Parallel Cartesian Planning in Dynamic Environments using Constrained Trajectory Planning

C. Park[1], F. Rabe[2], S. Sharma[2], C. Scheurer[2], U. E. Zimmermann[2], D. Manocha[1]

***Abstract*— We present a parallel Cartesian planning algorithm for redundant robot arms and manipulators. We precompute a roadmap, that takes into account static obstacles in the environment as well as singular configurations. At runtime, multiple paths in this roadmap are computed as initial trajectories for an optimization-based planner that tends to satisfy various constraints corresponding to demands on the trajectory, including end-effector constraints, collision-free, and non-singular. We highlight and compare the performance of our parallel planner using 7-DOF arms with other planning algorithms. To the best of our knowledge, this is the first approach that can compute smooth and collision-free trajectories in complex environments with dynamic obstacles.**

I. INTRODUCTION

High degree of freedom (DOF) robot systems and arms are widely used for different applications. These include use of industrial manipulators for manufacturing and assembly tasks. Over the last few decades, the advances in human-like robots (e.g. humanoids) have resulted in many new applications related to service robots, entertainment and search/rescue applications. Most of these human-like robots have very high DOFs (20-40) and this combinatorial complexity gives rise to many challenges with respect to realtime task and motion planning. Not only we need to deal with the high dimensionality of the configuration space, but also need to satisfy various constraints, including generating collision-free and smooth trajectories as well as satisfying kinematic and dynamic constraints.

Most humanoids consists of arms or manipulators with redundant DOFs (i.e. each arm has more than six DOF). Moreover, these arms are used to perform dexterous tasks, and many of such tasks are reduced to Cartesian planning, i.e. the end effector of the arm needs to follow a certain trajectory. This was evident in the recent DARPA DRC challenge where the robots had to perform many Cartesian tasks such as drilling a hole or rotating a valve. At the same time, the robot arm needs to avoid collisions with the static and dynamic obstacles in the scene (i.e. collision avoidance) and also avoid singular configurations for each arm (i.e. singularity avoidance). In addition, we need to ensure that the resulting trajectory in the configuration space is smooth and satisfies other constraints corresponding to limits on joint angles, velocities and accelerations.

[1]Chonhyon Park and Dinesh Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill. E-mail: {chpark, dm}@cs.unc.edu.

[2]Fabian Rabe, Shashank Sharma, Christian Scheurer, and Uwe E. Zimmermann are with KUKA Laboratories GmbH. E-mail: {Florian.Rabe, Shashank.Sharma, Christian.Scheurer, Uwe.Zimmermann}@kuka.com.

In this paper, we address the problem of Cartesian planning for such redundant arms or manipulators that can take into account various constraints highlighted above. We assume that there is a gripper or end-effector attached to the robot arm and the Cartesian trajectory planning problem is specified in terms of the position and the orientation of the end-effector. The underlying path planning problem is specified as a trajectory in the workspace that the end-effector needs to follow [1]. This constrains the position or orientation of the end-effector in the task space of the robot, which corresponds to the Cartesian space.

There is extensive work on path planning with Cartesian trajectory constraints. Many approaches are based on sampling-based planning [2], [3]. However, trajectories computed from sampling-based approaches can be jerky, and may need a post-processing step for trajectory smoothing [4]. Moreover, there are applications where the robots must work reliably in dynamic environments with humans and other moving objects (e.g. industrial parts). Therefore, it is important that the robots should be able to perform the tasks in a safe manner in dynamic environments. Current planning algorithms for dynamic environments are mainly limited to motion planning from an initial configuration to a goal configuration [5], [6], [7].

Along with the Cartesian trajectory constraint, it is also important for high-DOF manipulators to avoid kinematic singular configurations along the computed trajectory, which allows stable use of control approaches to handle errors during the planned trajectory execution.

Main Results: In this paper, we present a parallel planning algorithm to compute smooth, collision-free, and non-singular motions, while taking into account Cartesian trajectory constraints of the end-effector. Our work builds on an optimization-based framework that has been useful for trajectory computation for human-like robots with dynamic stability constraints [8]. We extend this framework to take into account different constraints for high DOF manipulators (described above) and integrate them with the Cartesian trajectory specifications for a given task. In order to perform reliable trajectory optimization, we precompute a roadmap that takes into account collision-free constraints (with static obstacles) and non-singular constraints. We use a parallel trajectory optimization algorithm that takes into account dynamic obstacles and optimizes the trajectory cost function by performing parallel search. We also present techniques for multiple path selection that increases the probability of computing an optimal solution satisfying all constraints.

We highlight the performance of our parallel algorithm

in challenging scenarios with static and dynamic obstacles, and show the benefits of parallelism in terms of improving the success rate. To the best of our knowledge, this is the first approach that can compute smooth and collision-free trajectories in complex environments with dynamic obstacles.

The rest of the paper is organized as follows. In Section II, we give a brief overview of prior work on planning with end-effector constraints and the trajectory optimization. We present an overview of our planning algorithm in Section III. We describe the details of the precomputation of trajectories and the parallel trajectory optimization in Section IV and V, respectively. We highlight our algorithm's performance in different scenarios in Section VI and highlight the benefit of parallel optimization in Section VII.

II. Related Work

In this section, we give a brief overview of prior work in the planning with Cartesian trajectory constraints and the trajectory optimization.

A. Motion Planning with End-effector Constraints

The problem of path planning with Cartesian constraints can be specified with a Cartesian trajectory that the end-effector must follow [1], [9].

There are algorithms that try to directly compute motions in the task space of the robot. These approaches use potential fields [10], A* search, cell decomposition [11], sampling-based planning [12], or a reachable volume [13] in the task space of the robot, which is represented using Cartesian coordinates.

