Research
Robotics—Article

Engineering 2015, 1(1): 46–57
DOI 10.15302/J-ENG-2015009

Efficient Configuration Space Construction and Optimization for Motion Planning

Jia Pan[1*] and Dinesh Manocha[2]

ABSTRACT The configuration space is a fundamental concept that is widely used in algorithmic robotics. Many applications in robotics, computer-aided design, and related areas can be reduced to computational problems in terms of configuration spaces. In this paper, we survey some of our recent work on solving two important challenges related to configuration spaces: ① how to efficiently compute an approximate representation of high-dimensional configuration spaces; and ② how to efficiently perform geometric proximity and motion planning queries in high-dimensional configuration spaces. We present new configuration space construction algorithms based on machine learning and geometric approximation techniques. These algorithms perform collision queries on many configuration samples. The collision query results are used to compute an approximate representation for the configuration space, which quickly converges to the exact configuration space. We also present parallel GPU-based algorithms to accelerate the performance of optimization and search computations in configuration spaces. In particular, we design efficient GPU-based parallel *k*-nearest neighbor and parallel collision detection algorithms and use these algorithms to accelerate motion planning.

KEYWORDS configuration space, motion planning, GPU parallel algorithm

1 Introduction

Intelligent robots are becoming increasingly important in both industry and everyday life. In industry, rising labor costs are motivating manufacturers to consider using more robots in factories. For example, in China the average minimum wage increased by more than 20% in 2012, while the supply of manufacturing robots also increased by 51% [1]. Europe and USA exhibit similar trends: Intelligent robots are being designed in order to make workers more productive and make manufacturers more competitive in terms of price and quality. One recent example among these intelligent robots is the new "Baxter" robot [2], which is equipped with software that enables the robot to learn various tasks from human demonstration, recognize different objects, and react intelligently to external forces. Intelligent robots are expected to assist people in everyday life. In the future, such robots are expected to perform various tasks, including ① household and care support, such as cooking and laundry; ② healthy life support, such as chatting with the elderly and taking care of people with disabilities; and ③ labor support in unsafe working conditions such as chemical plants [3]. Several successful prototypes for assistant robots exist. For example, the PR2 robot from Willow Garage has been shown to assist people with severe physical disabilities such as quadriplegia [4]; and humanoid robots such as the HRP-4 can perform human-like actions, and can communicate with people using speech [5]. In addition to their applications in industry and everyday life, modern intelligent robots can be helpful in other areas, including autonomous vehicles [6], medical and surgical intervention [7], emergency and disaster rescue [8], and military tasks [9].

The tremendous improvement in the design and availability of intelligent robots over the last decade is based on progress in many related areas, including computer vision, artificial intelligence, machine learning, control, sensor systems, and mechanical systems, which correspond to different components of an intelligent robot system (Figure 1). For example, the simultaneous localization and mapping (SLAM) algorithm enables a robot to accurately track its position in an unknown environment [10]. In addition, with the help of advanced vision techniques, robots can now recognize and segment objects from background point clouds [11]. Compared to traditional industrial robots, one important feature of the modern intelligent robot system is *high-level planning and navigation*. Its main purpose is to compute low-level instructions based on high-level descriptions for the tasks to be executed; these low-level instructions are then provided to the robot actuator system. This planning and navigation component is composed of many different sub-components (Figure 1) and

[1] Department of Computer Science, The University of Hong Kong, Hong Kong, China; [2] Department of Computer Science, University of N. Carolina, Chapel Hill, NC 27599-3175, USA
* Correspondence author. E-mail: jpan@cs.hku.hk
Received 12 March 2015; received in revised form 20 March 2015; accepted 25 March 2015

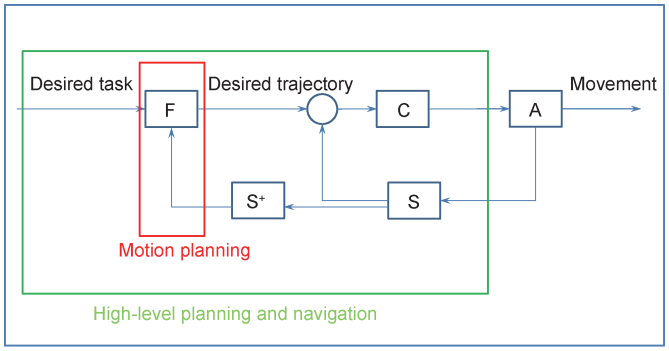


Figure 1. Important hardware and software subsystems in a robot system. ① Feed-forward system (F), including task planning, navigation strategy, motion planning, and trajectory generation; ② control system (C), including kinematics, dynamics, and control algorithms; ③ actuator system (A), including motors, servos, transmissions, and so forth; ④ sensor system (S), including various sensors such as camera, laser, IMU, and related low-level sensor data processing algorithms such as signal processing, estimation, and fusion; ⑤ sensor post-processing system (S^+), including localization, mapping, etc. The main software component of a robot system is the high-level planning and navigation, which determines the instructions sent to the actuator system, given the desired tasks to be executed. One important component of high-level planning and navigation is motion planning, which focuses on computing the trajectory from the environment description.

there has been extensive work in this area, such as task planning [12], feedback from observation [13, 14], optimal control [15], and adaptive control [16].

One of the most important sub-systems of the high-level planning and navigation component is the motion planning system, which enables the robot to move safely from an initial position to a goal position without colliding with any static or moving obstacles in the environment. A fast online motion planning algorithm is critical for many applications. For instance, there exists an increasing demand to incorporate mobile robots to assist humans in repetitive, non-value added tasks in the manufacturing domain [17]. To ensure a safe collaboration between robots and human beings, a robot needs to react to the changing environment in time, and this requires a real-time motion planning to avoid any accidents. Online motion planning is also critical while improving the level of automation for many traditional manufacturing process. For instance, for automatic bin picking, an efficient online motion planning is critical for the grasp performance [18]. In addition, a fast planning is also desirable for service robots working in hospitals and human homes. Motion planning problems can be directly formalized and solved in the 3D workspace, for instance with the widely-used potential field algorithms [19]. However, these workspace solutions cannot easily handle robots with different geometries and mechanical constraints. To overcome these difficulties, motion planning may be formalized and solved in a new space called the configuration space [20–22]. In the configuration space, a robot with a complex geometric shape in a 3D workspace is mapped to a point robot, and the robot's trajectory corresponds to a continuous curve in the high-dimensional configuration space (Figure 5). Based on the configuration space formulation, the motion planning problem can be solved in two steps:

(1) Construct a representation of the configuration space;

(2) Perform optimization based on the computed representation.

This motion planning pipeline based on configuration spaces is very successful and is adopted by many real-world planning applications that require optimal planning solutions. Many different representations for the configuration space have been proposed, including polyhedrons [23], semi-algebraic sets [24, 25], graphs [26], and trees/forests [27]. Different optimization approaches have been proposed for different configuration space representations, including computing a shortest path, computing the minimum distance to the boundary of a closed set inside the configuration space, and so forth. Moreover, the same pipeline is also implicitly used in some motion planning algorithms for only computing a feasible path (i.e., a collision-free path that does not violate other constraints). For example, many variants of rapidly exploring random tree (RRT) [27] use different heuristics to guide the search toward the goal configuration while growing a search tree structure as an approximate representation of the configuration space. Such a strategy can be viewed as a variant of the above pipeline, in which the configuration space construction alternates with the optimization computation.

However, this algorithmic pipeline based on configuration space still has many computational challenges. Two of the most important challenges are described here:

(1) Efficiently computing an approximate or exact representation for the configuration space is difficult, especially for high-DOF (degree of freedom) robots with high-dimensional configuration spaces. Such configuration space approximation problems would have exponential complexity.

