International Journal of Humanoid Robotics

HIGH-DOF ROBOTS IN DYNAMIC ENVIRONMENTS USING INCREMENTAL TRAJECTORY OPTIMIZATION

Chonhyon Park[1] and Jia Pan[2] and Dinesh Manocha[3]

Department of Computer Science, University of North Carolina at Chapel Hill, 201 S Columbia St., Chapel Hill, NC 27599, United States
chpark@cs.unc.edu[1], panj@cs.unc.edu[2], dm@cs.unc.edu[3]

Received 3rd July 2013
Revised 5th January 2014
Accepted 12th February 2014

We present a novel optimization-based motion planning algorithm for high degree-of-freedom (DOF) robots in dynamic environments. Our approach decomposes the high-dimensional motion planning problem into a sequence of low-dimensional sub-problems. We compute collision-free and smooth paths using optimization-based planning and trajectory perturbation for each sub-problem. The overall algorithm does not require a priori knowledge about global motion or trajectories of dynamic obstacles. Rather, we compute a conservative local bound on the position or trajectory of each obstacle over a short time and use the bound to incrementally compute a collision-free trajectory for the robot. The high-DOF robot is treated as a tightly coupled system, and we incrementally use constrained coordination to plan its motion. We highlight the performance of our planner in simulated environments on robots with tens of DOFs.

Keywords: Optimization-based Planning; Dynamic Environments; High-DOF.

1. Introduction

Motion planning algorithms are frequently used in robotics, CAD/CAM, and bio-informatics. Many task execution techniques repeatedly invoke motion planning algorithms for high-level planning. In some applications, motion planning and sub-task execution steps are performed in an interleaved manner. In this paper, we focus on the problem of motion planning for robots with high degrees-of-freedom (DOF), which include articulated robots with tens of joints. Many applications use articulated models for task planning, virtual prototyping or computer animation; since the models must perform different tasks and model various motions, they are represented using high-DOF articulated models.

Many efficient techniques have been proposed to compute collision-free paths in open spaces. Most of the earlier work on practical motion planning algorithms is based on randomized algorithms[1,2,3] and is mostly limited to static environments. It is important to develop techniques to enable a robot to safely navigate and perform tasks in the presence of moving obstacles. Furthermore, non-smooth and jerky paths

can cause actuator damages, and balancing constraints are important for humanoid robots. Randomized algorithms[4,5,6] can handle some constraints such as kinematic, contact, and balance constraints; however, it is relatively difficult to guarantee the quality of the trajectories (including torque or energy minimization, constraint handling, and smooth-path generation) computed by sample-based planners.

Optimization-based approaches pose the motion planning problem in a continuous setting and use optimization techniques to compute the trajectory.[7,8] Optimization-based approaches generate motion trajectories that can satisfy various constraints simultaneously (such as collision avoidance, smoothness, and dynamics constraints). Such trajectories are computed by posing the constraints in terms of appropriate cost functions. However, even the state-of-art applications of optimization-based motion planning for high-DOF robots[9,10] require a large amount of computation time, which makes them unsuitable for dynamic environments. Moreover, the convergence rate of the underlying numerical optimization techniques tends to decrease as the number of DOFs increases.

In order to overcome these challenges, we present a hierarchical optimization-based planner to compute smooth, collision-free trajectories for high-DOF robots in dynamic environments. Our formulation is based on the assumption that the optimal path lies in a lower-dimensional subspace, though the robot itself corresponds to a tightly coupled high-DOF system.[11] Our approach first decomposes a high-DOF robot into a hierarchical tree structure where each node represents one component of the robot (i.e., a set of joints and the related links). Based on this decomposition, we compute a trajectory for each component using an efficient replanning framework based on optimization techniques. In order to handle dynamic obstacles and perform realtime planning, our algorithm uses an incremental approach. We estimate the trajectory of the moving obstacles over a short time horizon using simple estimation techniques. Next, we compute a conservative bound on the position of the moving obstacles based on the predicted motion. We calculate a trajectory connecting the component's initial and goal configurations by solving an optimization problem that avoids collisions with the obstacles and satisfies smoothness and torque constraints. We incrementally compute the trajectory corresponding to each of the nodes that represents a sub-tree of the hierarchy. So that the robot can respond quickly to the changing environment, we interleave planning with task execution: that is, instead of solving the optimization problem completely, we assign a time budget for planning and interrupt the optimization solver when the time runs out. We repeat these steps until the robot reaches the goal position. The updated environment information is incorporated into the optimization formulation, which uses the sub-optimal result from the last step as the initial solution and tries to improve it incrementally within the given timing budget. We demonstrate the performance of our replanning algorithm in the ROS simulation environment, where 20-40 DOF robots are used to perform manipulation tasks. This approach can be extended to generate trajectories that are also dynamically stable.[12]

The rest of the paper is organized as follows. In Section 2, we survey related

work in motion planning. We give an overview of optimization-based planning and our hierarchical representation in Section 3. We present our algorithm for high-DOF robots in Section 4 and techniques to parallelize the computation on multi-core and many-core architectures in Section 5. We analyze the performance in Section 6 and highlight the performance in dynamic environments in Section 7.

2. Related Work

In this section, we give a brief overview of prior work on motion planning in dynamic environments, optimization-based motion planning, and hierarchical motion planning.

2.1. *Planning in Dynamic Environments*

Many approaches for motion planning in dynamic environments assume that the trajectories of moving objects are known *a priori*. Some algorithms discretize the continuous trajectory, and model dynamic obstacles as static obstacles within a short horizon.[13] Other techniques compute the velocity such that the robot can avoid a collision with moving obstacles within a short time step.[14,15] Some RRT variants can compute a trajectory in the state space ($\mathcal{C} \times T$, i.e., the Cartesian product of configuration space and time) directly;[16] other methods discretize the state space and use search techniques[17,18] or roadmap-based algorithms.[19]

Some planning algorithms for dynamic environments[17,19] assume that inertial constraints like acceleration and torque limit are insignificant for the robot; the assumption that the robot can stop and accelerate instantaneously is often not true for physical robots. These algorithms also attempt to plan the robot's path before execution begins, despite the fact that in many scenarios the computationally-expensive global path planning causes long delays in execution. These delays may cause collisions for robots operating in environments with fast-moving dynamic obstacles. Real-time replanning[16,20], which interleaves planning with execution, solves many of these problems. With real-time replanning, the robot need only compute partial or sub-optimal plans before execution to avoid collisions. Replanning allows for the use of different algorithms, including sample-based[16,20,21] or search-based planners,[22,23] as the underlying planners in the real-time replanning framework. Most replanning algorithms use fixed time steps when interleaving between planning and execution.[16] Some recent work[20] adaptively computes the interleaving timing step, which balances safety, responsiveness, and completeness over the entire system.

2.2. *Optimization-based Motion Planning*

The most widely-used method of path optimization is the so-called 'shortcut' heuristic, which selects pairs of configurations along a collision-free path and invokes a local planner to replace the intervening sub-path with a shorter one.[24,25] Another

approach uses Voronoi diagrams to compute collision-free paths.[26] Other techniques are based on elastic bands or elastic strips, which use a combination of mass-spring systems and gradient-based methods to compute minimum-energy paths.[27,28] All these methods use a collision-free path as an initial value for the optimization algorithm. Some recent approaches[29,8,30] do not need a collision-free path as an initial value. They directly encode the collision-free constraints and use an optimization-based solver to transform a naive initial guess into a trajectory suitable for robot execution. Although these planners do not guarantee planning completeness, they compute trajectories which optimize over a variety of criteria in many real-world planning scenarios efficiently.

The planning complexity of optimization-based motion planning algorithms tends to increase exponentially as the number of DOFs increases. Toussaint et al. use an approach to lower the planning DOF on a per-task basis.[31]

2.3. *Hierarchical Motion Planning*

The hierarchical mechanism decomposes a higher-dimensional planning problem into several lower-dimensional planning problems. This divide-and-conquer method can substantially reduce the complexity of the planning problem,[32] and the incompleteness of the resulting planning algorithms can be improved by greedy techniques based on back-tracing.[33]. Hierarchical mechanisms have been used to improve performance for articulated robots[32] or for multi-robot systems.[34] Different coordination schemes[35,36] have been proposed to guarantee that the decomposed planning finds solutions for the robots' whole bodies. Simple decomposition into lower- and upper-body has been used to plan the motion for human-like robots;[37] a more detailed decomposition has been used to accelerate whole-body planning for high-DOF robots using sampling-based planners.[38,39] Recently, hierarchical mechanisms have also been used to accelerate Markov Decision Process[40] and task planning.[41,42]

3. Overview

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

3.1. *Assumptions and Notations*

We use the symbol $\mathcal{C}$ to represent the configuration space of a robot, which includes $\mathcal{C}$-obstacles and the free space $\mathcal{C}_{free}$. Let the dimension of $\mathcal{C}$ be D. Each element in the configuration space, i.e., a configuration, is represented as a dim-D vector $\mathbf{q}$.