However, most of the approaches compute motion trajectories in the configuration space [14], [15]. Inverse kinematics solvers are used to convert an end-effector pose to a corresponding configuration. For redundant robots, numerical solvers can be used to find a solution [16], while robot-specific closed-from solvers are used for improved performance [17].

Many techniques are based on Rapidly-exploring Random Trees (RRT) [3]. IKBiRRT [14] generates bi-directional trees from multiple goal configurations that satisfy the end-effector goal pose. Some approaches [15], [18], [19] use a projection from a configuration to the nearest configuration which satisfies the trajectory constraints, based on expanding the RRT tree. The performance of RRT is improved using adaptive sampling strategies [20] or parallel collision checking [21].

Recent planning frameworks use these algorithms to implement the fully-constrained [22] or semi-constrained [23] planning.

B. Trajectory Optimization

Optimization techniques can be used to compute a robot trajectory that is optimal under some specific metrics (e.g., smoothness or length) and that also satisfies various hard constraints (e.g., collision-free) that the robot should satisfy. These algorithms can be used as a post-process on computed collision-free trajectories. The simplest trajectory smoothing algorithms use the shortcut method to smoothen the motion trajectory. They optimize the paths between adjacent pairs of configurations along the computed trajectory, using local planning, to compute smooth paths [24], [4].

Many techniques based on numerical optimization have been proposed in the literature [25]. Some recent approaches, such as [26] and [27], use a numerical solver to directly compute a trajectory that satisfies all the constraints (e.g. collision-free, smoothness).

C. Our Approach

The sampling-based approaches [14], [15], [19] are able to search the configuration space for collision-free solutions. However, the computed trajectories from sampling-based approaches can be jerky and may not satisfy many other constraints. On the other hand, trajectory optimization-based approaches [10], [28], [29] compute smooth trajectories, but they provide no guarantees on finding a good solution and may only compute a local optima.

Our planning algorithm tries to take the advantages of both approaches. The precomputation step computes collision-free and non-singular solutions even in complex environments using random search and probabilistic completeness of sampling-based approaches. The parallel trajectory refinement step optimizes these solutions to efficiently compute a smooth trajectory which satisfies the end-effector constraints. Furthermore, our trajectory optimization algorithm can handle dynamic obstacles and can also be used to compute valid trajectories for human-like robots with dynamic stability constraints [8].

III. Planning Algorithm

In this section, we introduce the notation and terminology used in the rest of the paper and give an overview of our planning algorithm.

A. Assumptions and Notations

In this paper, we restrict ourselves to computing appropriate trajectories for high-DOF manipulators, though it can also be used for high-DOF human-like robots as well. For an articulated robot with n joints, each configuration of the robot is defined by the joint angles. The n-dimensional vector space defined by these parameters is used to define the configuration space $\mathcal{C}$ of the robot. We denote the subset of $\mathcal{C}$ which is collision-free as $\mathcal{C}_{free}$, and the other configurations belong to the $\mathcal{C}$-obstacle space, $\mathcal{C}_{obs}$. A pose of the end-effector is represented as a point in the end-effector coordinate frame, which corresponds to a $SE(3)$ Cartesian space, the six-dimensional space of rigid spatial transformations in the 3D workspace $\mathcal{W}$ of the robot. In our constrained planning approach, it is required that the end-effector follows a constraint trajectory $\mathbf{c}(t)$ in the task space frame T, which can be the workspace or the end-effector coordinate frame. $\mathbf{c}(t)$ is defined with the all six-dimensions of T, or with a lower-dimensional subspace of T. In this paper, we denote a point in $\mathcal{C}$ using uppercase letters such as $\mathbf{Q}$, and a point in $\mathcal{W}$ with the task coordinate frame T as

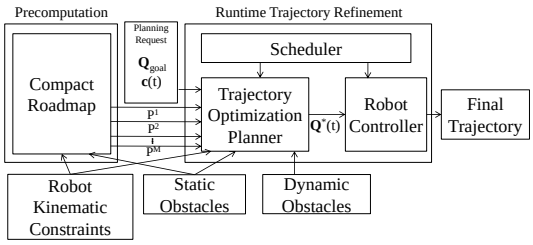


Fig. 1: An overview of our planning algorithm. The roadmap precomputation takes into account static obstacles and singularity constraints. For a given planning request, M paths $P^1, ..., P^M$ are computed using graph search. The computed paths are converted to trajectories, and then refined using trajectory optimization.

$\mathbf{q}$. Their trajectories, which are functions of time, are denoted as $\mathbf{Q}(t)$ and $\mathbf{q}(t)$, respectively.

We assume the robot has kinematic redundancy, which means $n = \dim(\mathcal{C}) > \dim(\mathcal{W}) = 6$. Here $\dim()$ represents the dimensionality of the space. The redundancy allows that there are multiple robot configurations that satisfy the Cartesian trajectory constraint.

