reference experience path $\Phi \in \mathbb{R}^{N_p \times 3}$. One straight-forward idea to avoid collision with obstacles is to shift the influenced points to the free space. To make the problem work well with high efficiency, DCS algorithm consists of 3 sub-steps described below.

3.1.1. Reference path generating

The reference database stores experience paths in the task object, a tuple of four elements (*start*, *end*, *obs*, *path*), where *start* and *end* are Cartesian positions, and *obs* is the obstacle sets. The path Φ, which has the highest similarity to the current task, is selected as the reference path from the database. The similarity S between the current task and tasks in the experience database can be calculated by the formula (1) and λ is a scale factor, which equals 0.1 in our case.

$$S(task1, task2) = \frac{1}{S_{points} + \lambda S_{obstacle}} \tag{1}$$

Where S_{points} utilizes second norm to measure the similarity between two tasks' starting and ending points respectively. In addition, if the total distance difference is larger than 0.1, this term is set to 0 directly.

$$S_{points} = \|start_1 - start_2\| + \|end_1 - end_2\| \tag{2}$$

The Hausdorff distance is naturally suitable to measure the similarity of two obstacle sets, so the $S_{obstacle}$ is defined as formula (3), with the distance function d defined as the distance between two obstacles' centers.

$$S_{obstacle} = \max\{ \sup_{a \in obs_1} \inf_{b \in obs_2} d(a, b), \sup_{b \in obs_2} \inf_{a \in obs_1} d(b, a)\} \tag{3}$$

Because calculating the Hausdorff distance is inefficient, a practical approach is adopted. Initially, S_{points} for all tasks are calculated and the top-K task candidates are selected, where $K = 8$ in our case. Following this, the total similarity for each of these candidates is determined to identify the best reference path.

If no similar path can be found (i.e., the best similarity is smaller than a threshold), the line segment from the current position to the destination is used as the reference path. Two techniques to increase the quality of the generated reference path are applied. First, for each waypoint on the reference path, it is modified to be the point at which the robot's end effector and the end arm are closest to any obstacle. This allows for better modeling of the distance between the reference trajectory and obstacles, increasing the success rate of solving the later problem. In addition, to match the problem construction part, adjusting the reference path is needed so that all waypoints of the path are all on the same side of each obstacle. An example is shown in Fig. 2. The fitted plane of the internal points of each obstacle is obtained by techniques such as linear regression. Then, all points are shifted to the same side of the plane that passes through the center of the obstacle and is parallel to the fitted plane. In this way, all the reference points are in the same side of the obstacle, which brings much convenience in further problem formulation in Section 3.1.3.

3.1.2. Reference points re-sampling

A uniform-sampled waypoint reference path, as discussed in Section 3.1.1 is not required. This is due to the fewer points needed to construct a new path in an obstacle-free space. On the contrary, in the space near the obstacle, more points are required to build a smoother path. Inspired by [24], an algorithm is designed to resample the waypoints on the path. First, the path is uniformly partitioned to N_m parts. Second, distance between each partition to the nearest obstacle surface is calculated, denoting as d_i. Then, a designed probabilistic function $p_i = f(d_i)$ (formula (4)) is applied to each segment. Finally, all $p_i = p_i / \sum_{j=1}^{N_m} p_j$ are normalized, and a piece-wise constant probability density function is produced along the path segments. Total of $\lceil 0.6 \cdot N_m \rceil$ points from this distribution are sampled using inverse transform sampling method. This procedure allocates more waypoints to the

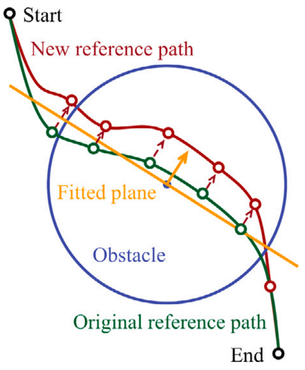


Fig. 2. Path modification.

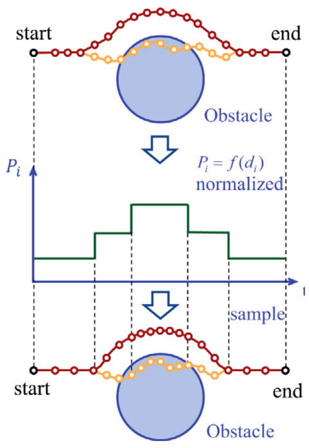


Fig. 3. Path resample.

regions near the obstacle-conflicting space while leaving few points in the free space. As a result, the total number of waypoints along the path decreased, which reduces problem complexity without affecting the solving stage. Fig. 3 shows a resampling example.

$$f(d_i) = \begin{cases} 3, & d_i \in (-\infty, 0.025 \text{ m}) \\ 2, & d_i \in [0.025 \text{ m}, 0.075 \text{ m}) \\ 1, & d_i \in [0.075 \text{ m}, +\infty) \end{cases} \tag{4}$$

3.1.3. DPP construction

By the definition of convexity, traditional collision avoidance constraints such as the norm distance larger than a constant value are non-convex. To overcome this gap, the problem is constructed in a novel way, that is, approximating the feasible region into a frustum. To maintain the size of search domain, DCS would also check if the new path can be on the other side of the obstacle. As shown in Fig. 4, if there is an obstacle blocked on the reference path (shown in blue line), the algorithm would try to obtain the solution on two sides of the obstacle (shown in red and green lines).

It is worth noting that if there is n collided obstacles,A batch of 2^n convex problems need to be solved, which may be very time consuming. Therefore, it is further formulated into a disciplined parameterized programming problem. A good property of DPP is that the consecutive compilation of the same problem with different parameters only takes a much shorter time.

Now, the discussion focuses on how to formulate the disciplined parameterized convex problem. The set *Free*, denotes the points that are out of any obstacles, and the set *Occupied* denotes the points that are in any obstacles. Each consecutive index range in *Occupied* are defined as a collision segment. The set of waypoint index ranges is denoted

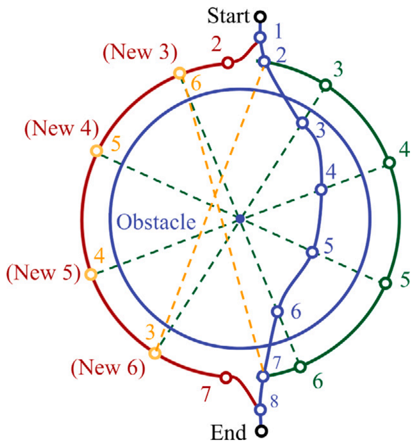


Fig. 4. Paths in two directions around an obstacle.

as $Intersect = \{(start_i, end_i)\}_{i=1}^{N_c}$ for a total of N_c collision segments. In each range, $start_i$ denotes the starting waypoint index of ith collision segment, and end_i denotes the ending waypoint index of ith segment. The occupied points in each collision segment $\{(start_i, end_i)\}$ are mirrored through the center of the obstacle to form the opposite side path. As shown in Fig. 4, it is easy to observe that the path waypoint traverse indices in original side and opposite side are different. For example, in this figure, the original side traverse path (in green) in order of [2-3-4-5-6-7]. On the contrary side, the points (in orange) are visited in the order of [2-6-5-4-3-7].