For a single planning step, suppose there are N_s static obstacles and N_d dynamic obstacles in the environment, and that all obstacles are rigid. The number of dynamic obstacles can change between the steps: new obstacles may be added into the robot's workspace, and other obstacles may be out of range during the planning interval. We assume that these obstacles are all rigid bodies. We denote

the positions of static obstacles as o_j^s, $j = 1, ..., N_s$, and the volumes of the obstacles in the workspace as $\mathcal{O}_j^s$. Since the positions of dynamic obstacles vary with time, we denote the trajectories of positions as functions of time t: $o_j^d(t)$, $j = 1, ..., N_d$, and the swept volumes of obstacles as $\mathcal{O}_j^d(t)$. i.e., $\mathcal{O}_j^d(t)$ is the volume that is swept by the boundary of j-th dynamic obstacle, as it moves along $o_j^d(t)$. For $\mathcal{O}_j^s$ and $\mathcal{O}_j^d(t)$, we denote the corresponding $\mathcal{C}$-obstacles in the configuration space as $\mathcal{CO}_j^s$ and $\mathcal{CO}_j^d(t)$, respectively.

In the ideal case, we assume that we have complete knowledge about the motion and trajectory $o_j^d(t)$ of each dynamic obstacle, i.e., we know $\mathcal{O}_j^d(t)$ and $\mathcal{CO}_j^d(t)$ exactly. In such cases, it is relatively simpler to compute a collision-free path. However, in real-world applications, we may only have local estimates of the trajectories of the dynamic obstacles (e.g. over a short time interval). Moreover, the recent position and velocity of obstacles computed from the sensors may not be accurate due to sensing errors. In order to guarantee the safety of the planning trajectory, we compute a conservative local bound on the trajectories of dynamic obstacles during planning. Given the time instance t_{cur}, the conservative bound at time $t > t_{cur}$ for an obstacle which moves along $o_j^d(t)$ bounds the shape corresponding to $\mathcal{O}_j^d(t)$, and is computed as:

$$\overline{O}_j^d(t) = c(1 + e_s \cdot t)O_j^d(t), \tag{1}$$

where e_s is the maximum allowed sensing error and c c is a predefined constant, which provides additional margins in terms of the conservative bound used to avoid collisions for obstacles. The computation is performed in the local coordinate system of the object, and the origin is placed at the center of the object. Therefore, as the sensing error increases, the conservative bounds of obstacles become larger. When an obstacle is moving with a constant velocity, it is guaranteed that the conservative bound on the trajectory includes the actual path traversed by the obstacle during the time period corresponding to $t > t_{cur}$, when $c = 1$. However, if an obstacle changes its velocity, we use a larger value of c in deriving our conservative bound, and it is valid for a shorter time interval. We can define the conservative bound for a time interval $I = [t_0, t_1]$ using Equation (1):

$$\overline{\mathcal{O}}_j^d(I) = \bigcup_{t \in I} \overline{\mathcal{O}}_j^d(t), \forall t \in I, t > t_{cur}. \tag{2}$$

Similarly, we can define conservative bounds in the configuration space, which are denoted as $\overline{\mathcal{CO}}_j^d(t)$ and $\overline{\mathcal{CO}}_j^d(I)$, respectively.

A configuration of a robot $\mathbf{q}$ is determined by all the actuated joints of the robot, as well as by the position and orientation of the robot in the workspace. The high-DOF robot is hierarchically decomposed into n different components $\{A^1, A^2, ..., A^n\}$. Accordingly, the configuration $\mathbf{q}$ can also be represented as the concatenation of the configuration $\mathbf{q}^i$ for each body component: i.e., $\mathbf{q} =$

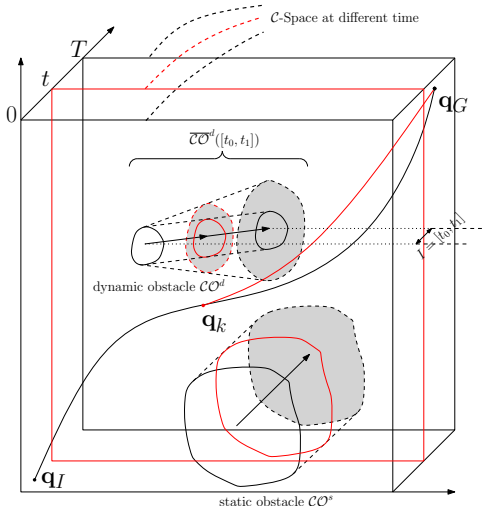


Fig. 1: Optimization-based motion planning for dynamic environments. We show how the configuration space changes over time: each plane slice represents the configuration space at time t. In the environment, there are two $\mathcal{C}$-obstacles: the static obstacle $\mathcal{CO}^s$ and the dynamic obstacle $\mathcal{CO}^d$. We need to plan a trajectory to avoid these obstacles. The trajectory starts at time 0, stops at time T, and is represented by a set of waypoints $\mathbf{q}_I$, $\mathbf{q}_1$, ..., $\mathbf{q}_k$, ..., $\mathbf{q}_N$, $\mathbf{q}_G$. Assuming that the trajectory is executed by the robot during the time interval $I = [t_0, t_1]$, we need only to consider the conservative bound $\overline{\mathcal{CO}}^d([t_0, t_1])$ for the dynamic obstacle during the time interval. The $\mathcal{C}$-obstacles shown in the red color correspond to the obstacles at time $t \in I$.

$[(\mathbf{q}^1)^T, (\mathbf{q}^2)^T, ..., (\mathbf{q}^n)^T]^T$, where $\mathbf{q}^i$ corresponds to the configuration of A^i. Moreover, $\mathbf{q}^i$ is determined by all A^i's actuated joints, including the joint through which A^i is connected to its parent component. $\mathbf{q}^1$ includes the position and orientation of A^1 component, which has the base link of the robot. We denote the trajectory with a fixed time duration T for a robot as $M(t)$, which is a discretized trajectory composed of $N+2$ waypoint configurations: $M(t) = \{\mathbf{q}_I, \mathbf{q}_1, ..., \mathbf{q}_N, \mathbf{q}_G\}$, where $\mathbf{q}_k$ is a trajectory waypoint at time $\frac{k}{N+1}T$. $\mathbf{q}_I$ and $\mathbf{q}_G$ represent the given initial and goal configurations, respectively. The trajectory for each component A^i is represented as $M^i(t)$, which also contains $N+2$ waypoints, i.e., $M^i(t) = \{\mathbf{q}_I^i, \mathbf{q}_1^i, ..., \mathbf{q}_N^i, \mathbf{q}_G^i\}$. We use symbol $\bar{\mathbf{q}}_k^i$ to represent the k-th waypoint corresponding to component A^i and all its previous components, i.e., $\bar{\mathbf{q}}_k^i = [(\mathbf{q}_k^1)^T, ..., (\mathbf{q}_k^{i-1})^T, (\mathbf{q}_k^i)^T]^T$. Similarly, $\bar{M}^i(t)$ corresponds to the trajectory of $\bar{\mathbf{q}}^i$. Fig. 1 illustrates the symbols used in our optimization-based planner.

3.2. *ITOMP : Incremental Trajectory Optimization*

The ITOMP planning algorithm[43] treats motion planning in dynamic environments as an optimization problem in the configuration space; we search for a smooth trajectory that minimizes the cost corresponding to collisions with moving objects and that takes into account some additional constraints (e.g., joint limit or acceleration limit). Similarly to previous work[29,8], our optimization problem finds the minimum cost trajectory; in other words, the N waypoint configurations are the optimization variables of the problem. It is formalized as:

$$\min_{\mathbf{q}_1,\ldots,\mathbf{q}_N} \sum_{k=1}^{N} (c_s(\mathbf{q}_k) + c_d(\mathbf{q}_k) + c_o(\mathbf{q}_k)) + \frac{1}{2}\|\mathbf{AQ}\|^2, \tag{3}$$

where $c_s(\cdot)$ is the obstacle cost for static objects, $c_d(\cdot)$ is the obstacle cost for moving objects (explained below) and $c_o(\cdot)$ is the cost for additional constraints, such as joint limit and torque limit. As $c_d(\cdot)$ changes along time due to the motion of the dynamic obstacles, we denote it as $c_d^t(\cdot)$ to highlight the dependency on time. $\mathbf{Q}$ is the serialized vector of a trajectory $M(t)$, which is defined as $[\mathbf{q}_I^T, \mathbf{q}_1^T, ..., \mathbf{q}_N^T, \mathbf{q}_G^T]^T$. $\mathbf{A}$ is a matrix that is used to represent the smoothness cost. We choose $\mathbf{A}$ such that $\|\mathbf{AQ}\|^2$ represents the sum of squared accelerations along the trajectory. Specifically, $\mathbf{A}$ is of the form

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix} \otimes \mathbf{I}_{D\times D}, \tag{4}$$

where $\otimes$ denotes the Kronecker tensor product and $\mathbf{I}_{D\times D}$ is a square identity matrix of size D. It follows that $\ddot{\mathbf{Q}} = \mathbf{AQ}$, where $\ddot{\mathbf{Q}}$ represents the second-order derivative of the trajectory $\mathbf{Q}$.