(2) Many robotics applications require real-time planning in order to work reliably and efficiently in human environments with moving obstacles, but performing optimization in the computed representation for the configuration space can be time consuming.

In this paper, we will discuss our recent work for solving these challenges related to the configuration space. In particular, we first demonstrate how to convert the configuration space construction problem into a machine learning problem, and then use active learning to compute an approximate configuration space efficiently and robustly (Section 4). Second, we provide parallel GPU-based algorithms to accelerate the optimization computations in the configuration space, which can allow for real-time planning computation in many challenging environments (Section 5).

2 Background and related work

2.1 Configuration space

The configuration space is a key concept used in classical mechanics to describe and analyze the motion of many important systems [28]. Generally, a *configuration* $\boldsymbol{q}$ is a vector of independent parameters uniquely specifying the state of a system; and a configuration space or C-space is a collection of all possible configurations for a given system. For example, for a system of n point particles, the configuration is a vector describing the positions of all the particles, and the corresponding C-space is $\mathbf{R}^{3n}$; the configuration of a 3D rigid body

consists of its position and orientation, and the configuration space is **SE**(3) if both rotation and translation are allowed, and $\mathbf{R}^3$ if only translation is allowed; and the configuration of an articulated object is the vector of all its joint angles.

The configuration space of a robot A is composed of two components: *collision-free space* $C_{free} = \{q : A(q) \cap B = \emptyset\}$ and *in-collision space or obstacle space* $C_{obs} = \{q : A(q) \cap B \neq \emptyset\}$, where B corresponds to the geometric representation of obstacles in the environment and $A(q)$ corresponds to A with the configuration q. C_{obs} is a closed set and its boundary is denoted as the *contact space* $C_{cont} = \partial C_{obs}$, which corresponds to the set of configurations where A and B just touch each other without penetration. Figure 2 shows an example of the C-space of two

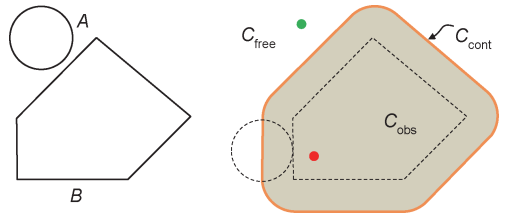


Figure 2. The configuration space of two objects. The orange curve highlights the contact space C_{cont} of A and B. A point inside/on the orange curve belongs to C_{obs} and a point outside the orange curve belongs to C_{free}. The red and green points denote configurations in C_{obs} and C_{free}, respectively.

objects where C_{cont} is highlighted with an orange curve.

In the special case when A and B are both rigid objects and robot A can only perform translational motion, C_{obs} is equal to the well-known Minkowski sum between A and B: $C_{obs} = A \oplus (-B) = \{x = x_A + x_B | x_A \in A, x_B \in -B\}$. One example of the Minkowski sum is shown in Figure 2. When robot A can perform general motion (i.e., both translation and rotation), the geometry of C_{obs} is much more complicated, as shown by the

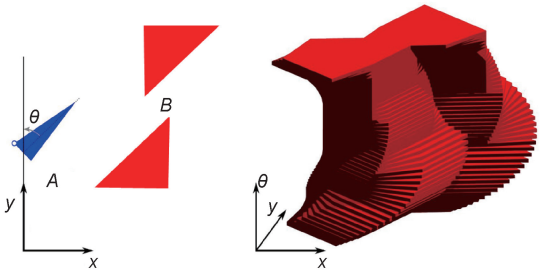


Figure 3. C_{obs} between 2D rigid objects *A* and *B*. For each rotation angle $\theta \in [0, 2\pi)$, we can compute the Minkowski sum between $A(\theta)$ and $-B$, where $A(\theta)$ is the resulting shape after rotating A about the origin with θ degrees. When stacking the Minkowski sums for all angles θ, we obtain the C_{obs} between A and B.

2D example in Figure 3.

Based on the notion of configuration space, the motion planning problem in 3D workspace can be reduced to path planning for a point robot in C-space, that is, finding a curve in C_{free} connecting the given initial and goal configurations of the robot.

2.2 Configuration space construction

Before performing the motion planning computation in the configuration space, one prerequisite is to compute the geometry of C-space in an appropriate representation (e.g., a graph or a surface). Since C-space = $C_{free} \cup C_{obs}$ and $C_{free} \cap C_{obs} = \emptyset$, we only need to construct the representation for either C_{free} or C_{obs}. Another equivalent solution is to compute C_{cont}, the boundary between C_{free} and C_{obs}.

Previous work on configuration space construction can be categorized into two different methods: geometry-based and topology-based. Geometry-based methods compute the exact geometric representation of the configuration space, while topology-based methods capture the connectivity of the configuration space.

Geometry-based methods are usually limited to low-dimensional configuration spaces, due to the combinatorial complexity involved in computing the boundary of C_{obs} for high-dimensional configuration spaces. Most previous work has focused on the special case when objects A and B are rigid bodies only performing translational motion. As mentioned in Section 2.1, the resulting C_{obs} is the Minkowski sum between A and $-B$. Even for this special case, the computational complexity involved in computing C_{obs} is still high: The complexity is $O(mn)$ when A and B are both convex-objects and is $O(m^3n^3)$ when A and B are both non-convex objects [29], where m and n are the number of triangles in A and B, respectively. In addition to the high complexity, most existing implementations for computing the Minkowski sum are prone to challenges that arise in the context of 3D geometric algorithms. In particular, these implementations are ① not robust to numerical errors, and ② susceptible to degeneracies (i.e., cannot reliably handle polygon soups or meshes with holes). Recent work has proposed methods [30–32] for computing the approximate Minkowski sum efficiently and reliably, but these methods are also prone to robustness issues and can have high complexity in terms of dealing with complex objects. Options other than the Minkowski sum exist for computing C_{obs}. For example, Varadhan et al. [33] computed the C_{obs} for 2D objects with rotation and translation by approximating the C_{obs} with an adaptive grid; and Zhang et al. [34] computed an approximation to 4D C-space using cell decomposition.

Topology-based methods capture the connectivity of the configuration space. Most previous approaches attempt to capture the connectivity of C_{free} using sampling techniques [26, 27]. The basic idea is first to generate random samples (called milestones) in C_{free} and then organize these samples using a graph structure or a forest of tree structures (Figure 4). As the topology of C_{free} can be rather complex, and may consist of multiple components or small, narrow passages, it is hard to capture the full connectivity of C_{free} using random sampling. There is extensive work on improving the connectivity computation by using different sampling strategies [35–40]. Recent work attempts to capture the topology of both C_{free} and C_{obs} [41]. Topology-based methods can compute an approximate C-space representation much faster than geometry-based methods. However, these methods do not work well with narrow passages and can be slow for high-DOF robots.