To make the problems unified as a disciplined parameterized convex problem, the side-control parameters are introduced, $P_i \in \{0, 1\}$, to control whether the new path would pass-by the ith obstacle on the original side or on the opposite side. The parameters work like a switch. When it is 1, the constraints make the path go from the original side activates, forming a path in the original side. When it turns to 0, the constraints force to form a path in the opposite side. In this way, the problem formulation can be easily changed without affecting its disciplined parameterized convex property. Details are explained in the following part.

Decision Variables. There are spatial variables, $D \in \mathbb{R}^{N_p \times 3}$, representing each reference point's offset to the new path. That is, new path $\boldsymbol{\Phi}' = \boldsymbol{\Phi} + D$. In addition, there are time variables, $T \in \mathbb{R}^{N_p}$, representing the timestamps that the robot arrives at the waypoints.

Objective Function. The goal is to minimize the robot motion time. Therefore, the robot should arrive the destination (at the last index, N_p) as soon as possible. So the objective function is:

$$\underset{D,T}{\text{minimize}} \quad T[N_p] \tag{5}$$

Constraints. First constraint are designed for decision variables' internal properties. The start position and the end position should not be changed during the optimization, which means that

$$D_1 = \mathbf{0}, \quad D_{N_p} = \mathbf{0} \tag{6}$$

Also, all new waypoints should be in the N_b workspace polygon boundary planes $\mathbf{w}^T \cdot x + b \leq 0$:

$$\mathbf{w}_{i,j}^T \cdot \boldsymbol{\Phi}'_i + b_{i,j} \leq 0, \quad \forall i = 1, \dots, N_p, \; j = 1, \dots, N_b \tag{7}$$

Second, the robot cannot arrive latter point earlier than the former point. This constraint sets are different among points in *Free* and points in *Occupied*. For points in *Free*, the constraints are

$$T_i \leq T_{i+1}, \quad \forall i \in \text{index}(Free) \tag{8}$$

where index($\cdot$) returns the index of the input point. As for points in *Occupied*, the constraints are related to the side-control parameter P_i.

For any $\{(start_j, end_j)\}$ segment in *Intersect*,

$$\begin{cases} P_j \cdot (T_{i+1} - T_i) \geq 0 \\ (1 - P_j) \cdot (T_{i+1} - T_i) \leq 0, \end{cases} \quad \forall i = start_j, \ldots, end_j \tag{9}$$

Note that when $P_j = 1$, first inequality $T_{i+1} \geq T_i$ takes effects while second equation $0 \leq 0$ always holds, taking no effects. This is the constraints for the original side. When $P_j = 0$, $T_{i+1} \leq T_i$, which are the constraints for the opposite direction.

Other than internal constraints, there are external speed constraints and no-collision constraints. The speed constraints need to be considered in *Free* and *Occupied* separately. For *Free* set, it is supposed that the speed between two consecutive reference points should be smaller than the maximum speed limitation $\lim_s$. That is,

$$\|\Phi'_{i+1} - \Phi'_i\| \leq \lim_s \cdot (T_{i+1} - T_i), \quad \forall i \in \text{index}(Free) \tag{10}$$

As for points in *Occupied* set, the similar idea is used as the time constraints mentioned above to construct the speed constraints. For any $\{(start_j, end_j)\}$ segment in *Intersect*,

$$\begin{cases} \|\Phi'_{i+1} - \Phi'_i\| \leq \lim_s \cdot (T_{i+1} - T_i) + M \cdot (1 - P_j), \\ \qquad \forall i = start_j - 1, \ldots, end_j; \\ \|\Phi'_{i+1} - \Phi'_i\| \leq \lim_s \cdot (T_{i+1} - T_i) + M \cdot P_j, \\ \qquad \forall i = start_j, \ldots, end_j - 1; \\ \|\Phi'_{start_j - 1} - \Phi'_{end_j}\| \leq \lim_s \cdot (T_{end_j} - T_{start_j - 1}) + M \cdot P_j, \\ \|\Phi'_{start_j} - \Phi'_{end_j + 1}\| \leq \lim_s \cdot (T_{end_j + 1} - T_{start_j}) + M \cdot P_j, \end{cases} \tag{11}$$

where M is a very large number (In all experiments, M is set to 10^6). When $P_j = 1$, the first inequality takes effects, while the remaining three inequalities always holds. The speed constraints are set for the trajectories at the original side. On the contrary, when $P_j = 0$, the first inequality always holds while last three inequalities take effects. Since on the opposite direction, the indices traverse order is inverted, last three inequalities helps forming the speed limitation for the opposite side.

The most critical point is the constraint sets for obstacle avoidance. For all points in $Free \cup Occupied$, the constraint requires them not to enter other obstacles. The feasible set for $\Phi'_i = \Phi_i + D_i$ is a half-space defined by:

$$D_i \cdot \frac{\Phi_i - c}{\|\Phi_i - c\|} \leq d_{safe} + r - \|\Phi_i - c\|, \quad \forall i. \tag{12}$$

where $c \in \mathbb{R}^3$, $r \in \mathbb{R}_+$ is the center and radius of the obstacle. To improve computation efficiency, the obstacle that is close enough (in the experiment, it is 0.5 m) to the reference waypoint Φ_i is considered only.

Second, for the points in *Occupied* set, they need to be moved out of the obstacle. As shown in Fig. 4, the feasible set for $\Phi'_i = \Phi_i + D_i$ is the intersection of a cone and some half-spaces. Integrated with side-control parameters, the constraints for any $\{(start_j, end_j)\}$ segment in *Intersect* can be written as:

$$\begin{cases} D_i \cdot (2P_j - 1) \cdot \frac{\Phi_i - c}{\|\Phi_i - c\|} \geq d_{safe} + r - (2P_j - 1) \cdot \|\Phi_i - c\|, \\ D_i \cdot (2P_j - 1) \cdot \frac{\Phi_i - c}{\|\Phi_i - c\|} \geq \cos(angle) \cdot \|D_i\|, \end{cases} \tag{13}$$

$$\forall i = start_j, \ldots, end_j$$

The first inequality forms a half-space, making the points move away from the intersected obstacle. The second inequality forms a cone with vertex angle equals *angle* degree. In all experiments, the angle is set to $\pi/6$. When $P_j = 1$, $2P_j - 1 = 1$, creating a feasible set at original side (Fig. 5, upper red region). When $P_j = 0$, the feasible set at the opposite side (Fig. 5, lower red region) activates.

Solving. Any mature DPP-enabled convex optimization solver can be used for solving this disciplined parameterized convex problem. The side-control parameters $\{P_i\}$ are continuously changed with all possible combinations, and the solver's results are recorded. The solution trajectories are then used in joint robots trajectory optimization in the next section.