The solution to the optimization problem in Equation (3) corresponds to the optimal trajectory for the robot $\mathbf{Q}^* = \{\mathbf{q}_I^T, (\mathbf{q}_1^*)^T, ..., (\mathbf{q}_N^*)^T, \mathbf{q}_G^T\}^T$. However, notice that $\mathbf{Q}^*$ is guaranteed to be collision-free with dynamic obstacles only during a short time horizon. Because we only have a rough estimation based on the extrapolation of the motion of the moving objects, rather than an exact model of the moving objects' motion, the cost function $c_d^t(\cdot)$ is only valid within a short time interval.

In order to improve robot's responsiveness and safety, we interleave planning and execution. We assign a time budget Δ_k to the k-th step of replanning, which is also the maximum allowed time for execution of the planning result from the last step. The interleaving strategy is subject to one particular constraint: the trajectory currently being executed cannot be modified. Therefore, if the replanning result is sent to the robot for execution at time t, it is allowed to run for time Δ, and no

portion of the computed trajectory before $t + \Delta$ may be modified. In other words, the planner should start planning from $t + \Delta$.

3.3. *Obstacle Costs*

Similarly to prior work[29,8], we model the cost of static obstacles using the signed Euclidean Distance Transform (EDT) method. We start with a boolean voxel representation of the static environment, obtained either from a laser scanner or from a triangle mesh model. Next, the signed EDT, $d(\mathbf{x})$, for a 3D point $\mathbf{x}$ is computed throughout the voxel map. This provides information about the distance from $\mathbf{x}$ to the boundary of the closest static obstacle; the distance is negative, zero, or positive when $\mathbf{x}$ is inside, on the boundary or outside the obstacles, respectively. After the signed EDT is computed, the planning algorithm can efficiently check for collisions by table lookup in the voxel map. In order to compute the obstacle cost, we approximate the robot shape $\mathcal{B}$ by a set of overlapping spheres $b \in \mathcal{B}$. The static obstacle cost is as follows:

$$c_s(\mathbf{q}_k) = \sum_{b \in \mathcal{B}} \max(\epsilon + r_b - d(\mathbf{x}_b), 0) \|\dot{\mathbf{x}}_b\|, \tag{5}$$

where r_b is the radius of one sphere b, $\mathbf{x}_b$ is the center of sphere b computed from the kinematic model of the robot in configuration $\mathbf{q}_k$, and ϵ is a small safety margin between the robot and the obstacles.

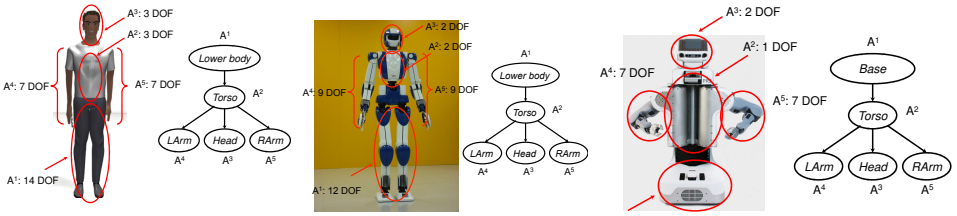
However, EDT computation cannot currently be applied to dynamic obstacles; EDT's recomputations, which take into account new positions of dynamic obstacles, is not fast enough on current CPUs for realtime performance. Instead, we perform geometric collision detection between the robot and moving obstacles and use the collision results to formalize the dynamic obstacle cost. Given a configuration $\mathbf{q}_k$ on the trajectory and the geometric representation of moving obstacles $\mathcal{O}_j^d$ at the corresponding time (i.e., $\frac{k}{N+1}T$), the obstacle cost corresponding to configuration $\mathbf{q}_k$ is given as:

$$c_d(\mathbf{q}_k) = \sum_{j=1}^{N_d} \texttt{is\_collide}(\overline{\mathcal{O}}_j^d(\frac{k}{N+1}T), \mathcal{B}), \tag{6}$$

where $\texttt{is\_collide}(\cdot,\cdot)$ returns one when there is a collision and zero otherwise. This function can be performed efficiently using object-space collision detection algorithms, such as OBBTree[44]. This obstacle cost function is only used during a short or local time interval, i.e. from replanning's start time t to its end time $t+\Delta$, since the predicted positions of dynamic obstacles become highly uncertain over long time horizons.

3.4. *Hierarchical Planning*

The optimal path tends to lie in a subspace which has a larger cost variation. For high-DOF robots shown in Fig. 2, we determine which degree-of-freedom has the



(a) A human-like model (34 DOFs)

(b) HRP-4 (34 DOFs)

(c) PR-2 (20 DOFs)

Fig. 2: An example of hierarchical decomposition for various robots. These hierarchical decompositions are used to divide a high-dimensional problem into a sequence of low-dimensional problems.

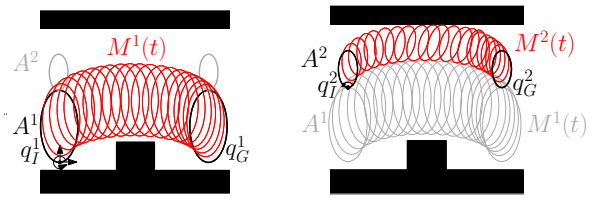
largest impact on the cost function when changed. Changes in some components influence the configuration of a large portion of the robot; for example, changing the pose of the legs affects the configuration of the whole upper body. Based on this observation, we decompose the robot body into a hierarchy of planning components. Fig. 2 shows a decomposition scheme for different robots. The high-DOF system is divided into several parts: a lower body (including legs and pelvis for human-like model, or a 3-DOF base for the PR2 robot), a torso, a head, a left arm and a right arm. For the same levels in the hierarchy, the physical volumes of the components are used to determine the order of the components.

We can incrementally plan the trajectory of a high-DOF robot based on this decomposition. First, we compute a trajectory $M^1(t)$ for the root component A^1. Then we fix the trajectory for A^1 and compute a trajectory for its child component A^2 by considering A^1 as a moving obstacle in the optimization formulation for A^2. However, there may be no feasible trajectory for A^2 because A^1 blocks it as an obstacle. In such cases, we first slightly modify the trajectory of A^1 based on workspace heuristics and search whether it is possible to compute a collision-free trajectory for A^2. If such local trajectory refinement does not result in a feasible solution, we perform back-tracing: we merge A^1 and A^2 into a larger component $A^{1,2}$ and then try to compute a collision-free path for this larger component using optimization-based planning. After the trajectory for A^2 is computed, we extend the approach in an incremental manner to compute a collision-free path for A^3, now treating A^1 and A^2 as moving obstacles. This process is repeated for all n components, and a trajectory for the overall robot is computed.

The hierarchical planner is implemented by decomposing Equation (3) into n optimization problems, one for each component A^i:

$$\min_{\mathbf{q}_1^i,\ldots,\mathbf{q}_N^i} \sum_{k=1}^{N} (c_s(\bar{\mathbf{q}}_k^i) + c_d(\bar{\mathbf{q}}_k^i) + c_o(\bar{\mathbf{q}}_k^i)) + \frac{1}{2}\|\bar{\mathbf{A}}^i \bar{\mathbf{Q}}^i\|^2, \tag{7}$$

where we compute the optimal waypoints $\mathbf{q}_k^i$ for components A^i while fixing the



(a) Planning of trajectory $M^1(t)$ for A^1 in stage 1. (b) Planning of trajectory $M^2(t)$ for A^2 in stage 2.

Fig. 3: Incremental trajectory planning. The robot model consists of $\{A^1$ (3 DOFs), A^2 (1 DOF)$\}$. (a) During stage 1, the algorithm computes trajectory $M^1(t)$ for A^1 while avoiding collisions between A^1 and the obstacle shown in the black region. (b) During stage 2 of the planning algorithm, the trajectory $M^2(t)$ for A^2 is computed while A^1 is assumed to move along the trajectory $M^1(t)$.

waypoints $\mathbf{q}_k^p$ for all the previous components $A^{1\leq p\leq i-1}$. $\bar{\mathbf{A}}^i$ is the smoothness matrix; it is similar to $\mathbf{A}$ in Equation (3), but it is resized to the length of $\bar{\mathbf{q}}_k^i$. $\bar{\mathbf{Q}}^i$ is defined as $\bar{\mathbf{Q}}^i = [(\bar{\mathbf{q}}_I^i)^T, (\bar{\mathbf{q}}_1^i)^T, ..., (\bar{\mathbf{q}}_N^i)^T, (\bar{\mathbf{q}}_G^i)^T]^T$.