Kinematic singularities of a manipulator correspond to the configurations when there is a change in the number of instantaneous degrees of freedom. In our approach, we mainly deal with *inverse singularities* [30], which cause the end-effector to lose one or more instantaneous DOFs. A robot configuration $\mathbf{Q}$ has *inverse singularity* if the rank of the $6\times n$ Jacobian matrix $\mathbf{J} = \frac{\partial \mathbf{q}}{\partial \mathbf{Q}}$ is less than 6. If a configuration is close to a singular configuration, the corresponding Jacobian matrix $\mathbf{J}$ becomes ill-conditioned, which is not a desired configuration. We define the near-singular space $\mathcal{C}_{singular^+}$, which is a subset of $\mathcal{C}$ that the distance to the closest singular configuration is smaller than a value ϵ. We can determine a configuration $\mathbf{Q}$ is near-singular if the smallest singular value is less than a threshold ϵ. i.e.,

$$\mathbf{J}(\mathbf{Q}) = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{1}$$

$$\mathbf{S}_{6,6} < \epsilon, \tag{2}$$

where $\mathbf{U}\mathbf{S}\mathbf{V}^T$ is a singular value decomposition of $\mathbf{J}(\mathbf{Q})$, and $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ are a $n \times 6$ orthonormal matrix, a 6×6 diagonal matrix, and a 6×6 matrix, respectively. $\mathbf{S}_{6,6}$ represents the value in the sixth row and the sixth column of $\mathbf{S}$, i.e., the smallest singular value.

Our goal is to find a continuous, collision-free, and non-singular trajectory $\mathbf{Q}^*(t)$ that the end-effector follows the given trajectory constraint $\mathbf{c}(t)$. $\mathbf{Q}^*(t)$ tends to be smooth, minimizes the joint acceleration along the trajectory and satisfies constraints corresponding to the joint position, velocity, and acceleration limits.

B. High-DOF Planning Algorithm: Overview

Fig. 1 gives an overview of our planning algorithm. The algorithm is decomposed into the roadmap precomputation step and the runtime trajectory refinement step.

In the precomputation step, we only take into account the static obstacles in the scene. The one-time precomputation of a roadmap is used to make the runtime planning efficient. In order to handle multiple queries, we use Probabilistic Roadmap (PRM) [2]-based approach to construct a roadmap graph $\mathbf{G}$ on the configuration space $\mathcal{C}$. However, the probabilistic approach of the original PRM generates a redundant dense graph, as many paths converge to the same solution with the trajectory optimization. Therefore, we compute a compact, and non-redundant roadmap $\mathbf{G}$ on the configuration space $\mathcal{C}$ using visibility checks to discard redundant nodes and edges, and using redundancy checks to discard redundant paths [31]. When we construct the roadmap $\mathbf{G}$, we only consider configurations that belong to $\mathcal{C}_{free}$ and do not belong to $\mathcal{C}_{singular^+}$. Furthermore, we also ensure that the edges of $\mathbf{G}$ satisfy these properties with respect to the free space and the singular space. This can be performed using discrete algorithms [32] with a certain resolution or continuous algorithms [33], depending on the required accuracy. Therefore, any path in the roadmap has no near-singular configuration and provides full dexterity or degrees-of-freedom motion for the end-effector along the path.

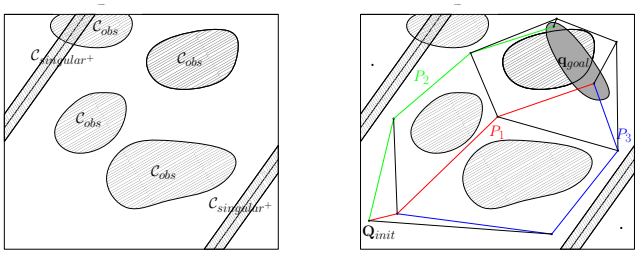
At runtime, we compute trajectories for constrained planning queries by refinement of selected initial trajectories from the precomputed roadmap $\mathbf{G}$. The selection of multiple non-redundant paths increases the coverage of the planning algorithm. Each planning request has a workspace goal region $\mathbf{q}_{goal}$, which can be a single end-effector pose or a set of poses, and the end-effector constraint $\mathbf{c}(t)$. The current configuration is used as the initial configuration $\mathbf{Q}_{init}$ to compute the trajectory. Also, there can be dynamic obstacles which are not considered in the precomputation step, but the robot can avoid collisions with its redundant DOFs while satisfying the end-effector Cartesian trajectory constraint.

IV. Roadmap Precomputation and Multiple Path Selection

In this section, we describe the roadmap precomputation and our novel multiple path selection algorithm.

A. Roadmap Precomputation

In the precomputation step, we build a roadmap graph $\mathbf{G}$ by adding nodes, which are configurations that lie in $\mathcal{C}$. We use nodes and edges which have no collisions, and we also want they are not near-singular configurations. Fig. 2(a) illustrates the configuration space. Given these criteria, we compute a roadmap $\mathbf{G}$ based on Path Deformation Roadmap algorithm [31]. The algorithm first computes a compact tree-like roadmap, then adds additional nodes and edges that correspond for paths which are difficult to be deformed from the existing paths in the tree-like roadmap. Fig. 2(b) shows an example of the generated compact roadmap which is collision-free and non-singular. The computed roadmap has the smallest number of nodes which are necessary to keep the coverage of the roadmap.



(a) A configuration space (b) Multiple path selection

Fig. 2: (a) Classification of the Configuration space. The obstacle space $\mathcal{C}_{obs}$ consists of disconnected regions, and the near-singular space $\mathcal{C}_{singular^+}$ is a region that the distance to the closest singular configuration is smaller than a value ϵ. (b) A roadmap graph built on Fig. 2(a) and multiple paths are shown. The nodes and edges on the graph are collision-free and correspond to non-singular configurations. For a path query from an initial configuration $\mathbf{Q}_{init}$ to the goal region $\mathbf{q}_{goal}$ (shown in dark gray region), different non-deformable paths P_1, P_2, and P_3 are shown in the graph.