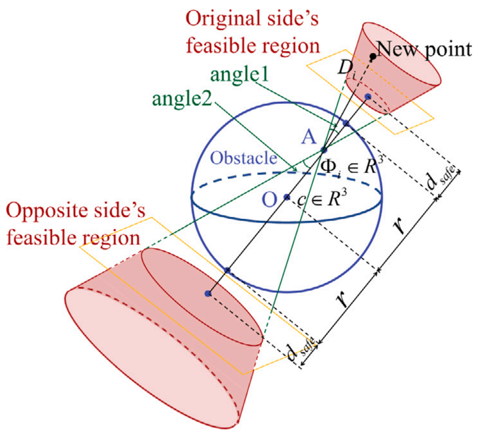


Fig. 5. Collision avoidance constraints in two directions around an obstacle.

3.2. Joint robots trajectory optimization

By now, two bundles of obstacle collision-free trajectories are obtained for left and right robots. As indicate in step 2 of Algorithm 1, best-N trajectories from two bundles are combined, forming N^2 candidate trajectories for the final solution. Some candidates do not have robots mutual collisions. But some candidate's trajectories may have collisions with each other. For these candidates, in step 3 of Algorithm 1 two robot's trajectories are jointly optimized, making it free of collisions.

Following are details in building this problem as a convex problem.

Decision Variables. The main strategy is to let low priority robot wait for high priority robot outside of the critical section. Therefore, the decision variable is the time delayed $T \in \mathbb{R}^{N_p}$ at each waypoint of low priority robot trajectory obtained from the last section.

Objective Function. The robot with high-priority's trajectory remains the same. The time for the low priority robot is intended to be shortened. That is,

$$\underset{T}{\text{minimize}} \quad T[N_p] \tag{14}$$

Constraints. The key constraints are two robots cannot get too close at the same time. First, all close waypoint pairs $\{l\_index, h\_index\}$ are generated from two robot trajectories, where l_index is the waypoint index for the low priority robot, and h_index for the high priority robot. Then the constraints can be simply defined as

$$T^{low}_{l\_index} + T_{l\_index} - T^{high}_{h\_index} \geq \lim_t \tag{15}$$

Same as the last section, there are both time constraints and speed constraints for all $i = 1, \ldots, N_p$:

$$\begin{cases} T^{low}_i + T_i \leq T^{low}_{i+1} + T_{i+1} \\ \|\Phi_{i+1} - \Phi_i\| \leq \lim_s \cdot (T^{low}_{i+1} + T_{i+1} - T^{low}_i - T_i) \end{cases} \tag{16}$$

where T_{low} and Φ is the low priority robot's timestamp and path obtained from the last section.

Solving. Any well-established commercial convex optimization solver can be used for solving. Notice that the path combinations are independent with each other, so multiprocessing techniques could be used in this process.

Finally, all candidates are free of collision with static obstacles and the other robot. The trajectory combination with the shortest operating time for both left and right robots is utilized as the final trajectory.

Now, the operating time with high priority robot remains the same, as optimal as before. The operating time of the low priority robot was also minimized by choosing the fastest combinations in the search space. Therefore, the proposed DCS-MCA algorithm guarantees the optimality of the trajectory of the left and right robots.

In addition, this new trajectory will be added in the experience database, which can be used later.

3.3. Interpolation

By this step, final result are two series of path waypoints. Quintic B-splines [25] interpolation can be applied to output continuous and smooth motion trajectories for industrial robot use.

The general equation of B-spline curve is described as

$$G(u) = \sum_{i=1}^{n} g_i N_{i,k}(u) \tag{17}$$

where u $(0 \le u \le 1)$ is the B-spline normalized trajectory parameter. k is the degree and $k+1$ is the order. g_i is the ith control point. $N_{i,k}$ is the ith basis function of the B-spline that satisfies the de Boor-Cox formula as follow

$$\begin{aligned} N_{i,k}(u) &= \frac{u-u_i}{u_{i+k}-u_i} N_{i,k-1}(u) + \frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}} N_{i+1,k-1}(u); \\ N_{i,0}(u) &= \begin{cases} 1, & u_i \le u \le u_{i+1} \\ 0, & \text{elsewhere} \end{cases} \end{aligned} \tag{18}$$

where $u_i\,(i = 1, 2, \dots, n+k+1)$ is the knot, which is parameterized by the time corresponding to the current position, and $U = [u_1, u_2, \dots, u_{n+k+1}]$ is the knot vector. Since $k = 5$, for N_p waypoints, a total of $N_p + 10$ knots are used for Quintic B-splines interpolation. After this step, the velocity and acceleration along the trajectory are smooth and continuous.

4. Experiments and result analysis

4.1. An industrial use case and implementation details

The approach is validated with an industrial use case of human–robot collaboration applied in component assembly. During the collaborative process, the human operator is at the side of the table, working in conjunction with two industrial robots that are positioned at the front corners of the workspace. The main tasks of two robots are moving metal parts between the loading area at the front and the assembly area at the back according to different assemble procedures. Fig. 6 shows the schematic diagram of the test environment. In this use case, there is a large intersection of the motion space of the two robots. In addition, some static obstacles (e.g., tools, monitors) and dynamic obstacles (e.g., operator's hands) interfere with two robots' motion space.

The test platform is a laptop computer configured with Apple M2 chip at 3.49 GHz. The workspace size is 0.8 m × 0.8 m × 0.5 m for both simulation and real experiments. The algorithm use the *CVXPY* [26] with solver *ECOS* [27] for solving the convex problems. *SciPy* [28] is used for path interpolation. In all of the simulation and real experiments, the algorithm's hyper-parameters are set as below: $N_p = 41$, $d_{safe} = 0.06$ m, $\lim_d = 0.20$ m, $\lim_t = 3s$, $\lim_s = 0.05$ m/s, $angle_1 = angle_2 = \pi/6$ rad. The reference paths utilized are consistently straight line segments between the starting and ending points.

4.2. Simulation experiment

4.2.1. Static simulation without human motion

In this part, three typical scenes are simulated to show the strength and properties of the proposed algorithm. The task information and obstacle information are manually input to the planner. The collision-free trajectories for both left and right robot are generated. The path is shown in the 3D figure, and the color on the path demonstrates the timestamp that the robot passes on that segment of path.

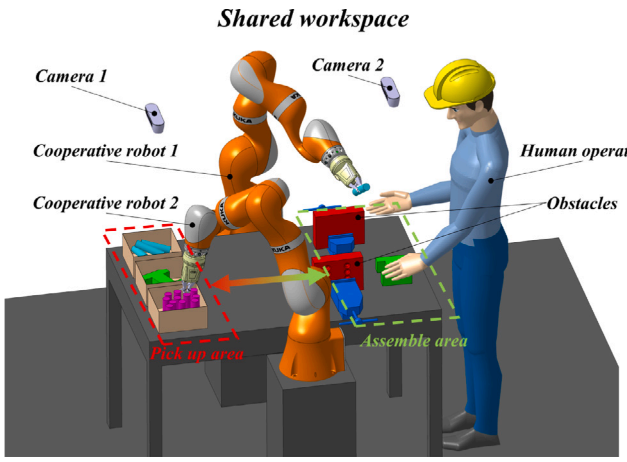


Fig. 6. Dual robot collaborative assembly workstation.