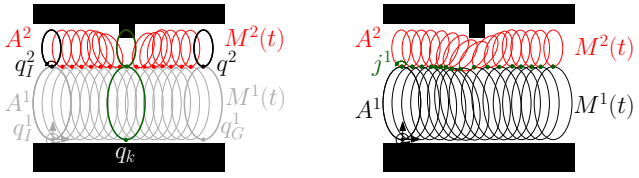
4. Hierarchical Optimization-based Planning

In this section, we present our hierarchical optimization-based planning algorithm. We first introduce our multi-stage trajectory optimization method. Next, we present the local refinement method, which uses the incremental coordination algorithm.

4.1. *Multi-stage Planning using Constrained Coordination*

Our algorithm traverses the entire hierarchy of the robot $\{A^1, A^2, ..., A^n\}$ sequentially in a breadth-first order using n planning stages. Stage i computes the trajectory for A^i and improves the trajectories of $\{A^1, ..., A^{i-1}\}$, which were computed during the prior stages. We use the incremental coordination approach[34,38] in our planning algorithm. During each planning stage, the algorithm computes the trajectory for a subset of robot components in order to compute the whole-body motion trajectory incrementally. According to our notation, we denote $\{M^1(t), M^2(t), ...M^i(t)\}$ as $\bar{M}^i(t)$. Given the input $\bar{M}^{i-1}(t)$, the planning algorithm during stage i computes $\bar{M}^i(t)$.

The trajectory for a new component is computed by treating the trajectories during the previous stages as constraints. In Fig. 3, the 2D robot has two components, A^1 and A^2. Each component has only one link. A^1 has 3 DOFs corresponding to the position and orientation of the robot in 2D space. A^2 has 1 DOF corresponding to the angle that connects A^1 and A^2. Therefore, the configuration vectors $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ have dimensions 3 and 1, respectively. The trajectory $M(t)$ is a sequence of N configurations at discretized time steps. During planning stage 1, the algorithm



(a) There is no collision-free configuration $\mathbf{q}_k^2$ for A^2 at time k if A^1 moves along the trajectory $M^1(t)$, which is computed in the prior stages.

(b) With local refinement, the planner finds a feasible solution. The trajectory $M^1(t)$ is updated to find a feasible trajectory $M^2(t)$.

Fig. 4: Planning with local refinement. By adjusting the configuration of the joint j^1 connecting A^1 and A^2, we can move A^1 away from the obstacle and leave more space for A^2 to pass through. As a result, the planner can compute a collision-free solution $\bar{M}^2(t) = \{M^1(t), M^2(t)\}$.

computes the trajectory M^1 for A^1, which connects the initial configuration $\mathbf{q}_I^1$ of A^1 with its goal configuration $\mathbf{q}_G^1$. During planning stage 2, the trajectory $M^2(t)$ for A^2 is computed, while A^1 is assumed to move along the trajectory $M^1(t)$.

4.2. *Trajectory Optimization with Local Refinement*

In this section we present the local refinement scheme used as part of trajectory optimization. In our incremental planning algorithm, the trajectory of a robot component is computed using an optimization formulation such that the trajectories of prior components are constrained to lie on the paths computed during previous stages. However, the optimization-based planner may fail to find a solution that satisfies all the constraints. Fig. 4(a) shows such an example for a simple 2D robot, which consists of two components, A^1 and A^2. The trajectory $M^1(t)$ for A^1, which is computed during planning stage 1, is collision-free. However, when computing a solution for A^2, A^1 is constrained to move along $M^1(t)$. This may result in no feasible solution for A^2 that avoids collisions with the environment. The back-tracing approach, which replans the trajectory with merged component $A^{1,2}$, can find a solution in such a case, but can be expensive for higher-dimensional problems. As shown in Fig. 4(b), we refine the trajectory $M^1(t)$ by adjusting the configuration of the joint connecting A^1 and A^2, then move A^1 away from the obstacles by a displacement $\mathbf{r}$. For the k-th waypoint, the vector $\mathbf{r}_k$ represents the position displacement of the first joint of the current component (component i for stage i), so the joint position is changed by $\mathbf{r}_k$ and the refined trajectory for A^1 is computed using inverse kinematics.

The trajectory optimization algorithm (Algorithm 1) uses a stochastic approach,[8] which computes the gradient of the cost for a trajectory waypoint by

Algorithm 1 Hierarchical Trajectory Optimization in Planning Stage i

Require: Robot components $\{A^1, ...A^i\}$
Trajectory $\bar{M}^{i-1}(t)$ which is computed in stage $i-1$
Start and goal configurations $\mathbf{q}_I^i$ and $\mathbf{q}_G^i$ for A^i
Planning time limit Δt_i
Ensure: Trajectory $\bar{M}_i^i(t)$
1: Generate an initial trajectory $M'(t)$ which connects $(\mathbf{p}_I^i) = [(\mathbf{q}_I^i)^T, (0,0,0)]^T$ and $(\mathbf{p}_G^i) = [(\mathbf{q}_G^i)^T, (0,0,0)]^T$.
2: $t_{start} \leftarrow getTime()$
3: **while** $getTime() - t_{start} < \Delta t_i$ **do**
4: Evaluate $\bar{\mathbf{q}}^{i-1}$ from $M'(t)$
5: Compute the trajectory cost of $M'(t)$
6: Compute the gradient of the cost
7: Update trajectory $M'(t)$ using the gradient
8: **end while**
9: Extract M^i from $M'(t)$
10: Compute $\bar{M}^{i-1}$ from $M'(t)$

evaluating the costs of randomly generated configuration points. Instead of optimizing $\mathbf{q}^i$ and $\mathbf{r}$ separately, we define a new term $\mathbf{p}^i$ (the concatenation of $\mathbf{q}^i$ and $\mathbf{r}$) and we optimize M' (the trajectory of N waypoints $\mathbf{p}^i$). At stage i, M' is initialized as a line connecting $\mathbf{p}_I^i = [(\mathbf{q}_I^i)^T, (0,0,0)]^T$ and $\mathbf{p}_G^i = [(\mathbf{q}_G^i)^T, (0,0,0)]^T$, in order to ensure that the resulting trajectory M will connect the initial and the goal configurations. For the given planning interval Δt_i the algorithm explores the configuration space of $\mathbf{p}^i$ to improve the trajectory M' using Equation (7). During each iteration, the algorithm computes $\bar{\mathbf{q}}^{i-1}$ from the value of $\mathbf{r}$. This new $\bar{\mathbf{q}}_k^{i-1}$ value is used for trajectory cost computation. When the planning time interval ends, M' is decomposed to two trajectories: M^i and the trajectory for $\mathbf{r}$. The refined trajectories of $\{A^1, ..., A^{n-1}\}$ are evaluated from the trajectory of $\mathbf{r}$.

4.3. *Dynamic Environment and Replanning*

The ITOMP algorithm makes no assumption about the global motion or trajectory of each moving obstacle. Instead, we predict the future position and the velocity of moving obstacles based on their recent positions, which may be generated from noisy sensors for physical robots. This prediction computation is used in combination with maximum error bounds to compute conservative bounds on the trajectories of the moving obstacles during the local time interval. Therefore, the planning result is guaranteed to be safe only during a short time period.

ITOMP takes advantage of this short time window of accuracy by interleaving the robot's planning and execution steps; this handles uncertainty from moving obstacles and provides high responsiveness. When the motion planner computes a

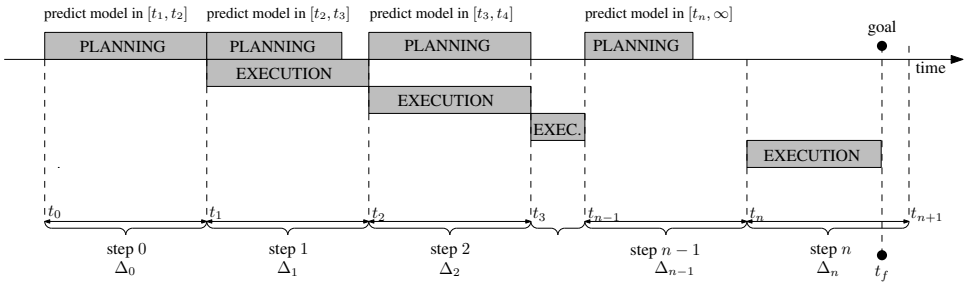


Fig. 5: Interleaving of planning and execution. The planner starts at time t_0. During the first planning time budget $[t_0, t_1]$, it plans a safe trajectory for the first execution interval $[t_1, t_2]$, which is also the next planning interval. In order to compute the safe trajectory, the planner needs to compute a conservative bound for each moving obstacle during $[t_1, t_2]$. The planner is interrupted at time t_1 and the robot starts the execution of the computed trajectory. Meanwhile, the algorithm starts the trajectory planning computation for the next interval $[t_2, t_3]$, after updating the bounds on the trajectory of the moving obstacles. This interleaving of planning and execution is repeated until the robot reaches the goal position.

new trajectory that is safe within a short horizon, the robot controller executes the trajectory. Meanwhile, the motion planner computes a safe trajectory for the next execution interval. This method of interleaving planning and execution phases is shown in Fig. 5. The i-th time step of short-horizon planning has a time budget $\Delta_i = t_{i+1} - t_i$, which is also the time budget for the current step of execution. During the i-th time step, the planner tries to generate a trajectory by solving the optimization problem in Equation (7). The trajectory should be valid during the next step of execution, i.e., during the time interval $[t_{i+1}, t_{i+2}]$.