Algorithm 1 $\{P^1,P^2,...,P^M\}$=MulPath($\mathbf{G}$, $\mathbf{Q}_{init}$, $\mathbf{q}_{goal}$)
: Extract M non-redundant paths from a roadmap graph $\mathbf{G}$

Input: roadmap graph $\mathbf{G}$=$\{\mathbf{V},\mathbf{E}\}$, the start configuration $\mathbf{Q}_{init}$, the goal region $\mathbf{q}_{goal}$
Output: M non-redundant paths P^1, ..., P^M that start from $\mathbf{Q}_{init}$ to a configuration $\mathbf{Q}_{goal}$ which corresponds to the goal region $\mathbf{q}_{goal}$

1: $\mathbf{E}_{init} = \emptyset$, $\mathbf{V}_{goal} = \emptyset$, $\mathbf{E}_{goal} = \emptyset$
2: **for** all node $n \in \mathbf{V}$ **do**
3: **if** $visibleNode(n, \mathbf{Q}_{init})$ **then**
4: $\mathbf{E}_{init}.insert(n, \mathbf{Q}_{init})$
5: **end if**
6: **end for**
7: $\mathbf{E}' = \mathbf{E} \cup \mathbf{E}_{init}$, $\mathbf{V}' = \mathbf{V} \cup \mathbf{Q}_{init}$, $ntry = 0$
8: **while** $ntry < ntry_{max}$ **do**
9: $\mathbf{Q}_{goal} = randomIK(\mathbf{q}_{goal})$
10: $ntry = ntry + 1$
11: // Do not add redundant nodes
12: **if** $\mathbf{V}_{goal}.hasVisibleNode(\mathbf{Q}_{goal})$ **then**
13: **continue**
14: **end if**
15: **for** all node $n \in \mathbf{V}'$ **do**
16: **if** $visibleNode(n, \mathbf{Q}_{goal})$ **then**
17: **if** $\mathbf{Q}_{goal} \notin \mathbf{V}_{goal}$ **then** $\mathbf{V}_{goal}.insert(\mathbf{Q}_{goal})$
18: **end if**
19: $\mathbf{E}_{goal}.insert(n, \mathbf{Q}_{goal})$
20: **end if**
21: **end for**
22: **end while**
23: $\mathbf{E}' = \mathbf{E} \cup \mathbf{E}_{goal}$, $\mathbf{V}' = \mathbf{V} \cup \mathbf{V}_{goal}$
24: $\{P^1, ..., P^M\} = shortestPaths(M, \mathbf{V}', \mathbf{E}', \mathbf{Q}_{init}, \mathbf{V}_{goal})$

B. Multiple Path Selection

When a roadmap is used to compute a path between the initial and goal positions, typically the shortest path in the roadmap graph between $\mathbf{Q}_{init}$ and $\mathbf{Q}_{goal}$ is returned as a single solution path. However, as we compute roadmap with collision-free and non-singular constraints which are invariant with multiple planning requests, this shortest path in the graph may not converge to a feasible solution in terms of trajectory optimization, due to the additional constraints. Moreover, the goal position is specified as a workspace goal region $\mathbf{q}_{goal}$, rather than a single configuration $\mathbf{Q}_{goal}$ in C-space. Therefore, we extract M different paths from the precomputed roadmap $\mathbf{G}$, which can cover different goal configurations. Fig. 2(b) illustrates the multiple path selection.

The resulting our novel algorithm for computing non-redundant multiple paths is given in Algorithm 1. We first add edges between $\mathbf{Q}_{init}$ and visible nodes in the roadmap $\mathbf{G}$ (line 4). For the given goal region $\mathbf{q}_{goal}$, we choose a random pose and compute a random IK solution $\mathbf{Q}_{goal}$ for that pose (line 9). Note that the mapping from a pose to a configuration is one to many for redundant robots. If there is a goal configuration in the graph which is visible from $\mathbf{Q}_{goal}$, $\mathbf{Q}_{goal}$ is redundant and therefore not added to the graph (line 12). If there are no other goal configurations visible from $\mathbf{Q}_{goal}$, $\mathbf{Q}_{goal}$ is added as a node and all possible edges from $\mathbf{Q}_{goal}$ to nodes in $\mathbf{G}$ are added. For the nodes and edges in $\mathbf{G}$ and $\mathbf{Q}_{init}$, $\mathbf{V}_{goal}$, $\mathbf{E}_{init}$ and $\mathbf{E}_{goal}$, we compute M shortest paths $\{P^1, ..., P^M\}$ from $\mathbf{Q}_{init}$ to any of the goal configurations using graph search algorithms [34]. These paths are used to generate multiple initial trajectories for the trajectory refinement computation as described in Section V.

V. Parallel Trajectory Refinement

In this section, we give the details of the runtime trajectory refinement, which include initial trajectory generation and the trajectory optimization framework and the optimization constraints.

A. Initial Trajectory Generation

At runtime, the planner computes M paths P^1, ..., P^M from the precomputed roadmap as described in IV-B and generates trajectories $\mathbf{Q}^1(t)$, ..., $\mathbf{Q}^M(t)$ from the paths. For each P^i, we discretize the path by adding N internal waypoints, based on uniform time intervals and distances along P^i. the trajectory $Q^i(t)$ is computed using the cubic interpolation of $N+2$ (including the two end points) waypoints. The interpolation step allows the trajectory optimization to start from a smooth trajectory. The trajectories are used as initial trajectories in the trajectory refinement step, and by optimizating M trajectories in parallel, we increase the probability of success in terms of finding a feasible or optimal solution to all the constraints.