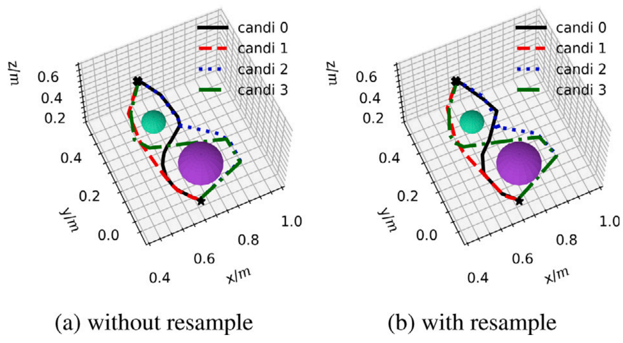


Fig. 7. Static Exp. 1 trajectories visualization.

For the static experiment 1, Fig. 7 shows that the DCS algorithm generates four candidate solutions when meeting two obstacles on the way. The search domain is large enough.

In addition, the Table 1 shows the effectiveness of the DPP formulation. It takes around four times of time to solve four different problems than do one problem without DPP technique, but with DPP, the computation time is merely a little more than that for solving one single problem.

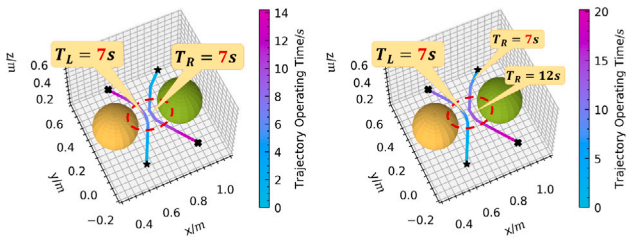
This scenario further explores the effectiveness of the proposed resampling technique. When the resampling procedure is enabled, the total solving time is 25% shorter than the time without resampling. Fig. 7 demonstrates that the path quality and optimality are kept the same under two settings.

Experiment 2 is a static test designed to demonstrate the third step in the DCS-MCA algorithm. In this experiment, the left robot has higher priority than the right robot. Fig. 8(a) shows the trajectories just after the step 2. The red dashed circle is the critical area, where the left and right robot will enter together 7 s after starting, so a crash may occur consequently. Fig. 8(b) shows the trajectories after the joint optimization (step 3), which optimizes the right robot speed. It can be observed that the lower priority robot, the right robot waits for the left robot to cross the middle channel outside of the critical section. By this schedule, two robots kept enough safety distance with each other.

The static experiment 3 shows the whole pipeline of the algorithm. After step1, each robot has two possible candidate paths. Then in the step 2 they are grouped, forming 4 combinations as shown in Figs. 9(a)–9(d). The step 3 of the DCS-MCA algorithm is applied to these combinations. For the combination 4, since two robots have tentative

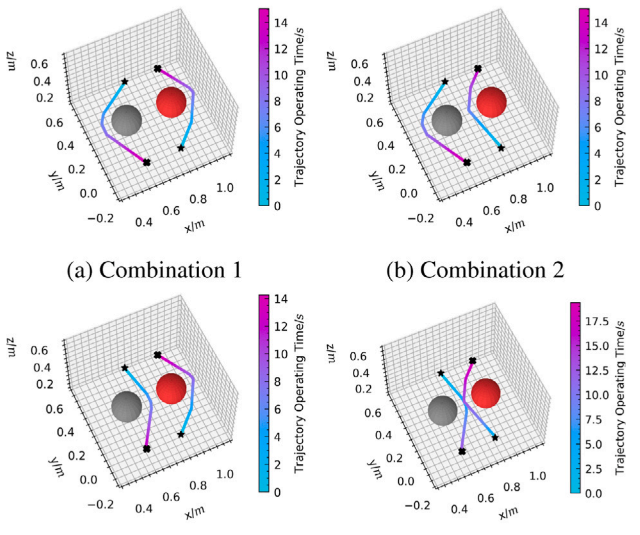
Table 1
Static Exp. 1, comparison of robot motion time, algorithm solving time among three DCS configurations in the same robot task.

Configuration	Candi No.	Robot motion time/s	Compile time/s	Solve time/s	Total solving time/s
w/o DPP w/o resample	1	14.06	0.1016	0.0057	0.4146
	2	13.36	0.0811	0.0042	
	3	17.65	0.0920	0.0046	
	4	20.35	0.0808	0.0042	
w/ DPP w/o resample	1	14.06	0.0984	0.0095	0.1674
	2	13.36	0.0004	0.0078	
	3	17.65	0.0004	0.0077	
	4	20.35	0.0004	0.0077	
w/ DPP w/ resample	1	14.02	0.0778	0.0078	0.1237
	2	13.29	0.0004	0.0064	
	3	16.33	0.0004	0.0061	
	4	20.12	0.0004	0.0063	



(a) before step 3   (b) after step 3

Fig. 8. Static Exp. 2, trajectories with time series.



(a) Combination 1   (b) Combination 2
(c) Combination 3   (d) Combination 4

Fig. 9. Final trajectory combinations in static Exp. 3.

collision in the middle channel, the algorithm is activated to optimize the timestamps of the right robot to avoid collision. Before step 3, combination 4 is the fastest, with $T_{L4} = 12.88$ s and $T_{R4} = 13.37$ s. After step 3's optimization, the right robot needs to wait for the left outside of the channel, making T_{R4} becomes 19.34 s, much slower than the $T_{R3} = 14.25$ s. As a final result, the combination 3 is chosen as the final solution. All statistics about this experiment can be found at Table 2.

4.2.2. Algorithm performance comparison

In this subsection, the proposed algorithm is compared with other robot trajectory planning algorithms in two tasks. The first task is single-robot trajectory generation with obstacles. The DCS algorithm is compared with the frequently-used sampling-based planner RRT-star [7], historical path based GMM/GMR method from Hu et al. [29], and the state-of-the-art optimization-based method GCS [20]. The test workspace is the same as mentioned in Section 4.1. All details including test case generation method, algorithm implementation procedures can be found in Appendix A. Three indicators are measured in this experiment: the generation success rate (success was defined as the algorithm generates the path with no collisions), average solving time, and path optimality. Average trajectory length ratio $\bar{R}_{len}$ is used, which is defined in formula (19) to indicate the path optimality. Where the N indicates number of test cases and the Len_others_i, Len_DCS_i represent the ith path length generated by the compared algorithm and the proposed DCS algorithm respectively.

$$\bar{R}_{len} = \frac{1}{N}\sum_{i=1}^{N}\frac{Len\_others_i}{Len\_DCS_i} \tag{19}$$

Several interesting points can be discovered from Table 3. In the task of single robot path generation, the DCS algorithm is ahead of the traditional planner RRT-star and the historical path based GMM/GMR in all aspects. This is because RRT-star and GMM/GMR methods are sampling-based and therefore their solutions do not necessarily fit the boundaries of the feasible domain in order to achieve the optimal. In addition, those algorithms use random search optimization methods, which converge slower than the convex optimization-based method DCS. In addition, since GCS's search domain in the workspace is a graph wrapped around the obstacles, which is larger than the search domain in the DCS algorithm, the GCS has higher solving success rate than the DCS. This experiment surprisingly shows that the proposed DCS algorithm's performance including solving speed and path optimality is comparable with the start-of-the-art planning algorithm GCS via a different modeling approach. More discussions between the proposed algorithm and GCS will be made in Section 5.