Due to the limited time budget, the planner may only be able to compute a sub-optimal solution before it is interrupted. The sub-optimal solution may not be collision-free, or it may violate some other constraints during the next execution interval $[t_{i+1}, t_{i+2}]$. We assign higher weights to the obstacle costs related to the trajectory waypoints during the interval $[t_{i+1}, t_{i+2}]$, which biases the optimization solver to reduce the cost during the execution interval. If the optimization computation is not optimal, the planner can improve it incrementally during subsequent time intervals. In order to balance robot responsiveness with trajectory safety, the time budget for each step of short-horizon planning can be changed adaptively according to the quality of the resulting trajectory.[20]

Usually, the optimization can quickly converge to local optima, since during the i-th step planning we use the result of $(i-1)$-th step as the initial value. On the other hand, the optimization algorithm tends to compute a sub-optimal solution when the robot is in a narrow passage or near a region that has multiple minima in the configuration space.

5. Parallel Trajectory Optimization

Our algorithm is designed to take advantage of the computational capabilities of modern multi-core parallel processors. Nowadays, many of the robot systems are equipped with multi-core CPU processors (for example, the Quad-Core i7 Xeon Processors in PR2 robot[a]). These robot systems are expansible, and can use many-core accelerators, such as graphics processing units (GPUs). These many-core accelerators are massively parallel processors offering a very high peak performance (e.g. up to 3 TFLOP/s on NVIDIA Kepler GPU). Our goal is to exploit the computational capabilities of these commodity parallel processors for optimization-based planners and real-time replanning in dynamic scenes. In this section, we present our parallel algorithm[45] to solve the optimization problem highlighted in Equation (3).

We parallelize our algorithm in two ways. First, we parallelize the optimization of a single trajectory by parallelizing each step of optimization using multiple threads on a GPU (Fig. 6). Second, we parallelize the optimization of multiple trajectories by using different initial seed values. Since it is a randomized algorithm, the solver may converge to different local minima, and the running time of the solver also varies based on the initial seed values. In practice, such parallelization can improve the responsiveness and the quality of the resulting trajectory.

5.1. *Parallelized Replanning with Multiple Trajectories*

When the planner interleaves planning and execution as shown in Fig. 5, the parallel planner computes multiple trajectories in parallel. Within the time budget, multiple initial trajectories are refined by the optimization algorithm to generate multiple solutions that are sub-optimal and have different costs. Some of the solutions may not be collision-free during the execution interval, which could be due to the limited time budget or to the local optima corresponding to that particular solution. However, the parallelization using multiple trajectories increases the probability that a collision-free trajectory will be computed. It also usually yields a higher-quality solution.

5.2. *Highly Parallel Trajectory Optimization*

Because we parallelize the computation of multiple trajectories, our approach improves the responsiveness of the planner. We parallelize various aspects of the stochastic solver on the GPUs by using random noise vectors. The trajectory optimization process and the number of threads used during each step are illustrated in Fig. 6. The algorithm uses $(K \cdot M \cdot N \cdot D)$ threads in parallel, according to these steps, to exploit the computational power of GPUs. The algorithm starts with the generation of K initial trajectories. As defined in Section 3.1, each trajectory of a component A^i (which has D DOFs) is generated in the configuration space $\mathcal{C}$, which

[a]http://www.willowgarage.com/pages/pr2/specs

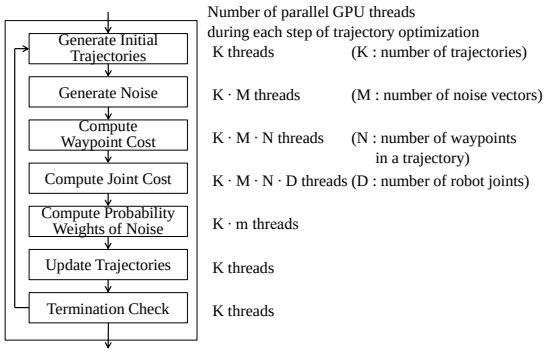


Fig. 6: The detailed breakdown of GPU trajectory optimization. It starts with the generation of K initial trajectories. From these initial trajectories, the algorithm iterates over stochastic optimization steps. The waypoint costs include collision cost, end effector orientation cost, etc. We also compute joint cost, which might include smoothness costs or the cost of computing the torque constraints. The current trajectory cost is repeatedly improved until the time budget runs out.

has N waypoints from an initial configuration $\mathbf{q}_I$ to a goal configuration $\mathbf{q}_G$. Then the algorithm generates M random noise vectors for all the N waypoints on the trajectory. These noise vectors are used to perform a stochastic update of the trajectory. Adding these M noise vectors to the current trajectory results in M noise trajectories. The total cost, including costs for static and dynamic obstacles, is computed for each waypoint in the noise trajectories. As described in the Section 3.3, the static obstacle cost is computed by precomputed signed EDT. Collision detection for the cost of dynamic obstacles is computed by the GPU collision detection algorithm[46]. Smoothness cost, computed by a matrix multiplication $\|\mathbf{AQ}\|^2$ for each joint, can be computed efficiently using the parallel capabilities of a GPU. When the costs of all noise trajectories are computed, the current trajectory is updated by moving it towards a direction which reduces the cost.

Fig. 7 shows the performance of the GPU-based parallel optimization algorithm. The GPU-based algorithm utilizes various cores to improve the performance of a single-trajectory computation, as shown in Fig. 6. Increasing the number of trajectories causes the system to share the resources for multiple trajectories. Overall, we observe that by simultaneously optimizing multiple trajectories, we obtain a higher throughput using GPUs.

6. Performance Analysis

In this section, we show that hierarchical decomposition can improve the performance of optimization-based planners like CHOMP[29], STOMP[8] and the ITOMP

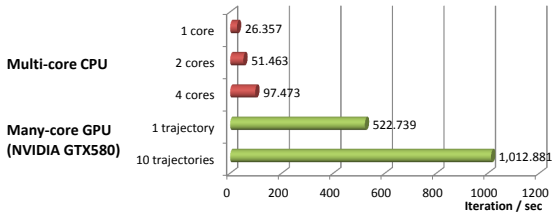


Fig. 7: Benefits of the parallel algorithm for the performance of the optimization algorithm. The graph shows the number of optimization iterations that can be performed per second. When multiple trajectories are used on a multicore CPU (by varying the number of cores), each core is used to compute one single trajectory. The number of iterations performed per second increases as a linear function of the number of cores. In the case of many-core GPU optimization, increasing the number of trajectories results in sharing of GPU resources among different trajectory computations, and the relationship is non-linear. Overall, we see a better utilization of GPU resources if we optimize a higher number of trajectories in parallel.

algorithm. All these planners solve an optimization problem expressed in the form of Equation (3) using steepest descent methods: CHOMP explicitly uses a numerical gradient, while STOMP and ITOMP use gradient information estimated using sampling. The convergence rate of steepest descent is related to the covariance matrix of the cost field based on the following theorem:

Theorem 1. *(Boyd and Vandenberghe)*[47] *Suppose we have a D-dimensional cost field $f(\mathbf{x})$, $\mathbf{x} \in \mathcal{R}^D$. For steepest descent search on the cost field starting with point $\mathbf{x}_0$, the error between k-th and $(k+1)$-th step is:*

$$\frac{f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*)}{f(\mathbf{x}_k) - f(\mathbf{x}^*)} \leq c = 1 - \frac{m}{M}, \tag{8}$$

where $0 < m \leq \lambda_D \leq \lambda_1 \leq M$. λ_1 and λ_D are the minimum and maximum eigenvalues of the cost field's covariance matrix $\nabla^2 f(\mathbf{x})$, respectively. $\mathbf{x}^*$ is the optimal solution point corresponding to the minimum cost of $f(\mathbf{x})$. In particular, we must have $f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \epsilon$ after at most $\frac{\log((f(\mathbf{x}_0) - f(\mathbf{x}^*)))/\epsilon)}{\log(1/c)}$ iterations.

