Constraint-Based Motion Planning Using Voronoi Diagrams

Maxim Garber and Ming C. Lin

University of North Carolina, Chapel Hill, NC, USA
{*garber,lin*}*@cs.unc.edu*
http://gamma.cs.unc.edu/cplan

Abstract. We present a novel algorithm for planning the motion of rigid and articulated robots in complex, dynamic, 3D environments. Our approach is to reformulate the motion planning problem as a simulation of a constrained dynamical system, and guide this system using generalized Voronoi diagrams (GVDs). In our framework, each rigid robot is subject to virtual forces induced by geometric and mechanical constraints. These may include constraints to have a robot follow an estimated path computed using a GVD, constraints to link rigid objects together to represent an articulated robot, or constraints to enforce a spatial relationship between multiple collaborative robots. The resulting algorithm uses all constraint forces to move the robot along an estimated path through the environment, while avoiding collisions with obstacles and enforcing joint and positional constraints. Our algorithm works well in dynamic environments with moving obstacles and is applicable to planning scenarios where multiple robots must move simultaneously to achieve a collision free path.

1 INTRODUCTION

Motion planning is one of most fundamental problems in algorithmic robotics. The classic motion planning problem, also referred to as the Piano Mover's problem, can be stated as the following: Given a robot $\mathcal{R}$ and a workspace $\mathcal{W}$, find a path, P, from an initial configuration $\mathcal{I}$ to a goal configuration $\mathcal{G}$, such that $\mathcal{R}$ never collides with any obstacle $\mathcal{O}_i \in \mathcal{W}$. The path P, if such a path exists, is a continuous sequence of positions and orientations (i.e. configurations) of $\mathcal{R}$. Planning tasks are typically characterized by geometric goal regions, a variety of mechanical and geometric constraints, and often a partially known environment with uncertainties and moving obstacles.

Main Results: In this paper, we present a novel motion planning algorithm for rigid and articulated robots. Our algorithmic framework is based on constrained dynamics [27] used for physically-based modeling. We reformulate the motion planning problem as a dynamical system simulation, where constraints are enforced by virtual forces imposed on the system. Each robot is treated as a rigid body, or a collection of rigid bodies, and is moved subject to all types of constraint forces. These may include constraints that are inherent to the planning scenario, such as constraints that enforce the joint angle limits and link connectivity of articulated robots or constraints that enforce

some required spatial relationships between multiple collaborative robots. In addition to these required constraints, we introduce constraints designed to guide the robot through the environment to the desired goal configuration. Using global geometric analysis from the generalized Voronoi diagram [12] of the workspace, we define constraints that move the robot to avoid both static and moving obstacles, and also follow an estimated path to the goal. The solution to the motion planning problem is the collection of configurations of the dynamical system that satisfy all geometric and mechanical constraints.

Although our current implementation is only designed to handle certain types of planning scenarios, our proposed framework is rather general. The framework can be extended to incorporate many different constraint types and also many different constraint solving methods, such as optimization, local iteration, symbolic algebraic solvers, etc. Our planning planning framework has the following characteristics:

- **Portable:** It can plan paths for both rigid and articulated robots of any topology, including both revolute and prismatic joints, and of any number of degrees of freedom. It can also be extended to handle robots that deform over time governed by the law of physics.
- **Dynamic:** It can plan collision free motion for an object in the presence of moving, or deforming, obstacles, as well as other collaborating robots, whose motion is not known a priori.
- **General:** It allows the user to specify a wide range of relationships between robots as well as between robots and other objects in the scene. Examples of such relationships include: a robot avoiding other moving robots, multiple robots following a leader, and multiple robots collaborating to manipulate an object.

To demonstrate our framework we have implemented a planner for rigid and articulated robots in dynamic environment that runs at interactive rates for many moderately complex scenes. We demonstrate the effectiveness of our planner for virtual assembly and electronic prototyping. We show that it works well in a changing scenes with moving obstacles and multiple collaborating robots, in applications to assembly line planning, automated car painting, and maintainability studies.

Organization: The rest of the paper is organized as follows. In section 2, we briefly survey related work. Section 3 presents an overview of our planning framework. We describe the detailed formulation of various constraints in section 4. Section 5 explains how planning scenarios are represented in our framework. We demonstrate our planner working on several virtual prototyping problems in section 6. In section 7, we offer a conclusion and suggest some future research directions.

2 RELATED WORK

Motion planning has been extensively studied in robotics, computational geometry and computer-aided manufacturing for more than three decades [19]. Most of the earlier work has focused on the Piano Mover's problem.