The second task is the dual-robot trajectory generation with obstacles. The algorithm is compared with a historical path-based method from Kyrarini et al. [2]. The test cases are same as task 1, and the implementation details of Kyrarini's method can be found in appendix as well. The measurement indicators in this experiment are generation success rate, algorithm solving time and time optimality. It is noteworthy that in the dual robot scenarios, robot motion finish time can better measure the optimality than the robot trajectory length, since two robots need temporal information to avoid collision with each other. Similar as formula (19) in task 1, average motion time ratio $\bar{R}_{time}$ is utilized to indicate the time optimality.

From the Table 4, it is observed that the DCS-MCA algorithm has higher generation success rate and time optimality than Kyrarini's method, while the DCS-MCA method is slower than Kyrarini's method. The complexity of Kyrarini's method is lower than the DCS-MCA because the obstacle avoidance strategy of that algorithm is to directly

Table 2
Statistics of four candidate combinations in experiment 3.

Combination	Do Joint Optim	L robot motion time/s	R robot old motion time/s	R robot new motion time/s
1	No	15.06	14.25	14.25
2	No	15.06	13.38	13.38
3	No	12.88	14.25	14.25
4	Yes	12.88	13.38	19.35

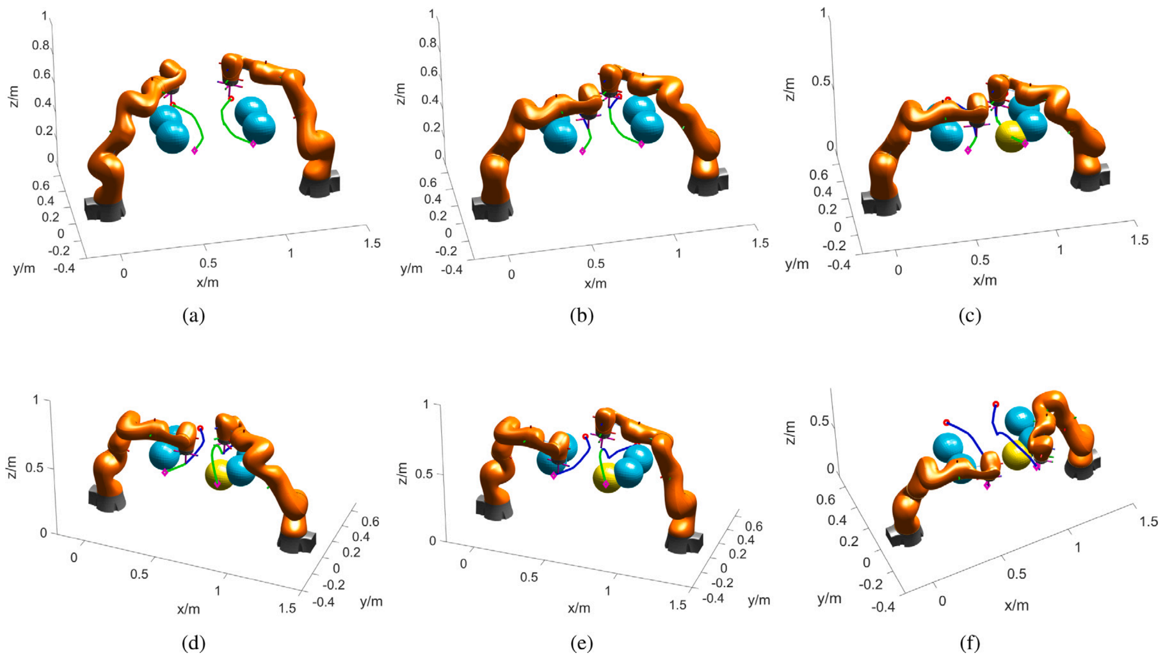


Fig. 10. Dynamic simulation with human motion (Blue spheres indicate static obstacles. The yellow sphere indicates a dynamic obstacle, simulating a human arm entering the scenario.)

Table 3
Algorithm efficiency comparison (task 1).

Method	Success rate	Solve time/s	$\bar{R}_{len}$
Ours	0.988	0.0367	1.000
RRT-star [7]	0.956	0.1990	1.129
GMM/GMR [29]	0.800	10.8624	1.199
GCS [20]	1.000	0.0347	1.013

Table 4
Algorithm efficiency comparison (task 2).

Method	Success rate	Solve time/s	$\bar{R}_{time}$
Ours	0.928	0.2935	1.000
Kyrarini [2]	0.747	0.0526	1.301

raise the z-axis height when the robots encounters an obstacle. Therefore, her method is faster than ours. However, if the obstacles locate at a relative high position, merely raising z-axis would cause the robot to go beyond the workspace and thus make the solution infeasible. To conclude, the DCS-MCA algorithm reaches better performance than the Kyrarini's method.

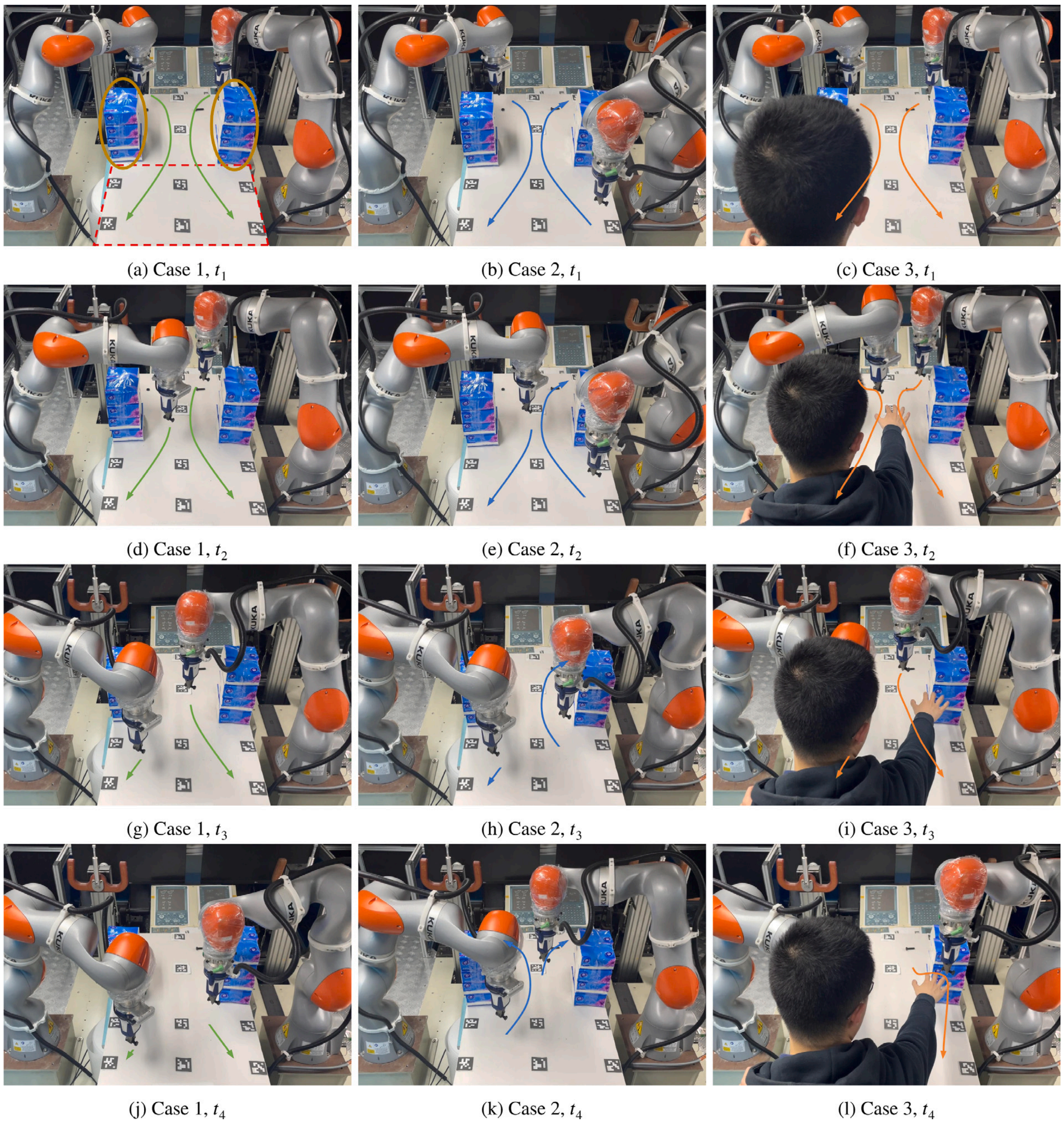
4.2.3. *Dynamic simulation with human motion*