2.3 Optimization in configuration space

Once an exact or approximate representation for the configu-

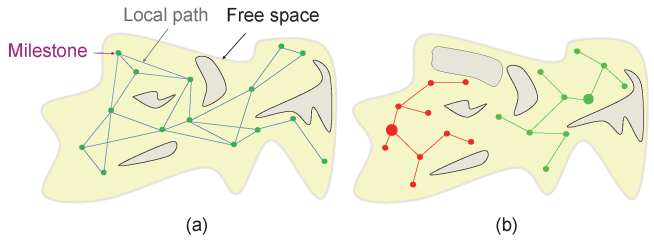


Figure 4. Topology-based methods for configuration space computation. (a) Capture C_{free} using a graph structure; (b) capture C_{free} using a forest of tree structures.

ration space is computed, we next need to perform optimization in this C-space representation. For example, the goal of motion planning is to compute a trajectory in C-space, as shown in Figure 5. The trajectory should satisfy the following constraints: ① It should be completely inside C_{free}; and ② it should be feasible, e.g., for humanoid robots, the robot should not fall down when following the trajectory. Moreover, it is preferable for the trajectory to be optimal under some metric. For instance, the optimal trajectory could be the shortest, take the least time to execute, or maintain the maximum distance from obstacles. As a result, motion planning can be formalized as a constrained optimization problem in C-space. Similar formulation can be applied to different applications,

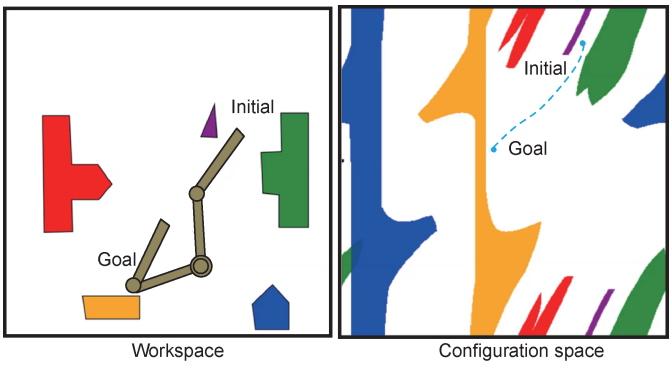


Figure 5. Motion planning in workspaces and in configuration spaces. The left figure shows obstacles (with different colors) and a 2-linked robot in the workspace (both the initial and goal settings). The right figure shows the configuration space corresponding to the workspace in the left figure, where different colors describe the correspondence between obstacles in the workspace and obstacles in the configuration space. The blue curve is a trajectory connecting the initial and goal configurations, and is the result of a motion planning algorithm.

such as penetration depth computation [39, 42–44].

Optimization in C-space is usually computationally expensive, especially for a high-dimensional C-space with a complicated structure and topology. To illustrate the computational challenge for C-space optimization problems, we take motion planning in C-space as an example. Theoretically, motion planning using the exact representation of C-space has high computational complexity. Planning algorithms are considered to be "complete" if for any planning problem instance, the algorithm will either find a solution or will correctly report that no solution exists. Complete planning algorithms have been proved to be PSPACE-hard [45] and PSPACE-complete [24], and kinodynamic motion planning (i.e., motion planning with simple kinematic or dynamic constraints) has been shown to be NEXPTIME-hard [25]. The decidability is still unknown for motion planning with general differential constraints [46]. When the approximate representation of C-space is used (e.g., using a graph or forest to approximate the connectivity of C_{free}), there exist approximate motion planning algorithms that provide guarantees of probabilistic completeness [26, 27] and/or asymptotic optimality [47]. The complexity of these approximate motion planning algorithms is usually bounded by $O(n\ln n)$ where n is the number of configuration samples used in the approximate representation of C-space [47]. Since n can be very large when C_{free} has narrow passages and/or high dimensionality [48], the performance of these approximate algorithms is still far from real-time.

Various planning methods related to C-space have been proposed in the past decades, including optimization-based planning algorithms such as CHOMP [49] and TrajOPT [50], and search-based algorithms such as Anytime A* [51]. For motion planning of high-DOF robots, most of the practical methods are based on randomized algorithms, including probabilistic roadmap (PRM) [26] and RRT [27].

3 Overview

Our solutions to the challenges related to the configuration space include two parts, for the configuation space construction and the optimization in configuration spaces, respectively.

3.1 Efficient configuration space construction

In the first part [52], we present a novel algorithm to efficiently approximate a high-dimensional configuration space using machine learning techniques. The main idea is to generate samples in the configuration space and then use these samples to approximate the contact space C_{cont} by a separating surface that can correctly separate all the in-collision and collision-free samples. This separating surface is computed using support vector machine (SVM) classification. Our method greatly reduces the required number of samples by leveraging incremental and active learning techniques. When the number of samples increases, the approximate contact space computed by our method can quickly converge to the exact contact space; we also provide bounds on the expected error in the approximate contact space. We evaluate the performance of our algorithm on high-dimensional benchmarks.

To construct a representation of the configuration space, we use an offline learning algorithm, as shown in the left box in Figure 6. We first generate a small set of uniform samples in a subspace of C-space for two given objects. Next, we justify whether these configurations lie in C_{free} or in C_{obs} by performing exact collision checking between the two objects. We use the notation $c(\boldsymbol{q}) \in \{-1, +1\}$ to denote the collision state of a configuration $\boldsymbol{q}$, that is, $c(\boldsymbol{q}) = +1$ if $\boldsymbol{q} \in C_{obs}$ and $c(\boldsymbol{q}) = -1$ if $\boldsymbol{q} \in C_{free}$. Given the collision states of all configuration samples, a coarse approximation of the contact space, LCS_0 (Figure 6(b)), is computed using classifiers, where LCS stands for *learned contact space*. Next, we select new samples in C-space to further improve the accuracy of the initial representation LCS_0 using active learning. During active learning, we either select samples that are far away from prior samples (*exploration*) (Figure 6(c)) or samples that are near LCS_0 (*exploi-*

tation) (Figure 6(d)). After the new samples are generated, we compute an updated approximation LCS_1 (Figure 6(e)) based on incremental machine learning techniques. We repeat this process, generating a sequence of approximate representations LCS_0, LCS_1, ..., with increasing accuracy. This iterative process is repeated until the collision states of all the new samples can be correctly predicted by the current approximation. The final result *LCS* (Figure 6(f)) corresponds to a smooth surface approximation of the contact space.

3.2 Efficient optimization in configuration spaces

The approximate contact space computed by the first part of our work can potentially be directly used for motion planning, for instance, by using it as the collision-free constraint in the trajectory optimization framework [53]. However, this constraint is highly non-convex. Thus, solving the resulting optimization problem is still non-trivial, and this is one focus of our ongoing work. Another way to solve the motion planning problem is using a sample-based approximation to the configuration space, which has been adopted by many traditional planning algorithms such as the PRM. In the se-cond part [54] of this paper, we present a novel parallel algorithm for real-time motion planning of high-DOF robots that exploits the computational capability of a $400 USD commodity graphics processing unit (GPU). Current GPUs are programmable many-core processors that can support thousands of concurrent threads. We use them for real-time computation of a PRM and a lazy planner. We describe efficient parallel strategies for constructing the roadmap that include sample generation, collision detection, connecting nearby samples, and local planning. The query phase is also performed in parallel based on a graph search. In order to design an efficient single query planner, we use a lazy strategy that defers collision checking and local planning. We also describe new hierarchy-based collision detection algorithms, to accelerate the overall performance.

We choose the PRM algorithm as the underlying method for parallel planning, because it is most suitable to exploit multiple cores and data parallelism on GPUs. The PRM algorithm has two phases: roadmap construction and querying. The roadmap construction phase includes four main steps: ① Generate samples in the configuration space; ② compute milestones that correspond to the samples in the free space by performing discrete collision queries; ③ for each milestone, find other milestones that are nearest to it; and ④ connect nearby milestones using local planning, and form a roadmap. The query phase includes two parts: ① Connect initial and goal configurations of the query to the roadmap, and ② execute a graph search algorithm on the roadmap and find collision-free paths.

Parts of the PRM algorithm, such as the collision queries, are embarrassingly parallel [55]. We can use a many-core GPU to significantly enhance the performance of the other components as well. The framework of our PRM algorithm on the GPU is shown in Figure 7. We parallelize each of the six steps of the PRM algorithm efficiently. First, each thread of a multi-core GPU generates a random robot configuration, and some of these configurations will collide with obstacles. All of the collision-free samples are milestones and become vertices of the roadmap graph. Next, each GPU thread computes the *k*-nearest neighbors of a single milestone and collects all the neighborhood pairs. Each thread then checks whether it is possible to connect these adjacent pairs by performing local planning. If there is a collision-free path between that neighborhood pair of milestones, we add the edge to the roadmap. Once the roadmap is built, queries are connected to the roadmap in parallel and we use a parallel graph search algorithm to find paths.