2.1 Global vs. Local Planning Methods

There have been two class of approaches to motion planning: global and local methods. Global planning methods, including the first Roadmap Algorithm [6], PRM [14,15] and other geometric or "criticality-based" methods [10,19], are guaranteed to find a path, if one exists, although they may take a long time computing it. Many of the global methods, with the exception to variants of PRM, have been applied with limited success for mostly lower-dimensional planning queries in static environments, due to high computational costs.

On the other hand, local methods such as artificial potential field methods [16], are usually fast, but are not guaranteed to find a path, even if one exists. Algorithms based on artificial potential field methods are frequently used in industrial applications [7,21]. In a dynamic environment, where the motion of obstacles in the scene is not know a priori, it is difficult for global methods to compute the complete solution path in real time to avoid collision with the moving obstacles. Local methods have known limitations as well. For example, potential field methods are known for their entrapment problems at local minima of the potential function.

Our method is an incremental construction of a roadmap, whose the curves locally satisfy all constraints imposed on the robot while remaining maximally clear of nearby obstacles. At the same time, to help avoid the local minima problem, the framework takes global geometric analysis of the environment, obtained from the Generalized Voronoi Diagram, into consideration while performing the local planning.

2.2 Voronoi Diagrams in Motion Planning

Generalized Voronoi diagrams (GVDs) have long been used as a basis for motion planning algorithms [5,8–10,13,22,25,26]. The GVD represents the connectivity of a space but has a dimension lower by one, and (in three dimensions) it is composed of surfaces of maximal clearance. Unfortunately, no good and practical algorithms are known for computing the Voronoi diagrams of large environments. In the worst case the complexity of Voronoi diagrams is $O(n^2)$, where n is the number of polygons in the environment. Moreover, the Voronoi diagram of a polygonal environment is composed of quadric surfaces and degree four curves that meet at junctions whose algebraic degree is eight. It is hard to accurately compute an arrangement of these curves and surfaces using fixed precision arithmetic.

Our approach takes advantage of a method proposed in [12] which uses graphics hardware to quickly compute a discretized, error-bounded approximation to the GVD of the workspace, to provide useful information for motion planning.

2.3 Constraint Solving

Geometric constraint solving has been extensively studied in many different fields, such as CAD/CAM, molecular modeling, and theorem proving [3,4,17]. There are two basic strategies: *instance solvers* and *generic solvers*. Some of the common approaches include numerical algebraic techniques, graph-based algorithms, logical interference and term rewriting, symbolic algebraic solvers, and propagation methods [3]. Any one of these techniques could be used in our planning framework.

In our current implementation, we borrow a combination of ideas from some of these techniques, and specializes them for motion planning. The goal of our implementation is to achieve real-time performance for dynamic scenes with moving obstacles and multiple, rigid or articulated, robots. Since, the constraint solving problem for motion planning can be NP-hard for arbitrarily high degree-of-freedom manipulators, we consider the intended solution and infer certain metric and topological properties of the planning problem as a dynamical system, and deduce a few heuristics that succeed with high probability under the assumptions of compatible constraints and temporal coherence. The details of our constraint solving approach will be given in Sec. 4.

3 FRAMEWORK OVERVIEW

Our approach is an opportunistic planning framework powerful enough to handle dynamic environments, and general enough to incorporate many different types of constraints and global analysis to govern an object's motion. It also allows the incorporation of complex relationships between collaborating entities. In this framework even articulated robots are represented using constraints, linking collections of rigid objects. In addition, this framework allows natural extension to planning of flexible robots and incorporation of dynamics, non-holonomic, and other types of constraints.

3.1 Simulation Framework

The basic essence of our framework is to describe each rigid object in the planning scene as a dynamical system, which is characterized by its state variables (i.e. position, orientation, linear and angular velocity). In this framework, a robot can be a rigid body, or a collection of rigid bodies, subject to the influence of various forces in the workspace, and restricted by various motion constraints. This transforms a motion planning problem into a problem of defining suitable constraints, and then simulating the rigid body dynamics of the scene with each constraint acting as a virtual force on the objects. That is, if q is the configuration of the robot at some time t, then each constraint can be represented as a function of q, $C(q)$. The virtual force induced by each constraint is simply $f_c = \frac{-\partial E(C(q))}{\partial q}$ where the energy function, $E(C(q))$, is defined as $E(C(q)) = \frac{k_s}{2} C(q) \cdot C(q)$ and k_s is a generalized stiffness constant. We'll use the following notation for the rest of the paper:

- Let $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$ be a set of m rigid objects.
- For each $\mathcal{R}_i$ at time t, let a state vector $s_i^t = (pos_i^t, rot_i^t, lin_i^t, ang_i^t)$ represent the objects position, rotation, linear and angular velocity.
- Let S^t be the system state vector, obtained by concatenating the state vectors s_i^t for all i.
- Let $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ be a set of n constraints.
- For each constraint $\mathcal{C}_j$, let $F_j(S^t)$ be the force induced by constraint j given the object system state S^t.