B. Trajectory Optimization Framework

We use the covariant gradient descent approach for trajectory optimization [26], which preserves the smoothness

of the trajectory during optimization. The approach refines the positions of internal waypoints $\{\mathbf{Q}_1, ..., \mathbf{Q}_N\}$ of each trajectory $\mathbf{Q}^i(t)$ by minimizing the cost function to compute the optimal trajectory:

$$\mathbf{Q}^*(t) = \underset{\mathbf{Q}_1,...,\mathbf{Q}_N}{\arg\min} \sum_{k=1}^{N} (C(\mathbf{Q}_k) + \|\mathbf{Q}_{k-1} - 2\mathbf{Q}_k + \mathbf{Q}_{k+1}\|^2), \tag{3}$$

where the term $C(\mathbf{Q}_k)$ represents the cost function for a waypoint configuration $\mathbf{Q}_k$, and the second term $\|\mathbf{Q}_{k-1} - 2\mathbf{Q}_k + \mathbf{Q}_{k+1}\|^2$ represents the smoothness of the entire trajectory. The waypoint smoothness is computed based on the finite-difference accelerations on the joint trajectories. The two end-point configurations $\mathbf{Q}_{init}$ and $\mathbf{Q}_{goal}$ are used as $\mathbf{Q}_0$ and $\mathbf{Q}_{N+1}$, respectively in the smoothness computation.

As illustrated in Fig. 1, the trajectory refinement step has a scheduler, which repeatedly triggers planning a new trajectory $\mathbf{Q}^*(t)$ with the updated dynamic environment information.

C. Cartesian Planning Constraints

We formulate the waypoint cost function $C(\mathbf{Q}_k)$ of our Cartesian planning problem to include the costs for the collision constraint, the singularity constraint, and the end-effector Cartesian trajectory constraint for a waypoint $\mathbf{Q}_k$. These costs can be expressed as

$$\begin{aligned} C(\mathbf{Q}_k) =& w_{Collision} \cdot C_{Collision}(\mathbf{Q}_k) \\ &+ w_{Singularity} \cdot C_{Singularity}(\mathbf{Q}_k) \\ &+ w_{Cartesian} \cdot C_{Cartesian}(\mathbf{Q}_k), \end{aligned} \tag{4}$$

where w_i represents the weight of each cost. The weights can be optimized to find the best values.

1) Collision cost: $C_{Collision}(\mathbf{Q}_k)$ represents collision cost for both static and dynamic obstacles. A feasible solution should satisfy $C_{Collision}(\mathbf{Q}_k) = 0$ for all $\mathbf{Q}_k$, which means the trajectory has no collisions. Euclidean Distance Transform is used for static obstacles like the previous work [26], [7]. For dynamic obstacles, we use the squared sum of the penetration depths between the robot and the environment obstacles.
2) Singularity cost: $C_{Singularity}(\mathbf{Q}_k)$ represents the cost for near-singular configurations. It allows the robot to have the full dexterity of the end-effector. As we discussed in Section III, it can be evaluated using the singular values of the Jacobian matrix $\mathbf{J}(\mathbf{Q}_k)$. From the singular value decomposition of (1),

$$C_{Singularity}(\mathbf{Q}_k) = max(0, \frac{1}{\mathbf{S}_{6,6}} - \frac{1}{\epsilon})^2 \tag{5}$$

adds a penalty cost for near-singular configurations, i.e., $\mathbf{S}_{6,6} < \epsilon$.
3) Cartesian trajectory cost: $C_{Cartesian}(\mathbf{Q}_k)$ represents the cost from the violation of the Cartesian trajectory constraint, which is specified by the end-effector trajectory $\mathbf{c}(t)$. The error $\Delta\mathbf{x}$ is computed from the poses of $\mathbf{c}(t)$ and $\mathbf{Q}_k$ at the time of waypoint $\mathbf{q}_k$,

$$\Delta\mathbf{x}(\mathbf{Q}_k) = \mathbf{c}(t_k) - \mathbf{C}\mathbf{q}_k, \tag{6}$$

where t_k represents the time at $\mathbf{q}_k$, and $\mathbf{q}_k$ represents the end-effector pose that corresponds to $\mathbf{Q}_k$. $\mathbf{C}$ is a $d \times 6$ selection matrix, where $d = dim(\mathbf{c}(t))$ which selects only the constrained elements of $\mathbf{Q}_k$. In many problems, there is a tolerance vector $\mathbf{tol}$ defined in the same dimension with $\mathbf{c}(t)$. Therefore the cost function is defined as,

$$C_{Cartesian}(\mathbf{Q}_k) = \sum_d max(0, |\Delta\mathbf{x}(\mathbf{Q}_k)_d| - \mathbf{tol}_d)^2, \tag{7}$$

where $\Delta\mathbf{x}(\mathbf{Q}_k)_d$ and $\mathbf{tol}_d$ represents the d-th element of each vector.

The joint limit constraints can be formulated as an additional cost function. However, in our optimization formulation, we use the smooth projection method to remove the joint violations. We rescale the trajectory update of each iteration to ensure that each joint value in the trajectory is within the joint limits.

VI. Results

In this section, we describe the implementation of our planning algorithm and present the results for different scenarios. We have used our algorithm for KUKA LBR4+ robot (Fig. 7). The robot has redundant DOFs (7 joints), and each joint has minimum and maximum angle limits. We use MoveIt [22] for both the simulation environment and the interface to the real robot. We set the variables for planning: the number of internal waypoints in a trajectory $N = 100$, the singular value threshold $\epsilon = 10^{-3}$. The weights for the cost functions in (4) are set as $w_{Collision} = 100.0, w_{Singularity} = 1.0, w_{Cartesian} = 1.0$. We evaluate the performance of our planning algorithm on two sets of static benchmarks. Timing results were generated on a PC equipped with an Intel i7-2600 8-core CPU 3.4GHz. We use discretized collision and singularity checking at a fixed resolution for all experiments, but they can be replaced by continuous checking algorithms. For static benchmarks, the optimization terminates when one of the trajectories becomes feasible, which means the trajectory is collision-free, non-singular, and satisfies the end-effector constraints.