The resulting GPU-based framework is very efficient for a multi-query version of the planning problem. The most expensive step in this computation is the local planning algorithm; thus, we use new collision detection algorithms to im-

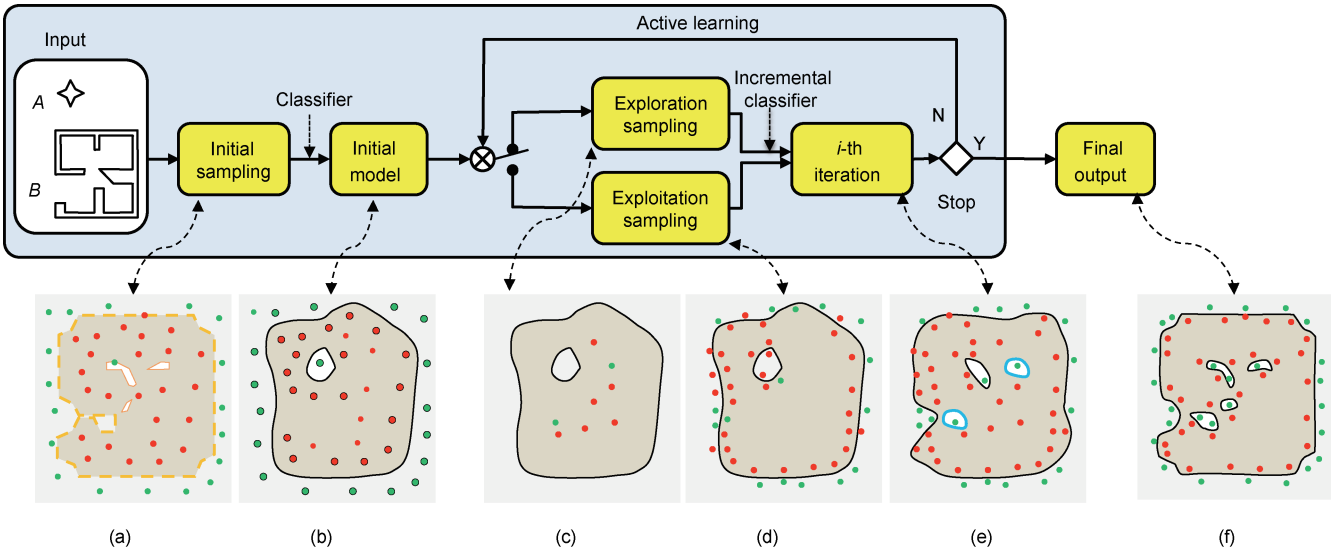


Figure 6. Offline computation pipeline for C_{cont} approximation. The different approximations of *LCS* are shown below the corresponding stages. We use green points to indicate collision-free configuration samples and red points to indicate in-collision samples. (a) Exact contact space (for reference); (b) initial model; (c) exploration; (d) exploitation; (e) solution after *i*-th iteration; (f) final approximate contact space.

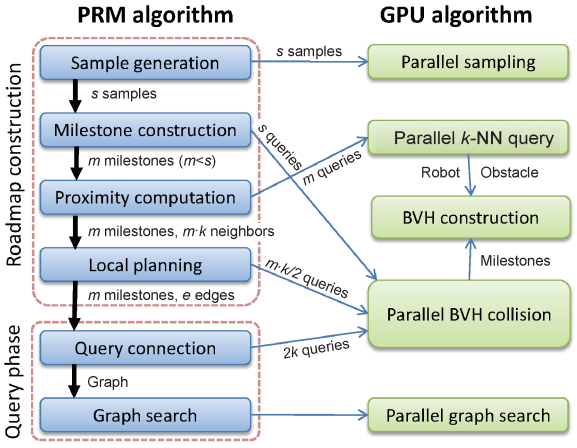


Figure 7. PRM overview and parallel components in our algorithm.

prove its performance. In order to accelerate the single-query algorithm, we introduce a solution that uses a lazy strategy and defers collision checking for local planning. In other words, the algorithm connects all the edges corresponding to the nearest neighbors and searches for paths between the initial and final configurations. After that, it performs local planning on the edges that constitute these paths.

4 Learning-based approximate contact space construction

In this section, we present our algorithm for the offline learning of the contact space and the computation of *LCS*. Different stages of this algorithm are shown in Figure 6.

4.1 Initial sampling

We perform uniform sampling in C-space to obtain a set of configuration points. Rather than sampling the entire C-space, we generate samples in a subspace that contains C_{cont}. Given two objects A and B, the contact space C_{cont} is contained in the in-collision space of their bounding volumes $BV(A)$ and $BV(B)$.

4.2 Compute LCS_0

Given a set of k samples from $C_{\text{obs}}(BV(A), BV(B))$, we perform exact collision queries between A and B to check whether these samples are within in-collision space or not. Our goal is to learn an approximate representation LCS_0 from these configurations. In particular, LCS_0 corresponds to a decision function $f(\boldsymbol{q}) = 0$ that is fully determined by a set of configurations $\boldsymbol{S}$ in C-space. We refer to $f(\boldsymbol{q})$ as the *classifier* and use it to predict whether a given configuration $\boldsymbol{q}$ is collision-free ($f(\boldsymbol{q}) < 0$) or in-collision ($f(\boldsymbol{q}) > 0$). $\boldsymbol{S}$ corresponds to the support vectors, which are a small subset of configuration samples used in learning. Intuitively, $\boldsymbol{S}$ are the samples that are closest to C_{cont}.

We use the SVM classifier [56] to learn LCS_0 from the initial sampling of k configurations. An SVM generates a decision function that is a smooth nonlinear surface. We use the hardmargin SVM, as the underlying samples can always be separated into collision-free and in-collision spaces. Intuitively, an SVM uses a function to map the given samples $\{\boldsymbol{q}_i\}$ from the *input space* into a higher (possibly infinite) dimensional *feature space*. An SVM computes a linear separating hyperplane characterized by parameters $\boldsymbol{w}$ and b. The hyperplane's maximal margin is in the higher dimensional feature space. The hyperplane corresponds to a nonlinear separating surface in the input space. The $\boldsymbol{w}$ is the normal vector to the hyperplane, and the parameter b determines the offset of the hyperplane from the origin along the normal vector. In the feature space, the distance between a hyperplane and the closest sample point is called the "margin," and the optimal separating hyperplane should maximize this distance. The maximal margin can be achieved by solving the following optimization problem:

$$\min_{\boldsymbol{w},\, b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{s.t. } c_i(\boldsymbol{w}\cdot\phi(\boldsymbol{q}_i)+b) \geqslant 1, \quad 1 \leqslant i \leqslant k \tag{1}$$

where $c_i \in \{-1; +1\}$ is the collision state of each sample $\boldsymbol{q}_i$. We define $K(\boldsymbol{q}_i, \boldsymbol{q}_j) = \phi(\boldsymbol{q}_i)^{\mathrm{T}}\phi(\boldsymbol{q}_j)$ as the kernel function (i.e., a function used to calculate inner products in the feature space). We use radial basis function (RBF) kernel in our results.