In this section, a simulation experiment is given under dynamic human–robot collaboration scene. Fig. 10 shows the robots real-time planned trajectories and various obstacles during the process. Fig. 10(a) shows the task's initial configurations: the left and right robot are going to send parts to the assembly area, and left robot has higher priority. By now, the obstacles in this figures are static obstacles in the workspace. The two robots move along the planned path, passing the middle critical section one by one as shown in Fig. 10(b). In Fig. 10(c), the human arm enters the shared workspace, interfering with the right robot's path. The algorithm detects the situation and replans the trajectories based on new obstacle position. The result trajectories are displayed in Fig. 10(d). Then, the robots move along the new path till reaching the destination as shown in Figs. 10(e) and 10(f). It is worth noting that the algorithm could not plan a uniform height trajectory because of the obstruction of the human arm, so the newly generated trajectory took the approach of passing from above to reach the terminal point.

4.3. *Real scene experiment*

The real scene environment consists of a work table, a vision module, two collaborative robots and a human operator. The far end of the work table is the parts loading area, while the close end of the work table is the manual assembly area. Different static obstacles like monitors and assemble tools exist on the work table. The two robots are KUKA iiwa14 collaborative robot, including the robot body, control computer, smartpad and SCHUNK Co-act EGP-C 40-N-N-KTOE gripper. Two Intel RealSense D435i RGB-D cameras are used to obtain the views of parts and human in the workspace. The open source project MediaPipe [30] is used to perform human's arm positioning and tracking. Together with the input tasks, the proposed DCS-MCA planning algorithm plans the optimal trajectories and send it to the robots' control computer in real time through the TCP/IP protocol. A more detailed implementation description can be found in Appendix B.

A comprehensive schematic of the experiment is presented in Fig. 6. Fig. 11(a) depicts the actual test environment for the experiments, featuring two static obstacles (highlighted within golden circles) that necessitate the robot's navigation through a central channel. The pick up area is situated at the far-end of the work table, and the assemble


(a) Case 1, t_1 (b) Case 2, t_1 (c) Case 3, t_1

(d) Case 1, t_2 (e) Case 2, t_2 (f) Case 3, t_2

(g) Case 1, t_3 (h) Case 2, t_3 (i) Case 3, t_3

(j) Case 1, t_4 (k) Case 2, t_4 (l) Case 3, t_4

Fig. 11. Three real scene experiments (in columns) with the DCS-MCA algorithm.

area is at the close end (in red dashed box of Fig. 11(a)). For simplicity, the left robot's task is assumed to have higher priority than the right robot in all three cases.

For the first case, both robots commence within the loading zone, ready to transport the parts to the assembly area. The tentative path are drawn in green arrows in Fig. 11(a). As evidenced in Figs. 11(d) and 11(g), the left robot advances through the central channel first, followed by the right robot. Upon clearing the critical passage, each robot proceeds to its designated endpoint. The second scenario involves the robots moving in opposite directions, as shown in Fig. 11(b). In Figs. 11(e) and 11(h), the low priority robot (right) waits outside of the channel, letting the left to go through first. Fig. 11(k) illustrates that after the left robot reaching its destination, new trajectories are planned, and it returns back immediately. The final scenario examines the DCS-MCA algorithm's dynamic responsiveness within a human–robot collaboration context. Fig. 11(c) illustrates a human worker positioned in the close-end working area, with two robots tasked with delivering parts to this individual. As depicted in Fig. 11(f), the algorithm swiftly devises an alternative route when the human obstructs the lower spaces of the central channel, demonstrating the system's capability to adapt to dynamic obstacles. (For a comparative visual analysis, see Fig. 11(d)). Subsequently, Fig. 11(i) captures a moment after the left robot has navigated past the obstruction, at which point the human operator shifts their hand, impeding the right robot's newly calculated trajectory as shown in Fig. 11(f). This prompts the right robot to replan its trajectory to circumvent a potential collision, a process detailed in Fig. 11(l). (Refer to Fig. 11(j) for a comparative visual illustration).

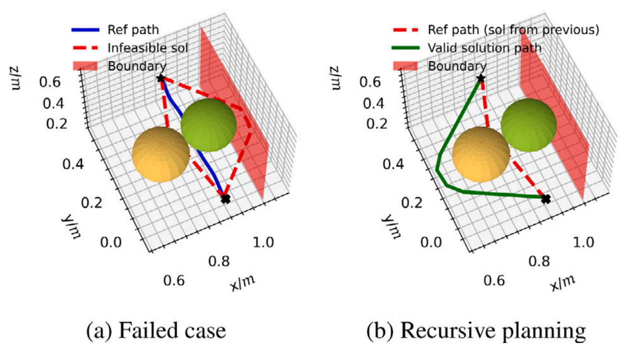


Fig. 12. Failed case visualization.

The above experiments test some typical cases in the collaboration process. The system is able to plan the collision-free dual robots trajectories autonomously. When the human collaborator interferes in, it can also immediately replan the new obstacle avoidance trajectories. The results demonstrate the ability of the proposed algorithm can avoid robot–robot collisions and human–robot collisions in the shared workspace production activities. For a comprehensive understanding of the system's capabilities and dynamics, please refer to the accompanying video documentation.