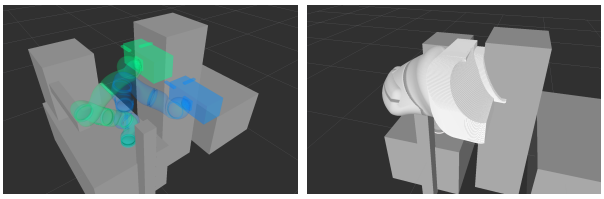
A. Benchmark 1 : Planning with Orientation Constraints

Our first benchmark (Fig. 3) corresponds to planning a trajectory with an orientation trajectory constraint of the end-effector. There are several static obstacles near the robot that restrict the pose of the robot. There is a tool attached to the robot, and the tool is only allowed to rotate along the Z-axis during the trajectory optimization. The X- and Y- axis of rotations of the end-effector should be less than the tolerance = 5°. The planning seems an easy problem, however the C-space has many narrow passages due to the robot joint limits and the static obstacle positions.

We compute the constrained trajectories for six planning queries with different start and goal pairs. Table I summarizes the planning results that each value is averaged with 10 trials. We measure the number of iterations, the planning time, and the success rate of the planner for the number

Benchmarks		M : Number of Trajectories			
		1	2	4	8
Benchmark 1	Iterations	1498.35	1354.71	1023.65	1040.28
	Planning time	23.80s	20.75s	15.48s	15.02s
	Success rate	70.00%	90.00%	90.00%	100%
Benchmark 2	Iterations	1635.95	1245.84	1141.41	943.73
	Planning time	25.84s	18.02s	15.54s	14.13s
	Success rate	80.00%	90.00%	100.00%	100.00%

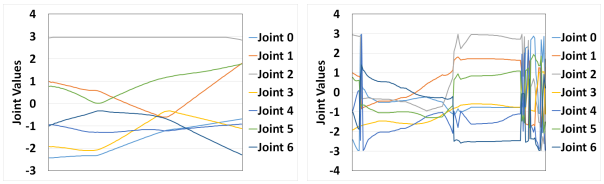
TABLE I: Planning results for our benchmarks. We measure the number of iterations for the trajectory optimization; planning time; success rate of the planning. We classify the planner as a success if it can find a solution in the maximum iteration limit (2000). As we increase M, the reliability of the planner improves with respect to various constraints.



(a) The benchmark environment and the start (green) and goal (blue) poses

(b) The computed trajectory of the robot

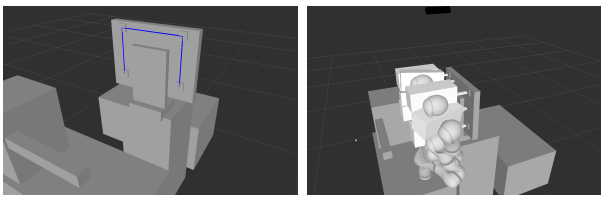
Fig. 3: Benchmark 1 computes a trajectory for end-effector constraints for X- and Y- axis rotations. (a) The start (green) and goal (blue) poses are shown. (b) The computed trajectory is shown.



(a) Plot of joint values computed using our approach

(b) Plot of joint values computed using Moveit and RRT*

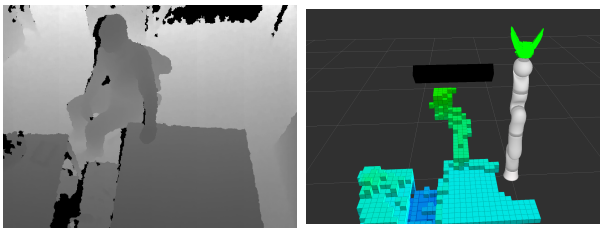
Fig. 4: Plots of joint values for the computed trajectory of Benchmark 1. (a) All joint values in the trajectory are smooth. (b) There are points that the joint values suddenly change.



(a) The end-effector position trajectory constraint

(b) The computed trajectory of the robot

Fig. 5: Benchmark 2 is following a trajectory defined for end-effector positions. (a) The environment and the constraint trajectory (blue path) are shown. (b) The computed trajectory is shown.



(a) Depth map images captured using Kinect for a human obstacle

(b) 3D octomap obstacles constructed from the depth map

Fig. 6: Dynamic environments: (a) We capture the depth map of a scene with a human arm approaching the arm using a Kinect. (b) 3D octomap is constructed from the depth-map, which is used as obstacle in the trajectory optimization.

of trajectories $M = 1$ to 8. We assume that a planner fails if the number of iterations reach the max iteration limit, set as 2000. It shows that increasing M reduces the planning time and increases the success rate of the planner. But when M becomes greater than 8, the maximum number of CPU cores, the planner can take more time. Because the parallel computations share the computational resources, increasing the number of trajectories beyond 8 may slow down the overall approach. Fig. 7 shows the execution of this benchmark on a real KUKA LBR4+ robot.

Comparison with Sample-based Planners: We also compute a solution to the same constrained planning problem with the constraint planner available as part of MoveIt. We use RRT and RRT* as the base planner for the constrained planning. RRT takes 274.399 seconds to compute a solution with the end-effector constraint. RRT* tends to spend the maximum planning time to improve the solution after a solution is found, and we set the maximum planning time of RRT* as 600 seconds. Fig. 4 shows the comparison of the the computed trajectories. RRT* computes shorter solution than RRT, but while our approach computes a smooth trajectory, the trajectories computed from constrained planning of Moveit framework have discontinuous points due to the redundant IK solutions.