The solution of Eq. (1) is a nonlinear surface in the input space (and a hyperplane in the feature space) that separates collision-free and in-collision configurations. This solution can be formulated as:

$$f(\boldsymbol{q}) = \boldsymbol{w}^*\cdot\phi(\boldsymbol{q}) + b^* = \sum_{i=1}^{k} \alpha_i c_i K(\boldsymbol{q}_i, \boldsymbol{q}) + b^* \tag{2}$$

where $\boldsymbol{w}^*$ and b^* are the solutions of Eq. (1) and $\alpha_i \geqslant 0$. The vectors $\boldsymbol{q}_i$ corresponding to the non-zero α_i are called the *support vectors*, which we denote as $\boldsymbol{S}$. Intuitively, the support vectors are those samples closest to the separating hyperplane $f(\boldsymbol{q}) = 0$, as shown by the larger red and green points in Figure 6(b). Thus, LCS_0 consists of an implicit function $f_{LCS_0}(\boldsymbol{q}) = f(\boldsymbol{q})$ and a set of samples $\boldsymbol{S}_{LCS_0} = \boldsymbol{S}$ (i.e., the support vectors), which are used to approximate the exact contact space.

4.3 Refine LCS_0 using active learning

We refine LCS_0 using active learning. The goal is to actively

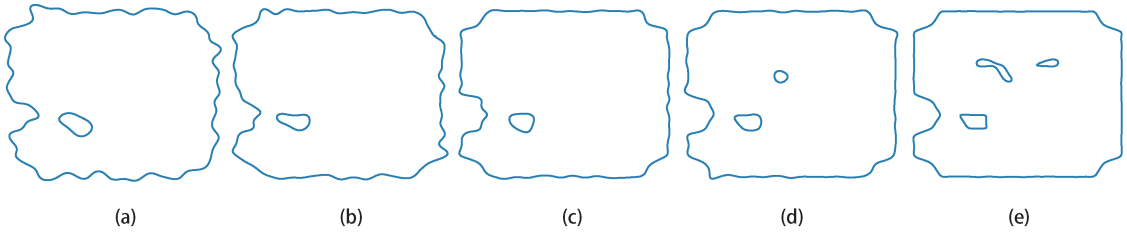
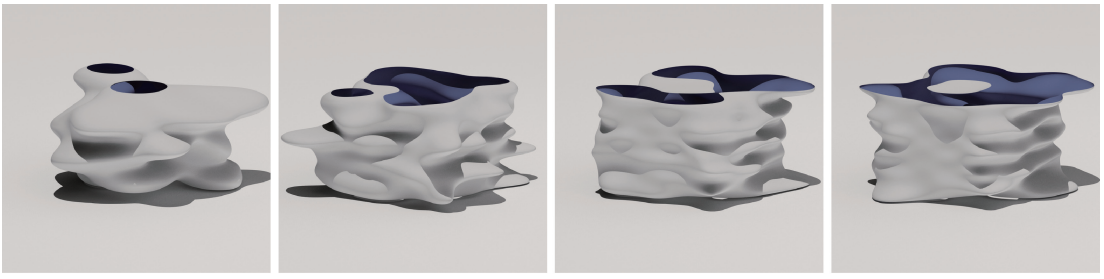


Figure 8. *LCS* computation using active learning for the 2D contact space between 2D star and room models shown as the input of learning pipeline in Figure 6. We highlight the number of support vectors corresponding to LCS_i. (a) i = 0, $|S|$ = 37; (b) i = 4, $|S|$ = 75; (c) i = 8, $|S|$ = 198; (d) i = 14, $|S|$ = 327; (e) i = 20; $|S|$ = 654.



(a) (b) (c) (d)

Figure 9. *LCS* computation using active learning for the 3D contact space between 2D star and room models shown as the input of learning pipeline in Figure 6. We highlight the number of support vectors corresponding to LCS_i. The vertical axis represents the rotational component of the C-space. (a) LCS_0, $|S| = 88$; (b) LCS_5, $|S| = 174$; (c) LCS_9, $|S| = 237$; (d) LCS_{12}, $|S| = 248$.

select new samples so that a better approximate contact space representation, LCS_1 , can be obtained by incorporating these samples into LCS_0. We use a combination of exploration and exploitation [57]. The idea is to determine whether to explore or to exploit by flipping a biased coin with a certain probability for landing on heads (initially 0.5). If the result is heads, we apply exploration; if it is tails, we apply exploitation. The probability of landing on heads is adjusted according to the fraction of exploration samples' collision states that are correctly predicted by the current LCS_i. The new samples are used to update LCS_0 and generate a new approximation LCS_1 (or refine from LCS_i to LCS_{i+1}). We repeat the active learning step until all the new samples can either be correctly predicted by the current LCS_i, or the final result (represented as LCS) has sufficient accuracy to approximate C_{cont}.

4.4 Incremental learning

Instead of computing a new decision function from scratch using all the previous samples, we apply incremental learning techniques to efficiently compute LCS_{i+1} from LCS_i. Incremental learning utilizes a small set of new samples to update LCS_i. The decision function of LCS_i serves as the initial guess for generating LCS_{i+1}. The incremental SVM [58] can update the current result generated using SVMs; the key is to retain the optimality condition of Eq. (1) (i.e., the Kuhn-Tucker condition) on all prior samples while adding new samples. This is achieved by adjusting the coefficients α_i and b in Eq. (2) and by adjusting support vector set $\boldsymbol{S}$. The coefficient adjustment and the support vector changes are guided by the gradient of the objective function in Eq. (2).

4.5 Terminating active learning

Active learning terminates when either of these conditions has been satisfied:

(1) The collision states of all the new samples generated during exploration and exploitation can be correctly predicted by the current approximation LCS_i.

(2) The total number of samples used in active learning iterations is more than a user-specified threshold.

The first condition guarantees that all the configurations used for learning LCS are consistent (i.e., they can be correctly predicted by LCS). This implies that the current LCS is a close approximation of the underlying contact space. The second condition controls the accuracy of the approximate C_{cont}: As more samples are used, we get a better approximation to C_{cont}.

5 GPU-based real-time optimization in configuration spaces

In this section, we present the details about how to use GPU to accelerate the optimization speed in configuration spaces.

5.1 Hierarchy computation

We construct a bounding volume hierarchy (BVH) for the robot and one for each of the obstacles in the environment, to accelerate the collision queries. We use the GPU-based construction algorithm introduced in Ref. [59], which can construct the hierarchy of axis-aligned bounding boxes or oriented bounding boxes (OBB) in parallel on the GPU, for given triangle representation. For collision detection, we use the OBB hierarchy, as it provides higher culling efficiency and improved performance on GPU-like architectures. These hierarchies are stored in the GPU memory and we apply appropriate transformations for different configurations.

5.2 Roadmap construction

The roadmap construction phase tries to capture the connectivity of the free configuration space, which is the main computationally intensive part of the PRM algorithm.

(1) Sample generation: We first need to generate random samples within the configuration space. Since samples are independent, we schedule enough parallel threads to utilize the GPU and use MD5 cryptographic hash function [60], which in practice provides good randomness without a shared seed.

(2) Milestone computation: For each configuration generated in the previous step, we need to check whether it is a milestone: i.e., a configuration that lies in the free space and does not collide with obstacles. We use a hierarchical collision detection approach using BVHs to test for overlap between the obstacles and the robot in the configuration defined by the sample. The collision detection is performed in each thread by using a traversal algorithm in the two BVHs. The traversal algorithm starts with the two BVH root nodes and tests the OBB nodes for overlap in a recursive manner. If two nodes overlap, then all possible pairings of their children should be recursively tested for intersection.

We also use GPUs to compute the actual BVH structure for both the robot and obstacles by using a parallel hierarchy construction algorithm [59]. Since the robot's geometric objects move depending on the configuration, its BVH is only valid for the initial configuration. In order to avoid recomput-

ing a BVH for each configuration, we instead transform each node of the robot's BVH with the current configuration sample before performing overlap tests. Thus, only nodes that are actually needed during collision testing are transformed.