In Equation (3), the dimension of the cost field is $D' = N \cdot D$, where N is the number of waypoints and D corresponds to the overall DOF. The time complexity to evaluate the cost for each point in the cost field is a constant proportional to the number of dynamic obstacles in the environment. According to the above theorem, we need $\frac{\log(\Delta/\epsilon)}{\log(1/c)}$ steps to converge to a local minima, where Δ is the error in the initial guess. As a result, the overall complexity for an optimization-based planner is

$$\mathcal{O}(\frac{\log(\Delta/\epsilon)}{\log(1/c)}N_d) = \mathcal{O}(1/\log(1/c)). \tag{9}$$

If c is very small, then we can approximate $1/\log(1/c) \approx \frac{M}{m}$, i.e., the complexity is decided by the ratio between the maximum and minimum variations along different directions of the cost field. Equation (9) implies that the optimization-based planners are most efficient on cost fields with similar variations along different directions; in other circumstances, the optimization procedure may instead follow a zigzagging curve and perform more iterations to converge to a local minima. Even in a 2D space, if an eigenvalue of $\nabla^2 f(\mathbf{x})$ is very small, the direction of the corresponding eigenvector has a weak correlation with the cost, in which case many areas of the cost field have local gradients which do not contribute toward the global solution. In this case, because of the large differences between the direction of the local gradient and the direction which leads toward the global solution, the optimization procedure may require many steps. On the other hand, in a high-dimensional space (say D-dimensions), if all D eigenvectors are related to the cost, the gradient descent methods can find a correct direction and can converge rapidly. Usually the time complexity of optimization-based planning algorithms grows as the DOFs of the robot increase,[48] because the configuration space of a high-DOF robot is more complex in the number of components and in the topology as the DOFs increase. This also increases the variance in eigenvalues of the cost field's covariance matrix; the eigenvectors therefore are only weakly correlated with cost in the cost fields of high-DOF robots.

We now use Equation (9) to explain the benefit of hierarchical decomposition. Suppose the eigenvalues for the cost field's covariance matrix are $\lambda_D \leq \lambda_{D-1} \leq ... \leq \lambda_2 \leq \lambda_1$, and that we decompose the robot into two components A^1 and A^2. First we assume that no back-tracing occurs, i.e., the trajectory for A^1 does not block the collision-free motion for A^2. Then the complexity for the decomposed planner is $\mathcal{O}(\frac{\lambda_1}{\lambda_m} + \frac{\lambda_{m+1}}{\lambda_D})$, which can be much smaller than the complexity of the original planner $\mathcal{O}(\frac{\lambda_1}{\lambda_D})$, if $\lambda_D \ll \lambda_{m+1} \approx \lambda_m \leq \lambda_1$. When back-tracing does occur, the decomposed planner may be less efficient than the original planner, as we are trying to compute the motion trajectory of a tightly-coupled system. However, such cases are not common in practice (also refer to Table 2 in Section 7). The components follow the hierarchy and volume order: components which influence the configuration of a large portion of the body are planned first, then components which influence smaller portions. A^1 affects the motion of a larger portion of the body than A^2, and therefore usually dominates the variation in the cost function. Let's take collision cost, which is measured by the intersected volume between the robot and the environment, as an example. Suppose A^1's cost value is a random variable C_1 within the range $[0, C_1^{max}]$ and A^2's cost value is a random variable C_2 within the range $[0, C_2^{max}]$, where 0 means collision-free, and C_1^{max} and C_2^{max} mean that the components are completely inside the obstacles. As A^1 is larger than A^2,

we have $C_1^{max} > C_2^{max}$. We also assume that C_1 and C_2 are symmetric random unimodal variables. Moreover, we have the following properties for the symmetric unimodal random variable:

Theorem 2. *For a symmetric unimodal random variable X defined on an interval $[a, b]$, there is*

$$\frac{(d-c)^2 \Pr[X \notin [c,d]]}{4} \leq \mathrm{Var}[X] \leq \frac{(b-a)^2}{12},$$

where $[c, d] \subseteq [a, b]$ is a subset of $[a, b]$.

As a result, if C_1^{max} is larger enough than C_2^{max}, we have

$$\mathrm{Var}[C_1] \geq \frac{(C_1^{max})^2 \Pr[X \notin [\frac{3C_1^{max}}{4}, \frac{C_1^{max}}{4}]]}{16} \geq \frac{(C_2^{max})^2}{12} \geq \mathrm{Var}[C_2];$$

that is, the variance in A^1's cost is larger than A^2's cost. In practice, the conclusion usually holds even when the assumption in Theorem 2 does not hold.

In other words, the decomposed planner first searches in the subspace with the larger cost variation and then in the subspace with the smaller variation. According to Vernaza and Lee[11], the optimal path usually lies in the subspace with the larger cost variation; therefore A^1's trajectory is usually optimal even though we do not consider A^2 during its computation. As a result, it is unlikely to block A^2, and the decomposed planner tends to be faster than the original planner.

7. Results

In this section, we highlight the performance of our hierarchical planning algorithm in static and dynamic environments. The experimental results of non-hierarchical planning can be found in previous works.[43,45] We have implemented our algorithm in the ROS simulator with both a human-like robot model and Willow Garage's PR2 robot model. We decompose the models into five components each (shown in Fig. 2). For the PR2 robot, we compute a trajectory of 20 DOFs, which are shown in Fig. 2(c). The human-like model has 34 DOFs, which are shown in Fig. 2(a). In this paper, we are focusing on efficient planning for high-DOF robots. The walking motions of human-like robots can be efficiently computed using motion generators[49,50]. Therefore, we compute a trajectory for the 3 DOFs lower body component using our motion planning algorithm; after that we use the motion generator to generate the walking motion, which is constrained by the trajectory of A^1. This reduces the DOFs for motion planning computations from 34 to 23. The constraints for legged robots, such as stability constraints or contact generation constraints, are discussed in Section 8.

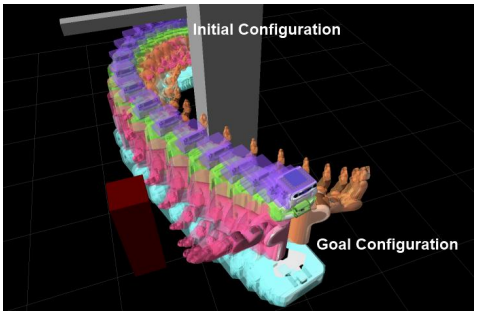
We highlight all the results of motion planning in different environments in Table 1. We compute the trajectories for the PR2 and the human-like robot in two static environments and two environments with dynamic obstacles. We compute the

		Non-hierarchical Planning				Hierarchical Planning			
		Iterations	Planning Time(s)	Cost	Success Rate	Iterations	Planning Time(s)	Cost	Success Rate
		Mean (Std. Dev.)				Mean (Std. Dev.)			
Human-like Robot	Static Environment 1	418.25 (344.90)	20.93 (16.24)	0.032 (0.011)	100/100	84.74 (18.00)	2.81 (0.50)	0.036 (0.000)	100/100
	Static Environment 2	461.26 (539.66)	30.78 (35.63)	0.017 (0.000)	100/100	54.02 (15.62)	2.21 (0.53)	0.025 (0.000)	100/100
	Dynamic Environment 1	13.99 (2.30)	1.71 (0.17)	0.058 (0.000)	89/100	18.89 (3.35)	1.33 (0.12)	0.101 (0.000)	95/100
	Dynamic Environment 2	20.15 (3.53)	2.80 (0.17)	0.163 (0.010)	76/100	26.48 (5.52)	1.79 (0.40)	0.201 (0.035)	93/100
PR2	Static Environment 1	102.06 (33.11)	8.20 (2.35)	0.033 (0.000)	100/100	90.75 (22.53)	5.11 (1.09)	0.032 (0.000)	100/100
	Static Environment 2	167.26 (239.65)	16.00 (22.42)	0.033 (0.000)	100/100	104.13 (73.08)	6.09 (4.11)	0.032 (0.000)	100/100
	Dynamic Environment 1	8.81 (3.90)	1.54 (0.42)	0.051 (0.000)	96/100	16.51 (12.12)	1.66 (0.66)	0.051 (0.004)	99/100
	Dynamic Environment 2	14.16 (3.67)	2.42 (0.51)	0.095 (0.002)	94/100	19.95 (6.40)	2.32 (0.49)	0.106 (0.006)	100/100