The simulation steps from time t to time $t + h$ and updates the state of each object subject to the forces induced by the constraints, using the following steps:

BEGIN LOOP

Update Obstacles: For each dynamic obstacle O_i, update its state in S^t.
Compute Constraint Forces: Summing up all virtual forces, $F_c(S^t) = \sum_{j=1}^{n} F_j(S^t)$.
Update System State: Compute S^{t+h} from S^t subject to $F_c(S^t)$ [27].
Update Object States: For each object $\mathcal{R}_i$, update s_i^{t+h} from S^{t+h}.
Increment Time: $t = t + h$

END LOOP

In this framework the solution to the motion planning problem, for a particular object R_i emerges as the sequence of states, $\{s_i^t, s_i^{t+h}, \ldots, s_i^{t+k*h}\}$, such that the object is in its initial configuration at time t, and achieves the goal configuration at time $t + k * h$. The simulation must run for as many time steps as necessary for all objects for which a planned path is desired, to reach their goal configurations.

3.2 Dynamic Scenes

Since our intent is to handles scenes in which obstacle motion is not known a priori, our planning framework must incorporate an explicit dependence on time. As a consequence, a path computed for a particular time interval may not be valid for the different time interval due to different configurations and trajectories of the obstacles. In the above simulation loop we ensure this interdependence by synchronizing the time step of the planning simulation with the time step that governs the motion of obstacles in the scene. It is important to note that the total number of time steps needed to plan a path through a dynamic scene can vary dramatically with the obstacle motion. This occurs, for example, when the robot must wait for a slow moving obstacle to move out of the way before proceeding to the goal.

3.3 Constraints

To achieve the desired results without robustness problems, we further classify the constraints into soft and hard constraints and treat them differently.

Hard Constraints are those that absolutely must be satisfied at every time step of the simulation. Common examples of hard constraints include:

- objects must stay within the bounds of the scene
- objects must never intersect or penetrate each other %vspace*-0.25em
- links of an articulated robot must remain attached together
- the joint angles in an articulated object must remain within some predefined limit

Soft Constraints serve as guides to *encourage* or influence the objects in the scene to behave in certain ways. They are simulated using penalty forces. Some common examples of soft constraints include:

- an object should move to follow the nearby medial axis of the workspace
- an object should move towards a goal configuration
- an object should move to avoid the nearest obstacles
- an object should move towards goals set by a higher level planner

More details on constraint formulation and constraint solving techniques will be presented in Sec. 4.

3.4 Use of Voronoi Diagrams

In our framework we define constraints that guide the robot towards the planning goal. To obtain these crucial constraints we use information about the environment obtained by Voronoi diagrams. This is done in two ways:

Estimated Paths are used in a manner similar to the basic philosophy taken by [2,8–10,26], to provide global information to the path planner. To obtain this information we use the generalized Voronoi diagram (GVD) of the environment. The GVD computation is accelerated by graphics hardware and can be recomputed as the environment changes. This global information is incorporated into our framework as a constraint that guides the robot along the path computed from the GVD of the environment. The estimated path based on the GVD is not a true collision-free path, but by using other constraints our planner will correct the estimated path, avoiding collisions as the simulation proceeds. This shares the similar theme with OBPRM [1].

Obstacle Avoidance is accomplished using a by-product of the GVD computation. This byproduct is a discretized distance field of the workspace. This distance field can be recomputed in real time around the regions of interests, using hierarchical bounding boxes and parallel computing on today's graphics hardware [12]. It is used to perform distance computation, as well as quick rejection tests, which can be followed by exact collision detection if needed. Using this distance field, we can also define soft constraints to maintain a minimum clearance between a robot and its environment. The penalty forces that result from this constraint will tend to align the robot to the medial axis of the environment.