5. Discussion

Highlights of this paper include formulating the original problem into a batch of DPP problems while maintaining the large search domain. Benefit from the mature convex optimization solvers, the DCS methods can generate the result fast. The algorithm efficiency comparison experiments reveals that for the task of generating single collision-free path, the solution speed and solution optimality of the DCS algorithm are competitive with several existing common trajectory planning algorithms. As for the overall dual-robots collaborative task, the proposed DCS-MCA algorithm tends to have better optimality than some existing methods.

The limitations and possible improvements of the DCS-MCA algorithm are as follows. First, although the nature of the convex problem guarantees the result is optimal under the given constraints, approximating the non-convex constraints to convex one may lose search domain and thus get stuck into the local minimum. In some cases, the algorithm even obtains an infeasible result. For example, Fig. 12(a) shows a failed cases example in the task 1 of Section 4.2.2. The feasible boundary is plotted in red plane and the reference path is plotted in blue solid line. In such situation, the algorithm plans the path on two sides (See two red dotted line in Fig. 12(a)). But on the left side, there is an additional obstacle in the space that make the new trajectory in this direction infeasible. And the trajectory on the right side will be exceeded the feasible boundary of the working area, which turn out to be infeasible as well. Therefore, the algorithm failed to plan the path. Additional strategies may be adopted to prevent the failure in those cases. (1) The algorithm can be called recursively, that the trajectories with collisions in the first call can be used as the reference path (the red dotted line in Fig. 12(b)) in the second call. The algorithm would generate a feasible path (shown in green solid line in Fig. 12(b)). (2) GCS [20] can be used as an alternative for such infeasible situations. GCS has a wider search space to obtain the near-global optimal solution under the obstacle constraints. It can find the optimal solution for single-robot. Later the step 3 of the DCS algorithm requires multiple solution trajectories from each robot, as merely two optimal single-robot trajectories are not conducive for dual-robot collaborative planning. Therefore, GCS can be only used as a backup solution to overcome infeasibility of the DCS algorithm. It cannot replace the single-robot planning step in the pipeline.

Second, the solving speed is not fast enough to meet the ISO 10218 [31] standard for collaborative robots. Although the DCS algorithm enables the solving process to be about 200 ms, the two-step spatial and temporal optimization are still time-consuming. In addition, to reduce the negative effects brought by the convex approximation, the proposed algorithm needs to consider a batch of possible solution candidates, which also slows down the algorithm. Unified one-step spatial and temporal convex optimization method may be developed in the future to accelerate the solving speed, making the algorithm capable in more dynamic and variant situations. Some obstacle motion predictor such as [32] can be applied to reduce the probability to replan the trajectory on the way. As a conclusion, the proposed method is currently applicable to some general moderate human–robot collaborative scenarios.

6. Conclusions

In this paper, an innovative convex-optimization based human-dual robots collaborative trajectory generating algorithm is proposed for industrial applications. The algorithm ensures collision avoidance property and time optimal property in the moderate dynamic environments. To formulate the naturally non-convex problem into a convex problem, the DCS algorithm optimizes the path and corresponding time series in two consecutive steps. In the first step DCS, a DPP problem was established to plan the collision-free path to the static obstacles (equipment, human arms). Some techniques are used to improve the generated path quality and solving speed. In the second step, another optimization problem is established to adjust the time series of the low priority robots. The best trajectory pairs are chosen from various candidates combinations obtained from the DCS. The overall pipeline of the method is called DCS-MCA. Simulation and real experiments show the effectiveness of the DCS method for single-robot collision avoidance trajectory planning and DCS-MCA method for dual-robot collaborative trajectory planning. The performance comparison experiments further show the advantages of the proposed method.

CRediT authorship contribution statement

Lai Wei: Writing – review & editing, Writing – original draft, Software, Investigation, Formal analysis, Conceptualization. **Yanzhe Wang:** Writing – review & editing, Writing – original draft, Software, Methodology. **Yibo Hu:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software. **Tin Lun Lam:** Writing – review & editing, Supervision, Project administration. **Yanding Wei:** Supervision, Funding acquisition.

Declaration of competing interest

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

Data availability

The authors do not have permission to share data.

Acknowledgments

The authors would like to express appreciation for the financial support provided by the National Key Research and Development Program of China (2019YFB1707505).

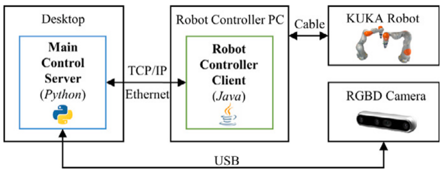


Fig. 13. The structure of our robot control framework.

Appendix A. Details in algorithm efficiency comparison

In this appendix, the test case generation method and implementation details of four compared algorithms in Section 4.2.2 are reported.

Test case generation. First, four positions in the pick-up area and four positions in the assemble area are chosen. For each test case, one point from pick-up area and one from assemble area are randomly sampled as the start and target, with 50% probability to swap the moving direction. Also, the priority sampled from Uniform[1, 2] is assigned to each case. Then, the obstacles are generated. In each test case, 1, 2, 3 obstacles are generated with 40%, 40%, 20% probability respectively. Each obstacle is randomly distributed in the workspace, and radius is sampled from Uniform[$0.04m$, $0.19m$]. Finally, 80 tasks are collected as the dual-robot test set. Meanwhile, each dual-robot test case is split into two single-robot test case, forming 160 cases for single-robot path generation.

Implementation details of RRT-star. The RRT-star algorithm [7] is implemented by [33]. The hyper-parameter settings are listed below: The search space size is a cube centered in the workspace with side length 2 m. The steering step is 0.04 m. The maximum number of samples before timeout is 3072. The number of nearby branches to rewire is 32. The probability of checking the end of the connection is set to 0.1.

Implementation details of GMM/GMR. The GMM/GMR algorithm in [29] is implemented by the authors. The Gaussian Mixture Model is implemented by [34]. The FFO algorithm is implemented in this algorithm. $T = 10$ initial trajectories are generated for each historical path with Gaussian(0, 0.02) noise. The GMM has $num\_gmm = 10$ Gaussian kernels. In the FFO algorithm, hyper-parameters $population = 10$, $max\_gen = 10$ are set. x_init, y_init are sampled from Gaussian(0, 0.005).

Implementation details of GCS. The GCS is implemented by [35]. The testing program uses the $LinearGCS$ class with the suggested Mosek [36] solver. In order to facilitate the construction of constraint space, obstacles are reduced to external cubes of spheres.

Implementation details of Kyrarini's method. The Kyrarini's method [2] is implemented by the authors. The Gaussian Mixture Model is modified from [34] to adapt the algorithm described in the paper. 32 human demonstrations are generated and augment them with Gaussian noise Gaussian(0, 0.02) to form total of $D = 160$ trajectories. The GMM has $num\_gmm = 12$ Gaussian kernels. In addition, in all experiments, the end-effector posture is set to be vertically downward.

Appendix B. Details in implementing real scene experiment

The overall control framework in the real scene experiment is shown in Fig. 13. The Main Control Server (implemented in Python) solves for trajectories and sends the step-by-step movement instructions directly to the Robot Controller Client by the interface designed by the authors. The Robot Controller Client directly execute the instructions to control the KUKA Robot with the built-in connectivity servo motion library. After each movement, the robot will also send the current position back to the main control server. This ensures that the robot moves exactly according to the expected trajectory on time.