B. Benchmark 2 : Planning with Position Constraints

Our second benchmark (Fig. 5) corresponds to planning a trajectory with a position that trajectory that the end-effector needs to follow. The orientation of the end-effector is not constrained. We compute constrained trajectories for three planning queries with different start and goal pairs. The planning results are shown in Table I. Like the benchmark 1, the planning result shows 100% success rate with 8 trajectories.

C. Dynamic Environments

In order to test the planner with dynamic obstacles, we captured depth map images of human obstacles in the scene using the Kinect (Fig. 6(a)) and construct the 3D octomap (see Fig. 6(b)). We use the octomap data with runtime trajectory optimization as dynamic obstacles. In the dynamic benchmark, the planner repeatedly updates the trajectory

Benchmarks		M : Number of Trajectories			
		1	2	4	8
Benchmark 1	Success rate	20.00%	50.00%	60.00%	80.00%
Benchmark 2	Success rate	30.00%	40.00%	80.00%	90.00%

TABLE II: Planning results for the benchmarks with dynamic obstacles. As we increase M, the success rate of the planner improves.

until the robot end-effector reaches the workspace goal region. We highlight the performance of the previous two benchmark scenes where a human moves his arms near the robot arm at a slow pace. As shown in Table II, the success rate of the planner increases as we increase the number of trajectories.

VII. Benefits of Parallelization

In this section, we highlight the benefits of parallel trajectory optimization.

A. Parallel Trajectory Optimization

As described in Section IV-B and V-A, our parallel trajectory optimization starts with the selection of M non-redundant paths from the roadmap. Then M initial trajectories are generated from the paths. We create M optimization threads that each thread optimizes an initial trajectory. If one of the threads finds a feasible solution, the optimization of all trajectories are interrupted and the computed best solution is returned. If the number of CPU cores is less or equal to M, the parallel optimization can be performed with the same time complexity of the single trajectory optimization, while it improves the success rate of the planning.

B. Improvement of the Success Rate of Planning

The runtime optimization problem in (3) has $n \cdot N$ degrees of freedom ($7 \cdot 100$ in our experiments). Extending the analysis in [35], we can show that the use of multiple non-redundant trajectories increases the success rate of planning using the following theorem.

Theorem 7.1: With a precomputed roadmap which has K different paths from $\mathbf{Q}_{init}$ to $\mathbf{Q}_{goal}$, the parallel optimization of M non-redundant initial trajectories will compute a feasible solution with the probability $(1 - \sum_{i_1}^{K}\sum_{i_2}^{K}...\sum_{i_M}^{K}\frac{|A_{i_1}||A_{i_2}|...|A_{i_M}|}{|S|^M})$*, where S is the entire search space, A_i is the neighborhood around a solution where the optimization converges to unfeasible local optima, and i are unique, i.e., $i_j \neq i_k$ if $j \neq k$. $|\cdot|$ is the measurement of the search space.*

Proof: In our planner, initial trajectories lie in the neighborhoods of different local optima and do not converge to the same solution, as they are chosen from the non-redundant roadmap. The probability that one of M trajectories lies in the neighborhood of a feasible solution is $1-$ (the probability that all M trajectories lie in the neighborhood of unfeasible solutions).

The probability that a trajectory lies in the neighborhood of K unfeasible local optima is $\sum_{i_1}^{K}\frac{|A_{i_1}|}{|S|}$, where A_{i_1} is the neighborhood of i_1-th local optimum. We choose a path different from the previous one for the second trajectory, and the probability that it is also lie in the neighborhood of unfeasible solutions is $\sum_{i_1}^{K}\sum_{i_2}^{K}\frac{|A_{i_1}||A_{i_2}|}{|S|^2}, i_1 \neq i_2$. Similarly, $(\sum_{i_1}^{K}\sum_{i_2}^{K}...\sum_{i_M}^{K}\frac{|A_{i_1}||A_{i_2}|...|A_{i_M}|}{|S|^M}, i_j \neq i_k \text{ if } j \neq k)$ measures the probability that M trajectories lie in the neighborhood of each unfeasible local optima $A_{i_1}, A_{i_2}, ..., A_{i_M}$. If the number of unfeasible local optima is less than M, the probability becomes 0 as one of the non-redundant trajectories should be in a neighborhood of feasible local optima. ■

It is not possible to measure the exact value of each $|A_i|$ in the configuration space $\mathcal{C}$, but it can be expected that $|A_i|$ will be smaller as the environment becomes more complex. Since $\frac{|A_i|}{|S|}$ is always less than 1, the increasing number of optimized trajectories M increases the probability that the planner computes a feasible solution.

VIII. Conclusions and Future Work

In this paper we present a parallel constrained planning algorithm for end-effector trajectory constraints. We use a two step approach : the precomputation step and the trajectory refinement step. In the precomputation step, we compute multiple trajectories that satisfy the collision-free and non-singular constraints from static obstacles. The trajectories are used as initial trajectories for the trajectory refinement step. Our planner optimizes the trajectories in the dynamic environment, using cost functions of the constraints. Therefore, our parallel planning algorithm tends to compute the trajectories that are smooth, collision-free, non-singular, and follow the given Cartesian trajectory of the end-effector. We validate our algorithm with several benchmark scenarios using a redundant KUKA manipulator. The results have been tested on the robot hardware (Fig. 7).