4 CONSTRAINT FORMULATION

4.1 Solving The Constraints

We use two main constraint solving techniques in the current implementation of our planning framework: penalty forces and iterative relaxation. Penalty

forces are used to represent soft constraints, while iterative relaxation is used to enforce hard constraints. Other constraint solving techniques can also be appropriately incorporated. Since we want the object states to satisfy all hard constraints at the end of each time step, we perform relaxation on the hard constraints after applying penalty forces based on soft constraints.

Applying Penalty Forces: We use penalty forces to apply soft constraints to the rigid objects in a planning scene. As we will describe in Sec. 4.3, each type of soft constraint, $\mathcal{C}_i$, has a method $Get\_Penalty\_Force(\mathcal{C}_i, \mathcal{R}_j, S^t, t)$ which returns the force and torque generated by the constraint $\mathcal{C}_i$, on object $\mathcal{R}_j$, at time t, with the scene in the state S^t. The total force on an object is the sum of all penalty forces acting on that object. To update the object state for each object, we integrate this total force using the Midpoint Method [27]. We use this method because some of the soft constraint penalty forces are computationally intensive to evaluate, and the midpoint method provides stable integration with only two force evaluations per time step.

Iterative Relaxation: Once objects in the scene are updated as a result of the penalty forces due to the soft constraints, their state may violate one or more of the hard constraints. For each hard constraint, $\mathcal{C}_h$, we define the residual, $Res(\mathcal{C}_h, S^t)$, to be a real number which represents the degree to which $\mathcal{C}_h$ is violated when the objects are in state S^t. For each type of hard constraint $\mathcal{C}_h$, we require an instance solver $Relax(\mathcal{C}_h, S^t)$, to be described in Sec. 4.2, which returns a new state, S, in which $Res(C_h, S) = 0$. To efficiently ensure that these constraints are satisfied, we use the well known Nonlinear Gauss-Siedel method [23]. The iterative relaxation method, described in Algorithm. 1, relaxes each constraint in sequence repeatedly, until the objects converge to a state for which the sum of the residuals, over all constraints, is zero.

Relax_Constraints

Input The state S^t, at time t, of all rigid objects, points and vectors in the simulation, the set $\mathcal{C}$ of all hard constraints.
Output New state vector S which satisfies all hard constraints.

Let $S \leftarrow S^t$.
Repeat:{
  for each hard constraint $\mathcal{C}_i$: {
    $S \leftarrow Relax(C_i, S)$.
  }
}**until** $\sum_{i=0}^{n} |Res(C_i, S)| = 0$
return The state S.

ALGORITHM 1: Relax Hard Constraints

Convergence of the Relaxation Method: Our framework allows the use of many different constraint solving techniques. The reasons that we choose, in our current implementation, to use the iterative relaxation method introduced in Sec. 4.1, instead of other techniques (e.g. Lagrangian formalism) to satisfy the hard constraints, is the method's simplicity and because it allows our implementation to achieve interactive performance in most practical scenarios This rapid convergence can be attributed to two factors: temporal coherence and compatible constraints.

Our system is able to take advantage of temporal coherence because, as a physical simulation, it uses small time steps during which the objects in the scene move very little. Moreover, if we assume that all hard constraints are satisfied at the beginning of a time step, the fact that the object motion due to soft constraints is small ensures that the iterative method starts in an initial configuration that is near a valid configuration, if one exists. This all but ensures that the iterative method converges to the solution. It also allows convergence to take place in a relatively small number of iterations, providing stable interactive performance in many practical scenarios. Of course, it is possible for the method to never converge when, for example, there are two incompatible constraints, such that satisfying one necessarily violates the other. This situation does not typically occur in practice because the hard constraints are defined so that they are initially compatible with each other.

4.2 Hard Constraints

To ensure that the simulation enforces the high level hard constraints, such as those mentioned in Sec. 3.3, we use three atomic hard constraints in our current implementation:

- Non-Penetration Constraints
- Point Distance Constraints
- Point Planar Angle Constraints

For each type of hard constraint we will now describe the instance solver $Relax(\mathcal{C}_h, S^t)$. This solver is used in the iterative algorithm of Sec. 4.1 to enforce all of the constraints in the scene. For the constraints that act on rigid objects the solver updates the state, including positions and orientations, of the objects; while for point (distance or angle) constraints, the solver modifies only the positions of the points.