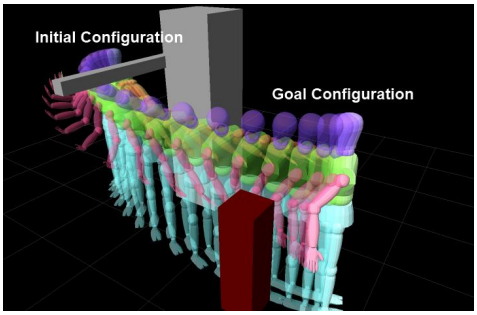
Table 1: The performance of our hierarchical planning algorithm is compared with the non-hierarchical ITOMP algorithm. We compute collision-free trajectories in static and dynamic environments. We measure the number of iterations used in the numerical optimization procedure; planning time to find the first collision-free solution; trajectory cost based on Equation (3); and the success rate of our planner, i.e., the total number of trials that found a collision-free trajectory. In the static scenes, our hierarchical planner results in up to 14X speedup over the non-hierarchical algorithm. The trajectory costs for the hierarchical and non-hierarchical algorithms are small (less than 0.1), which means the quality of the solution with the hierarchical planner is close to the trajectory computed by the non-hierarchical planner.

motion trajectory using our hierarchical planning algorithm and compare its performance with the motion trajectory computed using the non-hierarchical ITOMP algorithm. We measure the number of iterations in the optimization routines and the amount of planning time required to find the first collision-free solution. We also evaluate the quality of the computed trajectory by evaluating the cost functions and the success rate of the optimization-based planner. The results are shown in Table 1 and correspond to the means and standard deviations of 100 trials for each scenario. In most cases, hierarchical planning outperforms non-hierarchical planning. The only exception is the PR2 in dynamic environment benchmark in Table 1, where the planning time's mean and variance are larger for hierarchical planning than for non-hierarchical planning. This is because hierarchical planning has a higher success rate; non-hierarchical planning has many failed planning queries, whose time consumptions are not counted in the planning time statistics. Supplemental videos for the experiments can be viewed at http://gamma.cs.unc.edu/ITOMP/.

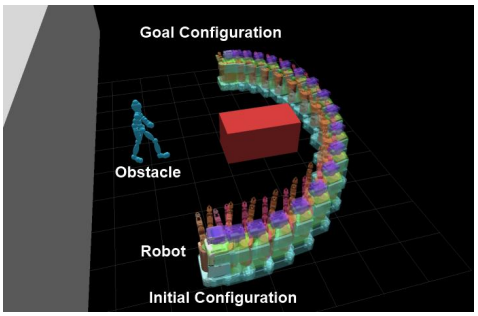
Fig. 8(a) and 8(b) show our first benchmark for a static environment. The environment has several static obstacles that prevent the initial trajectory from being collision-free; the robot must bend its two arms and its head to pass through a collision-free space, which is surrounded by obstacles. Using hierarchical planning, we incrementally compute the trajectory of the robot from components A^1 to A^5,



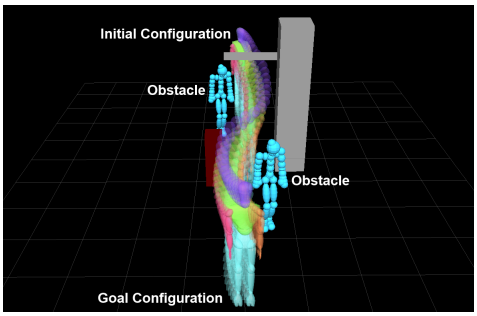
(a) The planned trajectory for PR2 robot in a static scene.



(b) The planned trajectory for a human-like model.



(c) The planned trajectory for PR2 robot in dynamic environment 1.



(d) The planned trajectory for a human-like model in dynamic environment 2.

Fig. 8: (a)(b) Hierarchical planning of a PR2 robot and a human-like robot in a static environment. The planned trajectory for different components is marked using different colors. (c)(d) Planning in dynamic environments. With the static obstacles, we also use human-like obstacles (shown in cyan) that follow a path generated from motion-capture data. The robot does not have any *a priori* information about the trajectory of this obstacle, which is designed to interrupt the robot's trajectory.

with no planning time limit. In Fig. 8, the trajectories of different components have different colors. In Table 2, we show the timings and the trajectory costs of each stage of hierarchical planning. Since the volume of the base of PR2 is much larger than the lower body of the human-like robot, the collision-free space is very tight for PR2. As a result, most of the planning time in this scenario is spent in the stage corresponding to A^1. Because PR2 is shorter than the human-like robot, the overhead obstacle has no effect on PR2; the components A^2 and A^3 are therefore collision-free on the computed trajectory of A^1, and the components require only a single iteration of the optimization algorithm for all trials. In the two other stages, which compute trajectories for the arm components A^4 and A^5, the planner runs tens of iterations to improve the trajectory to ensure that A^4 and A^5 have no col-

		Static Environment 1				Dynamic Environment 2			
	Stage	Iterations	Planning Time	Cost	Back -tracing	Iterations	Planning Time	Cost	Back -tracing
Human -like Robot	A^1 (3 DOFs)	7.33	0.25	0.009	0/100	6.98	0.37	0.051	0/100
	A^2 (3 DOFs)	15.18	0.53	0.009	0/100	7.12	0.39	0.125	2/100
	A^3 (3 DOFs)	24.10	0.65	0.000	0/100	1.52	0.07	0.000	0/100
	A^4 (7 DOFs)	18.81	0.69	0.012	0/100	4.26	0.39	0.021	1/100
	A^5 (7 DOFs)	19.32	0.69	0.005	0/100	6.60	0.57	0.004	2/100
	Overall Planning	84.74	2.81	0.036	0/100	26.48	1.79	0.201	5/100
PR2	A^1 (3 DOFs)	43.32	32.31	0.019	0/100	9.70	0.88	0.093	0/100
	A^2 (1 DOFs)	1.00	0.12	0.000	0/100	1.00	0.01	0.000	0/100
	A^3 (2 DOFs)	1.00	0.12	0.000	0/100	1.00	0.01	0.000	0/100
	A^4 (7 DOFs)	9.26	0.60	0.008	0/100	3.21	0.50	0.011	1/100
	A^5 (7 DOFs)	36.17	1.96	0.005	1/100	5.04	0.75	0.002	1/100
	Overall Planning	90.75	5.11	0.032	1/100	19.95	2.32	0.106	2/100

Table 2: We highlight the runtime performance of our planning algorithm in static and dynamic environments. We show the number of iterations; the planning time to find the first collision-free solution; the trajectory costs; and the number of trials in which back-tracings occur for each stage of our hierarchical planning algorithm, i.e., when a stage fails to find a collision-free trajectory for the corresponding component, the planner merges the component and its parent, then computes the trajectory of the merged component.

lisions. In the decomposition of the human-like model, each of the stages takes a similar amount of time and no one stage dominates the overall computation. This demonstrates that the decomposition used for the human-like robot divides the high-DOF planning problem into almost equal-sized low-dimensional sub-problems, which results in an overall performance improvement as compared to high-DOF planning. We observe that the speedup due to hierarchical planning is about 7X for the human-like model, with its equally decomposed sub-problems; it is about 1.6X for the PR2 model, which has a larger variance in the complexities of its sub-problems. In both cases, the trajectory cost corresponding to the optimization function with our hierarchical algorithm is close to the trajectory cost calculated by the non-hierarchical algorithm. This implies that the trajectories computed by both these algorithms are quite similar. In the second static environment benchmark, we use the same number of obstacles but the collision-free space is narrower than the first benchmark. This makes the motion planning more challenging, but still shows improvement using hierarchical planning: we observed 14X speedup for the human-like model and 2.6X for the PR2 model.

We also evaluated the performance of our algorithm in two dynamic scenes (Fig. 8(c) and Fig. 8(d)). We use a human-like obstacle that follows a path from motion-captured data, though the robots have no information about the global path of the obstacle. The path of the obstacles is designed to interrupt the path of the robot during execution. We set the replanning time step interval as 3 seconds; the planner fails if it cannot find a collision-free trajectory within that time interval. In

such dynamic scenes, the planner tends to improve trajectory computation during a given time step, but not for the overall duration. As a result, it is more important to measure the success rate of each planner rather than the overall planning time or the number of iterations. With the same replanning time step interval, our hierarchical planner has a higher success rate in dynamic environments than the non-hierarchical planner.

8. Conclusions, Limitations and Future Work

We present an optimization-based motion planning algorithm for high-DOF robots. Our algorithm decomposes the high-dimensional motion planning problem into a sequence of low-dimensional sub-problems and computes the solution for each sub-problem in an incremental manner. We use constrained coordination and local refinement to incrementally compute the motion. Our algorithm does not require *a priori* knowledge about global movement of moving obstacles; it tries to compute a trajectory that is collision-free and satisfies smoothness and dynamics constraints. In order to respond to unpredicted cases in dynamic scenes, we interleave planning optimization and task execution. This strategy can improve the responsiveness and safety of the robot. We highlight the performance on a 20 DOF PR-2 robot simulation and on a human-like robot with 34 DOFs, with which we also use a walking generator. In static environments, our algorithm offers up to 14X speedup while still generating smooth trajectories. In dynamic environments, we show that the algorithm can increase the success rate of the planning.