(3) Proximity computation: For each milestone computed, we need to find its *k*-nearest neighbors (*k*-NN). In general, there are two types of *k*-NN algorithms: exact *k*-NN and approximate *k*-NN, which is faster by allowing a small relaxation. Our proximity algorithm is based on a range query that uses a BVH structure of the points in configuration space.

For 3-DOF Euclidean space, we first construct the BVH structure for all the milestones using a parallel algorithm [59]. For each configuration $\boldsymbol{q}$, we enclose it within an axis-aligned box: A box with $\boldsymbol{q}$ as the center and with 2ε as the edge length. Next, we traverse the BVH tree to find all leaf nodes (i.e., configurations) that are within the ε box. This reduces to a range-query for $\boldsymbol{q}$. For a non-Euclidean DOF, we duplicate samples to transform it into a Euclidean space locally. For example, suppose one DOF is the rotation angle $\alpha \in [0, 2\pi]$. We add another sample $\alpha^* \in [-\pi, 3\pi]$ with a distance 2π to α. If all 3-DOF are rotations, we need to add another 7 samples for each milestone. Once the range query finishes, we choose the *k*-nearest ones from all the query results; this gives us the exact nearest neighbors. This approach can be extended to handle *k*-NN search in high dimensional space by using a decomposition strategy; the details are in our recent work [61].

To further improve the performance of proximity computation in high dimensional space, we have developed a new *k*-NN algorithm, which uses locality sensitive hashing (LSH) and cuckoo hashing to efficiently compute approximate *k*-NN in parallel on the GPU. For more details, please refer to our recent work [62].

(4) Local planning: Local planning checks whether there is a local path between two milestones, which corresponds to an edge on the roadmap. It is the most expensive part of the PRM algorithm. Suppose we have n_m milestones, and each milestone has at most n_k nearest neighbors. Then the algorithm performs local planning at most $n_m \cdot n_k$ times. If we use DCD, then we need to perform at most $n_l = n_m \cdot n_k \cdot n_i$ collision queries, which can be very high for a complex benchmark. For multi-query problems, this cost can be amortized over multiple queries, as the roadmap is constructed only once. For a single-query problem, computing the whole roadmap is too expensive.

Therefore, in the single-query case, we use a lazy strategy to defer local planning until absolutely necessary. Given a query, we compute several different candidate paths in the roadmap graph from the initial to final configuration and only check local planning for roadmap edges on the candidate paths. Local planning may conclude that some of these edges are not valid, and in that case, we delete them from the roadmap. If there exists one candidate path without invalid edges, the algorithm has found a collision-free solution. Otherwise, we compute candidate paths again on the updated roadmap and repeat the above process. This lazy strategy can greatly improve performance for single queries.

5.3 Query phase

The query phase includes two parts: connecting queries to the roadmap and executing graph searches to find paths.

(1) Query connection: Given the initial-goal configurations in a single query, we connect them to the roadmap. For both of these configurations, we find the *k*-nearest milestones on the roadmap and add edges between the query and milestones that can be connected by local planning. We use the same algorithm from the roadmap construction phase, except that the *k* used is 2–3 times larger in order to increase the probability of finding a path.

(2) Graph search: The search algorithm tries to find a path on the roadmap connecting initial and goal configurations. We use depth-first search (DFS) or breadth-first search (BFS) for the graph search. For the multi-query case, each GPU thread traverses the roadmap for one query using DFS, and the final results are collision-free paths. For the single-query case, we exploit all the GPU threads to find the path for one query using a BFS search: For nodes that are the same number of steps away from the initial node, we add their unvisited neighbors into the queue in parallel. In other words, different GPU cores traverse different parts of the graph. The main challenge of this method is that work is generated dynamically as the BFS traverse progresses, and the computational load on different cores can change significantly. To address the problem of load balancing and work distribution so that parallelism for all cores is maintained, we use the light-weight load balancing strategy in Ref. [63].

6 Experiments

In this section, we investigate the performance of our approaches while solving two challenges related to the configuration space.

6.1 Configuration space construction

We have used a set of complex benchmarks to evaluate the contact space approximation results of our algorithm.

We first compute the contact space for 2D objects shown as the input of the learning pipeline in Figure 6. In this experiment, object *A* can only undergo translational motion and object *B* is fixed. The resulting configuration space has 2 degrees of freedom, and the series of *LCS* surfaces computed by our learning-based approach is shown in Figure 8. We can see that after a few iterations with several hundred samples, the learned *LCS* can well approximate the exact contact space between these two objects.

Next, we compute the contact space for the same pair of 2D objects, but we now allow object *A* to perform rotation in the 2D plane. The resulting configuration space has 3 degrees of freedom, and the series of *LCS* surfaces generated by our approach is shown in Figure 9. Similar to the case of the 2D contact space, the learned contact space can quickly converge to a good approximation of the exact contact space.

Finally, we compute the contact space for a pair of 3D objects shown in Figure 10(a), where the object *A* can only un-

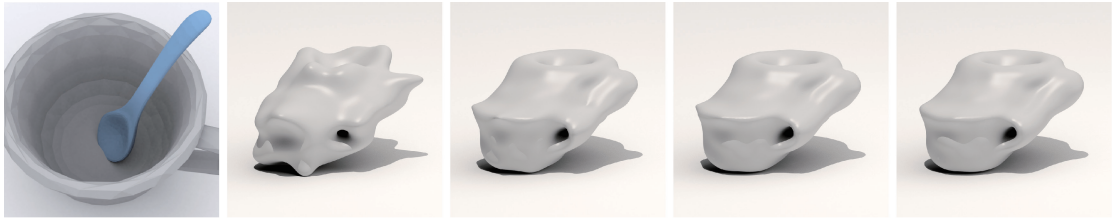


Figure 10. *LCS* computation using active learning for the 3D contact space between 3D cup and spoon. We provide the number of support vectors corresponding to LCS_i. As shown, the algorithm can compute a good approximation in a few iterations. (a) Objects A and B; (b) LCS_0, $|S| = 231$; (c) LCS_5, $|S| = 869$; (d) LCS_9, $|S| = 1350$; (e) LCS_{12}, $|S| = 1572$.

dergo translation and thus the corresponding configuration space is three-dimensional. Similarly, we can observe that the learned contact space sequence quickly converges to the exact contact space.

6.2 Optimization in configuration spaces

We present details of our implementation and evaluate the performance of our algorithm on a set of benchmarks. All the timings reported here were taken on a machine using a Intel Core i7 CPU (~$600) at 3.2 GHz CPU and 6 GB memory. We implemented our algorithms using CUDA on a NVIDIA GTX 285 GPU (~$380) with 1 GB of video memory.

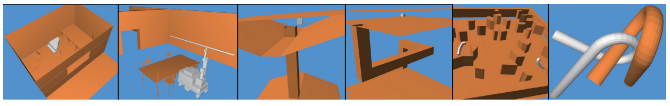


Figure 11. The benchmark scenes used for our algorithms in the following order: Piano (2484 triangles), helicopter (2484 triangles), maze3d1 (40 triangles), maze3d2 (40 triangles), maze3d3 (970 triangles), and alpha puzzle (2016 triangles).

We implement the PRM algorithm on the GPU (G-PRM) for multi-query planning problems, and implement its lazy version (GL-PRM) for single-query problems. We compare these with the PRM and RRT algorithms implemented in the OOPSMP library [64], which is a popular library for motion planning algorithms on CPU. The benchmarks used are shown in Figure 11. Our comparisons are designed as follows: For each benchmark, we find a suitable setting where CPU-PRM (C-PRM) finds a solution, and then we run G-PRM with a comparable number of samples. After that, we run GL-PRM with the same setting as G-PRM, and run CPU-RRT (C-RRT) with the same setting as C-PRM.