Non-Penetration Constraints: Assuming that all rigid objects in the scene represent closed volumes, we consider a non-penetration constraint between two objects to be satisfied as long as their volumes are disjoint. In most cases, this constraint can be weakened to only require that the objects' boundaries do not penetrate each other. Due to the importance of this constraint for correct motion planning, we use an in-house proximity (collision) query package, PQP [11,18], to detect when two objects penetrate. The residual for a non-penetration constraint is then just 0 if the object are disjoint

and 1 if the objects are not. The problem of separating objects that penetrate, in a physical simulation, is one that has been addressed in many ways [27]. The approach that we use to implement the *Relax* solver for non-penetration constraints is the impulse-based rigid body dynamic simulation [27]. The advantages of this method is that objects are guaranteed to be disjoint at the end of every time step, and that objects rebound from collisions in the most natural possible way. In most planning scenarios, non-penetration constraints should be applied between all objects in the scene, ensuring that the objects behave as if they are solid, although it is possible to have objects that are selectively solid, or completely permeable, if desired.

Point Distance Constraints: These constraints enforce a fixed separation between pairs of points. Thus at a time t, given:

- $p_1 \in \mathcal{R}^3$, and its world transformation T_1^t
- $p_2 \in \mathcal{R}^3$, and its world transformation T_2^t
- $d \in \mathcal{R}$, the constraint distance.

the constraint is satisfied when $dist(T_1^t(p_1), T_2^t(p_2)) = d$, where $dist()$ is the Euclidean distance. Given this formulation of a point distance constraint we define the residual of the distance constraint as:

$$Res(C_{dist}, S) = \Delta_d = dist(T_1^t(p_1), T_2^t(p_2)) - d, \tag{1}$$

where Δ_d represents the linear distance that the two points must travel to reach the required separation. To solve the distance constraint we simply move each point a straight line distance of $\Delta_d/2$ towards each other.

Point Planar Angle Constraints These constraints enforce the angle between two points, about a specified axis of rotation. To define these constraints we require, at time t:

- $p_1 \in \mathcal{R}^3$, and its world transformation T_1^t
- $p_2 \in \mathcal{R}^3$, and its world transformation T_2^t
- $o \in \mathcal{R}^3$ the origin of the joint, and its world transformation T_o^t
- $\overrightarrow{axis} \in \mathcal{R}^3$, axis of rotation for the joint, and its world transformation T_a^t
- $\theta_{min}, \theta_{max} \in \mathcal{R}$, the angle limits.

We define the angle θ as the planar angle between the vectors $T_1^t(p_1) - T_o^t(o)$ and $T_2^t(p_2) - T_o^t(o)$ in the plane normal to $T_a^t(\overrightarrow{axis})$. The constraint is satisfied as long as θ is in the interval $[\theta_{min}, \theta_{max}]$. Given this formulation of a point planar angle constraint between points the residual of the angle constraint is then defined as:

$$Res(C_{ang}, S) = \begin{cases} \theta - \theta_{max} & \text{if } \theta > \theta_{max} \\ \theta - \theta_{min} & \text{if } \theta < \theta_{min} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

To satisfy the angel constrain, if $\Delta_\theta = Res(C_{ang}, t) \neq 0$, the point $T_1^t(p_1)$ is rotated an angle $\Delta_\theta/2$, and $T_2^t(p_2)$ rotates an angle $-\Delta_\theta/2$ about the rotation axis, $T_a^t(\overrightarrow{axis})$.

Relating Constraints: Unlike the non-penetration constraints, the point distance and point planar angle constraints are independent of the object geometry; they instead enforce relationships between points and vectors defined in the scene. These points and vectors can be fixed in world coordinates or expressed relative to the coordinate frames of objects in the scene. In Sec. 5.1 we will address how these point constraints can be combined to achieve the high level hard constraints described in Sec. 3.3.

To allow these constraints between point and vectors to influence the behavior of objects in the scene we use a rigid structure, such as a tetrahedron, which we define using point distance constraints, to represent the coordinate frame of each rigid object. We chose four linearly independent points in the object's coordinate system, and set distance constraints between them to enforce their initial separations. When the object is transformed the rigid structure is also transformed and its distance constraints remain trivially satisfied. At the same time, as long as the constraints that define the rigid structure are satisfied we can uniquely determine the object's transformation from the world locations of the four points of the rigid structure.

To constrain the relative motion of objects in the scene we define constraints between the points of their rigid structures. Thus the order in which the hard constraints are relaxed in Alg. 1 is not arbitrary. At each frame of the simulation the constraints between all of these points are enforced first, possibly changing their locations. Then, from these new point locations, we update the world state of the associated rigid object to a state that respects the constraints. Only after the point constraints have been solved and the associated rigid body states' have been updated do we relax any constraints between the rigid objects themselves.