The current work has been used for high-DOF manipulators. Our next goal is to combine with human-like robots and demonstrate an integrated system that can be used for trajectory planning for humanoids and performing Cartesian tasks using these high-DOF arms. Furthermore, we would like to develop a parallel version of our Cartesian planner for many-core GPUs, which can considerably improve the responsiveness and performance of the planner.

IX. Acknowledgments

This research is supported in part by ARO Contract W911NF-14-1- 0437 and NSF award 1305286.

References

[1] Z. Guo and T. Hsia, "Joint trajectory generation for redundant robots in an environment with obstacles," *Journal of robotic systems*, vol. 10, no. 2, pp. 199–215, 1993.

[2] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

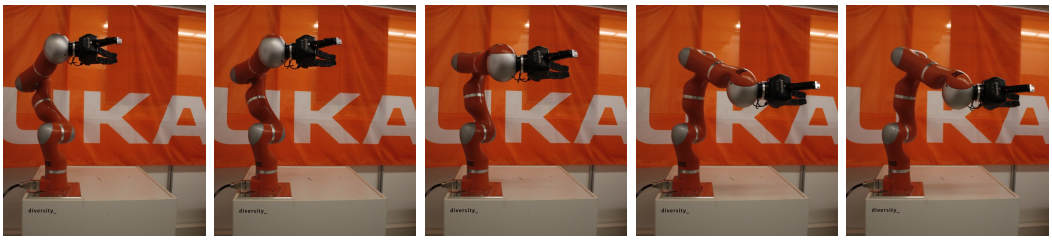


Fig. 7: Demonstration of our constrained planning algorithm in a static environment with KUKA LBR4+ robot.

[3] J. Kuffner and S. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2000, pp. 995 – 1001.

[4] J. Pan, L. Zhang, and D. Manocha, "Collision-free and curvature-continuous path smoothing in cluttered environments," *Robotics: Science and Systems VII*, vol. 17, p. 233, 2012.

[5] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2011, pp. 5628–5635.

[6] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, no. 1, pp. 35–48, 2012.

[7] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of International Conference on Automated Planning and Scheduling*, 2012.

[8] C. Park and D. Manocha, "Smooth and dynamically stable navigation of multiple human-like robots," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 497–513.

[9] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2005, pp. 2154–2160.

[10] A. Olabi, R. Béarée, O. Gibaru, and M. Damak, "Feedrate planning for machining with industrial six-axis robots," *Control Engineering Practice*, vol. 18, no. 5, pp. 471–482, 2010.

[11] C. Scheurer and U. Zimmermann, "Path planning method for palletizing tasks using workspace cell decomposition," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 1–4.

[12] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2006, pp. 1874–1879.

[13] T. McMahon, S. Thomas, and N. M. Amato, "Reachable volume rrt," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2977–2984.

[14] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2009, pp. 618–624.

[15] M. Stilman, "Task constrained motion planning in robot joint space," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 3074–3081.

[16] D. R. Baker and C. W. Wampler, "On the inverse kinematics of redundant manipulators," *The International Journal of Robotics Research*, vol. 7, no. 2, pp. 3–21, 1988.

[17] S. Sharma, G. K. Kraetzschmar, C. Scheurer, and R. Bischoff, "Unified Closed Form Inverse Kinematics for the KUKA youBot," in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, 2012, pp. 1–6.

[18] L. Jaillet and J. Porta, "Asymptotically-optimal path planning on manifolds," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[19] P. Kaiser, D. Berenson, N. Vahrenkamp, T. Asfour, R. Dillmann, and S. Srinivasa, "Constellation-An algorithm for finding robot configurations that satisfy multiple constraints," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2012, pp. 436–443.

[20] S. Rodriguez, S. Thomas, R. Pearce, and N. Amato, "Resampl: A region-sensitive adaptive motion planner," in *Algorithmic Foundation of Robotics VII*, S. Akella, N. Amato, W. Huang, and B. Mishra, Eds., 2008, pp. 285–300.

[21] S. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. Amato, "A scalable method for parallelizing sampling-based motion planning algorithms," in *International Conference on Robotics and Automation*, 2012, pp. 2529–2536.

[22] I. A. Sucan and S. Chitta, "Moveit!" *Online Available: http://moveit.ros.org*, 2013.

[23] S. Edwards, D. Solomon, J. Nicho, C. Lewis, P. Hvass, J. Zoss, and C. Flannigan, "Descartes : Cartesian path planner interface pipeline," *Online Available: https://github.com/ros-industrial-consortium/descartes*, 2014.

[24] P. Chen and Y. Hwang, "SANDROS: a dynamic graph search algorithm for motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, pp. 390–403, jun 1998.

[25] J. T. Betts, "Practical methods for optimal control and estimation using nonlinear programming," in *Advances in design and control*. Siam, 2001, vol. 3.

[26] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proceedings of International Conference on Robotics and Automation*, 2009, pp. 489–494.

[27] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[28] A. Gasparetto and V. Zanotto, "A technique for time-jerk optimal planning of robot trajectories," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 415–426, 2008.

[29] S. Alatartsev, A. Belov, M. Nykolaichuk, and F. Ortmeier, "Robot Trajectory Optimization for the Relaxed End-Effector Path," in *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2014.

[30] O. Bohigas, M. Manubens, and L. Ros, "Singularities of non-redundant manipulators: A short account and a method for their computation in the planar case," *Mechanism and Machine Theory*, vol. 68, pp. 1–17, 2013.

[31] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[32] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 171–180.

[33] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," in *Computer graphics forum*, vol. 21, no. 3. Wiley Online Library, 2002, pp. 279–287.

[34] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on computing*, vol. 28, no. 2, pp. 652–673, 1998.

[35] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using GPUs," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2013.