To ensure the operation stability, we traverse the operability of the working configuration in the motion space through the whole trajectory, and a certain safe distance can be maintained to avoid singular configuration. In the experiments, the working configuration and workspace have high operability, which can avoid the risk of singular problems.

In addition, we adopt measures such as joint speed inspection and end effector deceleration to ensure kinematic correctness. If the joint speed exceeds the limit, the main control server calculates the path point of the next joint after deceleration based on time interval and joint velocity limit. Finally, the corresponding end effector pose is obtained based on forward kinematics, which is sent to the robot controller.

Appendix C. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.rcim.2024.102850.

References

[1] L. Wang, R. Gao, J. Váncza, J. Krüger, X.V. Wang, S. Makris, G. Chryssolouris, Symbiotic human-robot collaborative assembly, CIRP Ann. 68 (2) (2019) 701–726.
[2] M. Kyrarini, M.A. Haseeb, D. Ristić-Durrant, A. Gräser, Robot learning of industrial assembly task via human demonstrations, Auton. Robots 43 (2019) 239–257.
[3] S. Makris, E. Kampourakis, D. Andronas, On deformable object handling: Model-based motion planning for human-robot co-manipulation, CIRP Ann (2022).
[4] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, D.G. Vallejo, OBPRM: An obstacle-based PRM for 3D workspaces, 1998.
[5] R. Bohlin, L.E. Kavraki, Path planning using lazy PRM, in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol. 1, 2000, pp. 521–528.
[6] S.M. LaValle, Rapidly-exploring random trees : A new tool for path planning, Ann. Res. Rep. (1998).
[7] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res. 30 (2011) 846–894.
[8] J.J. Kuffner, S.M. 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.
[9] K. Kaltsoukalas, S. Makris, G. Chryssolouris, On generating the motion of industrial robot manipulators, Robot. Comput.-Integr. Manuf. 32 (2015) 65–71, http://dx.doi.org/10.1016/j.rcim.2014.10.002.
[10] X. Wang, J. ding Gao, X. Zhou, X. Gu, Path planning for the gantry welding robot system based on improved RRT*, Robot. Comput. Integr. Manuf. 85, 102643.
[11] K. Merckaert, B. Convens, M.M. Nicotra, B. Vanderborght, Real-time constraint-based planning and control of robotic manipulators for safe human-robot collaboration, Robot. Comput. Integr. Manuf. 87 (2024) 102711.
[12] A.H. Qureshi, Y. Miao, A. Simeonov, M.C. Yip, Motion planning networks: Bridging the gap between learning-based and classical motion planners, IEEE Trans. Robot. 37 (2019) 48–66.
[13] Y. Wang, L. Wei, K. Du, G. Liu, Q. Yang, Y. Wei, Q. Fang, An online collision-free trajectory generation algorithm for human-robot collaboration, Robot. Comput. Integr. Manuf. 80 (2023) 102475.
[14] M. Duguleana, F.G. Barbuceanu, A. Teirelbar, G. Mogan, Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning, Robot. Comput.-Integr. Manuf. 28 (2) (2012) 132–146, http://dx.doi.org/10.1016/j.rcim.2011.07.004.
[15] N.D. Ratliff, M. Zucker, J.A. Bagnell, S.S. Srinivasa, CHOMP: Gradient optimization techniques for efficient motion planning, in: 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 489–494.
[16] M. Kalakrishnan, S. Chitta, E.A. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 4569–4574.
[17] B. Açikmese, J.M. Carson, L. Blackmore, Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem, IEEE Trans. Control Syst. Technol. 21 (2013) 2104–2113.
[18] X. Liu, P. Lu, Solving nonconvex optimal control problems by convex optimization, J. Guid. Control Dyn. 37 (2014) 750–765.
[19] J. Schulman, Y. Duan, J. Ho, A.X. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking, Int. J. Robot. Res. 33 (2014) 1251–1270.
[20] T. Marcucci, M.E. Petersen, D. von Wrangel, R. Tedrake, Motion planning around obstacles with convex optimization, 2022, ArXiv, arXiv:2205.04422.

[21] Z. Zhang, L. Zheng, Z. Chen, L. Kong, H.R. Karimi, Mutual-collision-avoidance scheme synthesized by neural networks for dual redundant robot manipulators executing cooperative tasks, IEEE Trans. Neural Netw. Learn. Syst. 32 (2021) 1052–1066.

[22] C.-C. Wong, S.-Y. Chien, H.-M. Feng, H. Aoyama, Motion planning for dual-arm robot based on soft actor-critic, IEEE Access 9 (2021) 26871–26885.

[23] Y.-S. Choi, D.-H. Kim, S. Hwang, H. Kim, N. Kim, C.-S. Han, Dual-arm robot motion planning for collision avoidance using B-spline curve, Int. J. Precis. Eng. Manuf. 18 (2017) 835–843.

[24] B. Mildenhall, P.P. Srinivasan, M. Tancik, J.T. Barron, R. Ramamoorthi, R. Ng, NeRF: Representing scenes as neural radiance fields for view synthesis, in: European Conference on Computer Vision, 2020.

[25] I.J. Schoenberg, Contributions to the problem of approximation of equidistant data by analytic functions. Part B. On the problem of osculatory interpolation. a second class of analytic approximation formulae, Quart. Appl. Math. 4 (1946) 112–141.

[26] S. Diamond, S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, J. Mach. Learn. Res. 17 (83) (2016) 1–5.

[27] A. Domahidi, E. Chu, S. Boyd, ECOS: An SOCP solver for embedded systems, in: European Control Conference, ECC, 2013, pp. 3071–3076.

[28] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nat. Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[29] Y. Hu, Y. Wang, K. Hu, W. Li, Adaptive obstacle avoidance in path planning of collaborative robots for dynamic manufacturing, J. Intell. Manuf. (2021).

[30] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M.G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, M. Grundmann, MediaPipe: A framework for building perception pipelines, 2019, ArXiv, arXiv:1906.08172.

[31] J. Saenz, N. Elkmann, O. Gibaru, P. Neto, Survey of methods for design of collaborative robotics applications- why safety is a barrier to more widespread robotics uptake, in: Proceedings of the 2018 4th International Conference on Mechatronics and Robotics Engineering, 2018.

[32] D. Vasquez, T. Fraichard, Motion prediction for moving objects: A statistical approach, in: IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, vol. 4, 2004, pp. 3931–3936, http://dx.doi.org/10.1109/ROBOT.2004.1308883.

[33] T.K. Sebastian Zanlongo, Rrt-algorithms, 2020, GitHub repository, GitHub, https://github.com/motion-planning/rrt-algorithms.

[34] B. Saduanov, Gaussian-mixture-models, 2019, GitHub repository, GitHub, https://github.com/BatyaGG/Gaussian-Mixture-Models.

[35] T. Marcucci, M.E. Petersen, D. von Wrangel, R. Tedrake, Gcs, 2022, GitHub repository, GitHub, https://github.com/mpetersen94/gcs.

[36] E.D. Andersen, K.D. Andersen, The Mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm, 2000.