4.3 Soft Constraints

Soft constraints, in our framework, are constraints that generate penalty forces to *guide* the motion of objects without imposing strict motion restrictions. The types of soft constraints in our current system include:

- Surface repulsion
- Goal attraction
- High-level path following

For each soft constraint, we will describe the function that is used to generate a force along the gradient vector of the constraint.

Surface Repulsion: These constraints use local proximity information, computed as part of the GVD calculation, to allow objects to be repelled from the surfaces of nearby obstacles. To define a surface repulsion constraint for a rigid body, $\mathcal{R}_1$, relative to a second rigid object $\mathcal{R}_2$, we specify distance threshold, δ, and a force coefficient $k \in [0, 1]$. In our method of assigning an estimated path for a robot, $\mathcal{R}_1$, using Voronoi information, the high level planner is able to provide an associated minimum distance tolerance along

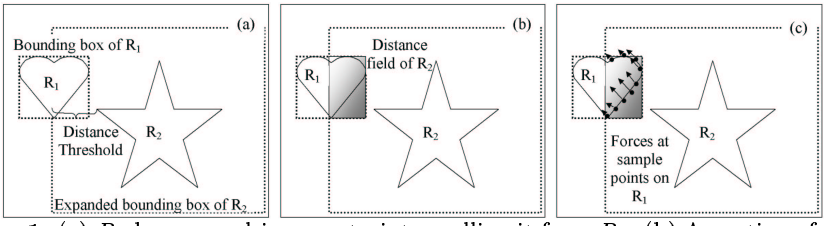

Fig. 1. (a) R_1 has a repulsion constraint repelling it from R_2. (b) A portion of the distance field of R_2. (c) Forces act on R_1 in the gradient direction of the distance field.

the path, which can be used to initialize δ. The priority, k, specifies the relative importance of repulsion constraints, so that if $\mathcal{R}_1$ is trapped between two obstacles, it will give more priority to evading one than the other. Soft constraint priorities are set equal to 1, unless the user specifies another value.

To apply the constraint, we first perform some computations using axis-aligned bounding boxes as approximations for the objects involved. For object $\mathcal{R}_2$ we take the axis-aligned bounding box and expand it by the distance threshold, δ, as shown in Fig. 1 (a). We intersect this expanded bounding box with the bounding box of $\mathcal{R}_1$ to perform a quick rejection test to determine if the two objects are further than the distance threshold δ apart. If this is the case, then we can terminate the computation. If the bounding box test fails, we compute the intersection, call it I, of the two bounding boxes. We then use a hardware accelerated distance field computation [12], a by-product of the GVD computation, to generate the distance values for the surface features of object $\mathcal{R}_2$ in the region I as shown in Fig. 1 (b). The fact that this computation is performed using graphics hardware enables the distance field of the object to be generated in real time without any precomputation or assumptions about the geometry. We intersect this distance field with sample points on the surface of object $\mathcal{R}_1$. For each sample point that lies in I, we check the distance from that point to the nearest point on the surface of $\mathcal{R}_2$ by referencing the distance field. A force is generated, at each sample point, that is in the direction of the gradient of the distance field, proportional to the distance between that sample point and the surface of $\mathcal{R}_2$, as seen in Fig. 1 (c). This force should be zero for sample points beyond the distance threshold, and increase to infinity as the distance between the surfaces decreases. In our implementation the force at each sample point, p_i, is given by:

$$force(p_i, \mathcal{R}_2) = \begin{cases} \frac{\delta^4}{dist(p_i, \mathcal{R}_2)^4} - 1 \text{ if } dist(p_i, \mathcal{R}_2) < \delta \\ 0 \qquad\qquad\qquad\quad \text{otherwise} \end{cases} \tag{3}$$

The force induced by this constraint on $\mathcal{R}_1$ is the sum of all forces on all l sample points on $\mathcal{R}_1$. This is a force that moves $\mathcal{R}_1$ away from $\mathcal{R}_2$, enforcing the surface repulsion constraint.

Goal Attraction: If the robot has not reached its goal configuration, this constraint generates a penalty force that attracts the robot towards its goal.

This constraint could be applied to one component of an articulated robot, such as the end effector, or to multiple components of the robot.

High Level Path Following Using GVDs: One well-known problem with using the penalty forces to achieve planning goals is that the robot can be caught in a local minima and fail to reach the goal. To address this issue we integrate global geometric analysis, generated by a high-level task planner [20], into our planning framework. There are many well known techniques for obtaining an estimated path for a robot based on the static obstacles in the scene, such as a medial axis based planner [10,12], a Probabilistic Roadmap Planner [14], a binary space partitioning of the workspace [21], or simply by taking input from a user [7]. Any of these can be integrated into our planning framework.