Our algorithm has some limitations. The performance of the hierarchical planner depends on the decomposition scheme and the motion trajectories computed for the previous stages. Since the underlying planner uses a stochastic optimization approach, the trajectories from the previous stages may not provide a good initial guess for local refinement. As a result, we cannot provide the completeness guarantee with our approach that it will always be able to compute a collision-free path within the given time interval. Currently, we need to assume the overall trajectory time and set the time for each waypoint on the trajectory before the optimization solver is used. In our implementation, we set the overall time as T and distribute the waypoints equally over the total time. These two requirements may cause failure of the planner in some situations. Moreover, our current implementation only takes into account two constraints: that the path be collision-free and that it meet smoothness requirements. In order to use such an approach for humanoids, we also need to take into account stability and control constraints. Recently, we have extended this approach to handle dynamic stability constraints (See Fig. 9).[?] Finally, we have only evaluated our planner in a simulated environment, and it does not take into account the various sensor errors and uncertainty that come with real data.

There are many avenues for future work. It would be interesting to extend our approach to compute whole-body trajectories for high-DOF human-like models.

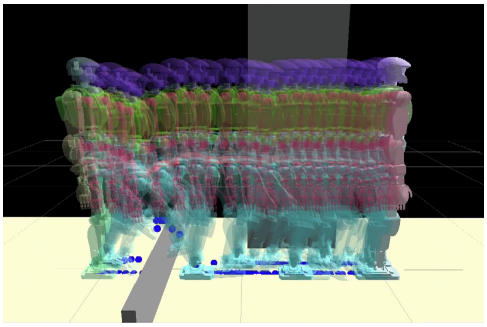


Fig. 9: Hierarchical planning of HRP-4 robot. Using stability constraints, the optimization-based planner computes physically plausible walking motion.

We would also like to evaluate its performance in more dynamic environments and ensure that the resulting motion of human-like robots is dynamically stable. Finally, we would like to use this approach for high-level planning and task execution, and evaluate its performance on physical robots.

9. Acknowledgments

This research is supported in part by ARO Contracts W911NF-10-1-0506 and NSF awards 1000579, 1117127, 1305286.

References

1. L. Kavraki, P. Svestka, J. C. Latombe and M. Overmars, *IEEE Transactions on Robotics and Automation* **12**, 566 (1996).
2. 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.
3. S. M. LaValle, *Planning Algorithms* (Cambridge University Press, 2006).
4. M. Stilman, *Robotics, IEEE Transactions on* **26**, 576 (2010).
5. D. Berenson, S. Srinivasa and J. Kuffner, *The International Journal of Robotics Research* **30**, 1435 (2011).
6. S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taıx and J.-P. Laumond, *The International Journal of Robotics Research (To appear in)* (2013).
7. M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell and S. S. Srinivasa, *International Journal of Robotics Research* (2012).
8. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor and S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2011), pp. 4569–4574.
9. A. El Khoury, F. Lamiraux and M. Taıx (2012).
10. S. Lengagne, J. Vaillant, E. Yoshida and A. Kheddar, *The International Journal of Robotics Research* (2013).
11. P. Vernaza and D. D. Lee, Learning dimensional descent for optimal motion planning in high-dimensional spaces, in *Proceedings of AAAI Conference on Artificial Intelligence*, (2011), pp. 1126–1132.

12. C. Park and D. Manocha, *Fast and Dynamically Stable Optimization-based Planning for High-DOF Robots*, tech. rep., Department of Computer Science, University of North Carolina at Chapel Hill (2013).
13. M. Likhachev and D. Ferguson, *International Journal of Robotics Research* **28**, 933 (August 2009).
14. P. Fiorini and Z. Shiller, *International Journal of Robotics Research* **17**, 760 (1998).
15. D. Wilkie, J. P. van den Berg and D. Manocha, Generalized velocity obstacles, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2009), pp. 5573–5578.
16. S. Petti and T. Fraichard, Safe motion planning in dynamic environments, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2005), pp. 2210–2215.
17. 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.
18. M. Phillips and M. Likhachev, Planning in domains with cost function dependent actions, in *Proceedings of AAAI Conference on Artificial Intelligence*, (2011).
19. J. van den Berg and M. Overmars, *IEEE Transactions on Robotics* **21**, 885 (Oct. 2005).
20. K. Hauser, *Autonomous Robots* **32**, 35 (2012).
21. D. Hsu, R. Kindel, J.-C. Latombe and S. Rock, *International Journal of Robotics Research* **21**, 233 (March 2002).
22. S. Koenig, C. Tovey and Y. Smirnov, *Artificial Intelligence* **147**, 253 (July 2003).
23. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz and S. Thrun, Anytime dynamic A*: An anytime, replanning algorithm, in *Proceedings of the International Conference on Automated Planning and Scheduling*, (2005).
24. P. Chen and Y. Hwang, *IEEE Transactions on Robotics and Automation* **14**, 390 (Jun 1998).
25. J. Pan, L. Zhang and D. Manocha, Collision-free and curvature-continuous path smoothing in cluttered environments, in *Proceedings of Robotics: Science and Systems*, (2011).
26. M. Garber and M. C. Lin, Constraint-based motion planning using voronoi diagrams, in *Algorithmic Foundations of Robotics V*, , Springer Tracts in Advanced Robotics Vol. 7 2004 pp. 541–558.
27. O. Brock and O. Khatib, *International Journal of Robotics Research* **21**, 1031 (2002).
28. S. Quinlan and O. Khatib, Elastic bands: connecting path planning and control, in *Proceedings of IEEE International Conference on Robotics and Automation*, (1993), pp. 802–807 vol.2.
29. 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.
30. A. Dragan, N. Ratliff and S. Srinivasa, Manipulation planning with goal sets using constrained trajectory optimization, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2011), pp. 4582–4588.
31. M. Toussaint, M. Gienger and C. Goerick, Optimization of sequential attractor-based movement for compact behaviour generation, in *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, (2007), pp. 122–129.
32. O. Brock and L. E. Kavraki, Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2001), pp. 1469–1474.

33. R. Alami, F. Robert, F. Ingrand and S. Suzuki, Multi-robot cooperation through incremental plan-merging, in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, (1995), pp. 2573–2579.
34. P. Isto and M. Saha, A slicing connection strategy for constructing PRMs in high-dimensional C-spaces, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2006), pp. 1249–1254.
35. M. Erdmann and T. Lozano-Pérez, On multiple moving objects, in *Proceedings of IEEE International Conference on Robotics and Automation*, (1986), pp. 1419–1424.
36. M. Saha and P. Isto, Multi-robot motion planning by incremental coordination, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2008), pp. 5960–5963.
37. G. Arechavaleta, C. Esteves and J.-P. Laumond, Planning fine motions for a digital factotum, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2004), pp. 822–827.
38. L. Zhang, J. Pan and D. Manocha, Motion planning of human-like robots using constrained coordination, in *IEEE-RAS International Conference on Humanoid Robots*, (2009), pp. 188–195.
39. J. Pan, L. Zhang, M. C. Lin and D. Manocha, *Computer Animation and Virual Worlds* **21**, 137 (2010).
40. J. L. Barry, L. P. Kaelbling and T. Lozano-Pérez, DetH*: Approximate hierarchical solution of large markov decision processes, in *Proceedings of the International Joint Conference on Artificial Intelligence*, (2011), pp. 1928–1935.
41. L. P. Kaelbling and T. Lozano-Pérez, Hierarchical task and motion planning in the now, in *Proceedings of IEEE International Conference on Robotics and Automation*, (2011), pp. 1470–1477.
42. L. P. Kaelbling and T. Lozano-Pérez, Pre-image backchaining in belief space for mobile manipulation, in *Proceedings of International Symposium on Robotics Research*, (2011).
43. 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).
44. S. Gottschalk, M. C. Lin and D. Manocha, Obbtree: a hierarchical structure for rapid interference detection, in *SIGGRAPH*, (1996), pp. 171–180.
45. 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).
46. J. Pan, C. Lauterbach and D. Manocha, g-Planner: Real-time motion planning and global navigation using GPUs, in *Proceedings of AAAI Conference on Artificial Intelligence*, (2010).
47. S. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, New York, NY, USA, 2004).
48. S. Basu, R. Pollack and M.-F. Roy, *Journal of the American Mathematical Society* **13**, 55 (2000).
49. Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi and K. Tanie, *Robotics and Automation, IEEE Transactions on* **17**, 280 (2001).
50. S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi and H. Hirukawa, Biped walking pattern generation by using preview control of zero-moment point, in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, (2003), pp. 1620–1626.