Table 1 shows the comparison of timings between algorithms. In general, G-PRM is about 10 times faster than C-PRM, and GL-PRM can provide another 10-fold acceleration for single-query problems. G-PRM is faster than C-PRM even for dynamic scenes. The current C-RRT and C-PRM are both single-core versions. However, even a multi-core version of PRM would only improve the timing by 4-fold at most, because on an 8-core CPU it is hard to scale the hierarchy computations and nearest neighbor computations linearly. Our GPU algorithms can still provide performances 1–2 orders of magnitude higher than CPU algorithms.

Table 1. The left two columns evaluate the performance of the PRM and RRT algorithms in the OOPSMP. The right two columns evaluate the performance of our GPU-based algorithms.

	C-PRM	C-RRT	G-PRM	GL-PRM
Piano	6.53 s	19.44 s	1.71 s	111.23 ms
Helicopter	8.20 s	20.94 s	2.22 s	129.33 ms
Maze3d1	138.00 s	21.18 s	14.78 s	71.24 ms
Maze3d2	69.76 s	17.40 s	14.47 s	408.60 ms
Maze3d3	8.45 s	4.30 s	1.40 s	96.37 ms
Alpha1.5	65.73 s	2.80 s	12.86 s	1446.00 ms

Figure 12 shows the timing breakdown between various steps for G-PRM and GL-PRM. The difference between the performance of the two algorithms is clear: In G-PRM, local planning is the bottleneck and dominates the timing, while in GL-PRM the graph search takes longer because local planning is performed in a lazy or output-sensitive manner. In GL-PRM, three components take most timing: milestone

Figure 12. Split-up of timings: the fraction of time spent in different parts of the G-PRM and GL-PRM.

construction, proximity computation, and graph search, because all of them may perform collision queries heavily. If the environment is cluttered and the model has complex geometry, milestone construction will be slow (alpha puzzle in Figure 12). If the environment is an open space and has many milestones, proximity computation will be the bottleneck (maze3d2 in Figure 12). If the lazy strategy cannot guess a correct path, then the graph search will be computationally intensive due to the large number of collision queries (maze3d3 in Figure 12). However, in all these environments, GL-RPM outperforms all other methods.

We tested the scalability of G-PRM and GL-PRM on the maze3d3 benchmark, and the result is shown in Figure 13. It is obvious that GL-PRM is generally faster than G-PRM, and both algorithms achieve near-linear scaling on the benchmark. However, observe that as the number of samples increases, GL-PRM slows down faster than G-PRM. This is because when the number of samples increases, proximity computation becomes increasingly expensive and dominates the timing when the number of samples is near 1 million.

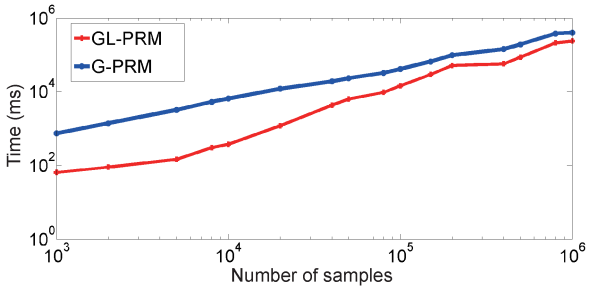
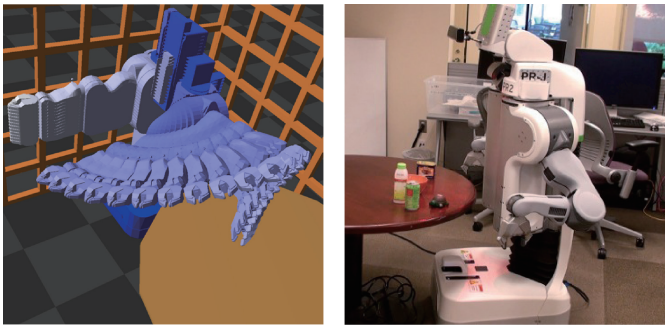


Figure 13. The scalability of the G-PRM and GL-PRM.

Our method can be extended for efficient planning for articulated bodies, and can achieve real-time performance for the PR2 grasping operation, as shown in Figure 14.



(a) (b)

Figure 14. Our GPU-based motion planner can compute a collision-free path for PR2 in less than 1. (a) PR2 simulation; (b) real PR2.

7 Conclusions

In this paper, we have addressed two computational challenges related to configuration spaces, that is, configuration space construction and efficient optimization in configuration space. As research in configuration space continues, we expect progress in all these areas and more. While there is always more to do, the work presented in this paper has addressed many of the important issues in this field.

To summarize the main results presented in this paper, we first presented a novel approach to the approximation of configuration spaces. The main idea is to sample the configuration space and approximate the contact space based on machine learning classifiers, particularly support vector machines. Furthermore, we use active learning techniques to select the samples during precomputation. Next, we introduced a whole motion planning algorithm on GPUs. Our algorithm can exploit all the parallelism within the PRM algorithm, including the high-level parallelism provided by the PRM framework and the low-level parallelism within different components of the PRM algorithm, such as collision detection and graph search. This makes our work the first to perform real-time motion planning and global navigation in general environments using GPUs.

Acknowledgements

This research was partially supported by the Army Research Office, the National Science Foundation, Willow Garage, and the Seed Funding Programme for Basic Research at the University of Hong Kong.

Compliance with ethics guidelines

Jia Pan and Dinesh Manocha declare that they have no conflict of interest or financial conflicts to disclose.

References

1. G. Litzenberger. Professional service robots: Continued increase. *Statistical Department, International Federation of Robotics, Tech. Rep.*, 2012
2. E. Guizzo, E. Ackerman. How Rethink Robotics built its new Baxter robot worker. *IEEE Spectrum*, http://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker, 2012
3. K. Yamazaki, et al. Home-assistant robot for an aging society. *Proc. IEEE*, 2012, 100(8): 2429–2441
4. S. Cousins. Robots for humanity. http://www.willowgarage.com/blog/2011/07/13/robots-humanity, 2012
5. S. Kajita, et al. Cybernetic human hrp-4c: A humanoid robot with human-like proportions. In: C. Pradalier, R. Siegwart, G. Hirzinger, eds. *Robotics Research, ser. Springer Tracts in Advanced Robotics*, vol 70. Berlin: Springer, 2011, 70: 301–314
6. M. Montemerlo, et al. Junior: The Stanford entry in the urban challenge. *J. Field Robot.*, 2008, 25(9): 569–597
7. M. Bonfe, et al. Towards automated surgical robotics: A requirements engineering approach. In: *Proceedings of IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2012: 56–61
8. E. Guizzo. Robots enter fukushima reactors, detect high radiation. *IEEE Spectrum*, 2011. http://spectrum.ieee.org/automaton/robotics/industrial-robots/robots-enter-fukushima-reactors-detect-high-radiation
9. E. Ackerman. Latest alphadog robot prototypes get less noisy, more brainy, 2012. http://spectrum.ieee.org/automaton/robotics/military-robots/latest-ls3-alphadog-prototypes-get-less-noisy-more-brainy
10. S. Thrun, W. Burgard, D. Fox. *Probabilistic Robotics*. Boston, MA: The MIT