In our current implementation, we use a high level path planner based on the GVD [10]. As described in Sec. 3.4 this planner can be utilized throughout the planning simulation to provide updated estimated paths for the robot. Although this is not a complete solution, it does ensure that the robot is not easily trapped in a local minimum of the constraint force fields. The object's orientation along the estimated path is left arbitrary, so that it can be determined by other soft constraints.

5 SCENE REPRESENTION

In this section we will briefly outline how to render the given geometry and other sensory input into hard and soft constraints in the constraint-based planning framework. Assume that the geometry representing the robots and obstacles is given, as well as prescribed motion, simulated or scripted, for the obstacles over time. Our system then defines constraints that will restrict the motion of the robots to meet the design specifications, and also guide the robots to complete the planning tasks.

5.1 Hard Constraints and Articulated Robot Joints

We will now illustrate how the connectivity and angle limit constraints of an articulated object can be represented with a combination of point constraints as described in Sec. 4.2. We rely on the ability to use constraints defined between points of a rigid structure of point distance constraints, that represents the coordinate frame of each object, to enforce relationships between the objects themselves, as described in Sec. 4.2.

Ball Joint Example: Suppose that we have two rigid objects, $\mathcal{R}_1$ and $\mathcal{R}_2$, and wish to constrain them to only two relative rotational degrees of freedom, i.e. a ball joint, about some world point o between them. We would then define $p_1 = o$ in the coordinate frame of $\mathcal{R}_1$, and $p_2 = o$ in the coordinate frame of $\mathcal{R}_2$. Then we would link p_1 and p_2 to the rigid structures of $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively, using three linearly independent distance constraints

each. These constraints ensure that p_1 is rigidly attached to the structure representing $\mathcal{R}_1$ and p_2 is rigidly attached to the structure representing $\mathcal{R}_2$. We then define a distance constraint, with constraint distance of 0, between p_1 and p_2, so that their world positions are constrained to coincide. This ensures that when all constraints are satisfied the two objects, $\mathcal{R}_1$ and $\mathcal{R}_2$, are rigidly attached to a common rotation point at o, whose position is now expressed in the coordinate frames of the two objects, as shown in Fig. 2.

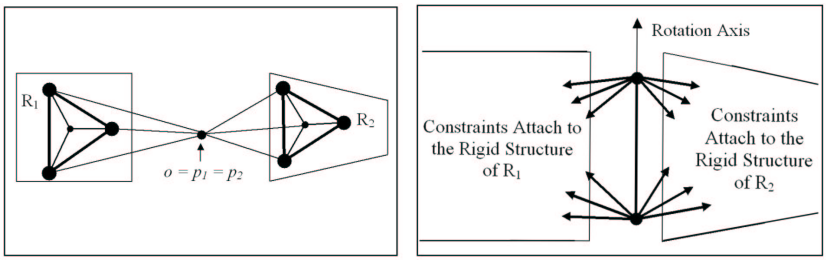

Fig. 2. LEFT: A ball joint, built from distance constraints linking the rigid structures of the two objects. RIGHT: A revolute joint, built from distance constraints linking the rigid structures of the two objects.

Revolute Joint Example: We can extend our formulation of a ball joint to obtain a revolute joint between two rigid objects from the specification of the joint location, axis of rotation, and angle limits. Given such a specification, it is possible to automatically generate the required point distance and planar angle constraints to limit the robot's degrees of freedom to only allow rotation about the joint axis, within the angle limits. To define a revolute joint between two rigid objects, $\mathcal{R}_1$ and $\mathcal{R}_2$, we use two ball joints. As long as the two ball joints have distinct centers of rotation that lie on the intended rotation axis, the constraints will limit the rigid objects to only one relative rotational degree of freedom about the axis. For additional stability we define a redundant distance constraint between the two rotation centers to maintain their separation along the rotation axis.

To limit the angle of the revolute joint, we use a point planar angle constraint, defined to limit the angle about the rotation axis between one point attached to the rigid structure of R_1, and one point attached to the structure of $\mathcal{R}_2$.

There has been previous work, in the field of robot control, on specifying joints using simple constraints [24]. The advantage of our approach is that, using the rigid structure formulation, we reduce the problem of solving constraints between rigid objects to a much simpler problem of solving constraints between points. As we have shown in Sec. 4.1, this allows for a fast and stable constraint solving method to be used.