Press, 2005
11. R. B. Rusu, N. Blodow, Z. C. Marton, M. Beetz. Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments. In: *Proceedings of International Conference on Intelligent Robots and Systems*, 2009: 1–9
12. T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O' Donnell. Task-level planning of pick-and-place robot motions. *Computer*, 1989, 22(3): 21–29
13. L. Kaelbling, T. Lozano-P'erez. Unifying perception, estimation and action for mobile manipulation via belief space planning. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2012: 2952–2959
14. L. Kaelbling, T. Lozano-Perez. Hierarchical task and motion planning in the now. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2011: 1470–1477
15. R. F. Stengel. *Optimal Control and Estimation* (*Dover Books on Advanced Mathematics*). New York: Dover Publications, 1994
16. K. J. Astrom, B. Wittenmark. *Adaptive Control*. 2nd ed. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1994
17. V. Unhelkar, J. Perez, J. Boerkoel, J. Bix, S. Bartscher, J. Shah. Towards control and sensing for an autonomous mobile robotic assistant navigating assembly lines. In: *IEEE International Conference on Robotics and Automation*, 2014: 4161–4167
18. L. P. Ellekilde, H. G. Petersen. Motion planning efficient trajectories for industrial bin-picking. *Int. J. Robot. Res.*, 2013, 32(9–10): 991–1004
19. O. Khatib. Real-time obstacle avoidance for manipulators and mobile robot. *Int. J. Robot. Res.*, 1986, 5(1): 90–98
20. T. Lozano-Pérez, M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 1979, 22(10): 560–570
21. T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Trans. Syst. Man Cybern.*, 1981, 11(10): 681–698
22. T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, 1983, C-32(2): 108–120
23. B. Chazelle. Approximation and decomposition of shapes. In: J. T. Schwartz, C. K. Yap, eds. *Algorithmic and Geometric Aspects of Robotics*. Hillsdale: Lawrence Erlbaum Associates, 1987: 145–185
24. J. F. Canny. Some algebraic and geometric computations in PSPACE. In: *Proceedings of ACM symposium on Theory of Computing*, 1988: 460–467
25. J. F. Canny, J. Reif, B. Donald, P. Xavier. On the complexity of kinodynamic planning. In: *Proceedings of Symposium on Foundations of Computer Science*, 1988: 306–316
26. L. Kavraki, P. Svestka, J. C. Latombe, M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 1996, 12(4): 566–580
27. J. J. Kuffner, S. LaValle. RRT-connect: An efficient approach to single-query path planning. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2000: 995–1001
28. V. I. Arnold. *Mathematical Methods of Classical Mechanics*. New York: Springer-Verlag, 1989
29. D. Halperin. Robust geometric computing in motion. *Int. J. Robot. Res.*, 2002, 21(3): 219–232
30. J. M. Lien. Covering Minkowski sum boundary using points with applications. *Comput. Aided Geom. Des.*, 2008, 25(8): 652–666
31. J. M. Lien, N. M. Amato. Approximate convex decomposition of polyhedral. In: *Proceedings of the ACM Symposium on Solid and Physical Modeling*, 2007: 121–131
32. J. M. Lien. A simple method for computing Minkowski sum boundary in 3D using collision detection. In: *Algorithmic Foundation of Robotics VIII, Springer Tracts in Advanced Robotics*, vol 57. Berlin: Springer, 2009: 401–415
33. G. Varadhan, Y. J. Kim, S. Krishnan, D. Manocha. Topology preserving approximation of free configuration space. In: *Proceedings of International Conference on Robotics and Automation*, 2006: 3041–3048
34. L. Zhang, Y. J. Kim, D. Manocha. A hybrid approach for complete motion planning. In: *Proceedings of International Conference on Intelligent Robots and Systems*, 2007: 7–14
35. N. M. Amato, O. B. Bayazit, L. K. Dale, C. Johns, D. Vallejo. OBPRM: An obstacle-based prm for 3D workspaces. In: *Proceedings of Workshop on the Algorithmic Foundations of Robotics on Robotics*, 1998: 155–168
36. V. Boor, M. Overmars, A. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 1999: 1018–1023
37. D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In: *Proceedings of Workshop on the Algorithmic Foundations of Robotics on Robotics*, 1998: 141–153
38. S. Rodriguez, X. Tang, J. M. Lien, N. M. Amato. An obstacle-based rapidly-exploring random tree. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2006: 895–900
39. L. Zhang, D. Manocha. An efficient retraction-based RRT planner. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2008: 3743–3750
40. Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, J. H. Reif. Narrow passage sampling for probabilistic roadmap planners. *IEEE Trans. Robot.*, 2005, 21(6): 1105–1115
41. J. Denny, N. M. Amato. Toggle PRM: Simultaneous mapping of C-free and C-obstacle–A study in 2D. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011: 2632–2639
42. L. Zhang, Y. J. Kim, G. Varadhan, D. Manocha. Generalized penetration depth computation. *Comput. Aided Des.*, 2007, 39(8): 625–638
43. L. Zhang, Y. J. Kim, D. Manocha. A fast and practical algorithm for generalized penetration depth computation. In: *Proceedings of Robotics: Science and Systems*, 2007
44. C. Je, M. Tang, Y. Lee, M. Lee, Y. J. Kim. Polydepth: Realtime penetration depth computation using iterative contact-space projection. *ACM Transactions on Graphics*, 2012, 31(3): 5:1–5:14
45. J. H. Reif. Complexity of the mover's problem and generalizations. In: *Proceedings of Annual Symposium on Foundations of Computer Science*, 1979: 421–427
46. P. Cheng, G. Pappas, V. Kumar. Decidability of motion planning with differential constraints. In: *Proceedings of International Conference on Robotics and Automation*, 2007: 1826–1831
47. S. Karaman, E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.*, 2011, 30(7): 846–894
48. D. Hsu, J. C. Latombe, H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robot. Res.*, 2006, 25(7): 627–643
49. N. Ratliff, M. Zucker, J. A. D. Bagnell, S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In: *Proceedings of International Conference on Robotics and Automation*, 2009: 489–494
50. J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Proceedings of Robotics: Science and Systems*, 2013
51. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 2005
52. J. Pan, X. Zhang, D. Manocha. Efficient penetration depth approximation using active learning. *ACM Transactions on Graphics*, 2013, 32(6): 191:1–191:12
53. J. Schulman, et al. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Robot. Res.*, 2014, 33(9): 1251–1270

54. J. Pan, C. Lauterbach, D. Manocha. g-Planner: Real-time motion planning and global navigation using GPUs. In: *Proceedings of AAAI Conference on Artificial Intelligence*, 2010: 1245–1251
55. N. M. Amato, L. Dale. Probabilistic roadmap methods are embarrassingly parallel. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 1999: 688–694
56. V. N. Vapnik. *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995
57. S. J. Huang, R. Jin, Z. H. Zhou. Active learning by querying informative and representive examples. In: *Proceedings of Advances in Neural Information Processing Systems*, 2010: 892–900
58. M. Karasuyama, I. Takeuchi. Multiple incremental decremental learning of support vector machines. *IEEE Trans. Neural Netw.*, 2010, 21(7): 1048–1059
59. C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha. Fast BVH construction on GPUs. *Comput. Graph. Forum*, 2009, 28(2): 375–384
60. S. Tzeng, L. Y. Wei. Parallel white noise generation on a GPU via cryptographic hash. In: *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2008: 79–87
61. J. Pan, C. Lauterbach, D. Manocha. Efficient nearest-neighbor computation for GPU-based motion planning. In: *Proceedings of International Conference on Intelligent Robots and Systems*, 2010: 2243–2248
62. J. Pan, D. Manocha. Bi-level locality sensitive hashing for k-nearest neighbor computation. In: *Proceedings of International Conference on Data Engineering*, 2012: 378–389
63. C. Lauterbach, Q. Mo, D. Manocha. gProximity: Hierarchical GPU-based operations for collision and distance queries. *Comput. Graph. Forum*, 2010, 29(2): 419–428