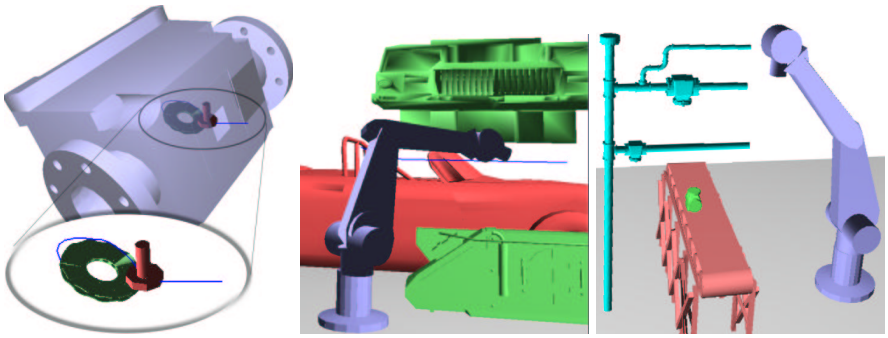

Fig. 3. Left to right: (a) Maintainability Study (Scene 1); (b) Automated Car Painting (Scene 2); (c) Assembly Line Planning (Scene 3).

5.2 Soft Constraints

While the specifications for all hard constraints are provided as part of the specific planning scenario, the soft constraints are defined by our framework to achieve the planning task. In our method, we automatically define soft constraints to both propel the robots toward their goals, and ensure that they do not collide with obstacles. It should be clear from our discussion of soft constraints in Sec. 4.3 how these constraints can be applied to achieve the desired behavior. To move the robots in the scene towards their planning goals we use one of two soft constraints: goal attraction and path following. To accomplish obstacles avoidance, we use surface repulsion constraints defined between the robots and all of the objects in the scene.

6 IMPLEMENTATION AND PERFORMANCE

6.1 Implementation

Our system was implemented in an object-oriented framework using C++. We use the Proximity Query Package [11,18] for collision detection, to enforce non-penetration constraints, and HAVOC3D [12] to generate distance fields for surface repulsion constraints.

6.2 System Demonstration

We have tested our motion planning system on following applications:

Scene 1: Maintainability Study

In assembly maintainability studies, motion planning is used to find whether it is possible to remove a particular part from an assembly, and if so, to find one possible removal path [7]. In our example, shown in Fig. 3(a), a bolt and a washer must avoid each other in the confines of tight compartment inside a pump assembly. The goal, to remove the bolt from the assembly, requires both objects to maneuver around each other without colliding.

Scene 2: Automated Car Painting

In this example seen in Fig. 3(b), an articulated robot arm, with 6 degrees of freedom, is used to trace a path along the body of a car for painting. The robot is composed of rigid components that are held together by constraints. For all of the components of the robot, the planner must compute paths that satisfy the joint constraints, do not collide with the obstacles or the car, and lead the end effector along the prescribed path.

Scene 3: Assembly Line Planning

In this example, shown in Fig. 3(c), the robot arm from scene 2 must access a part moving past it on a conveyer belt. The factory floor contains a piping structure that is moving over the conveyer belt in the opposite direction to the part's movement. The moving obstruction causes the robot to reactively modify its path to avoid collision.

The timings for these scenarios are presented in Table 1. The timings were taken on a PC with a 933MHz Pentium III processor, 256MB RAM and an nVidia GeForce3 graphics card. The motion sequences captured in MPEG are available at: *http://gamma.cs.unc.edu/cplan.*

Scene	Poly	Cons	**Per Step**	**Total**
(1) Maintainability	20470	4	0.093 sec	67 sec
(2) Auto Painting	25738	43	0.038 sec	18 sec
(3) Assembly Line	16962	43	0.0085 sec	16 sec

Table 1. Benchmark timings in seconds on three example scenes. *Poly:* The number of polygons in each scene. *Cons:* The total number of active constraints in each scene. *Per Step:* The average time for the planner to compute one time step of the simulation. *Total:* The total time taken to complete and execute the planning task.

6.3 Discussion

The planning tasks in the example scenes execute, on average, between 10 and 120 time steps per second. The primary bottleneck in our current implementation is the distance field computation used to determine estimated paths and the penalty forces for the surface repulsion constraints. We use a one-level bounding box culling to limit the application of this computation to areas near potential surface collisions. We also use simplified geometry for computing the distance field wherever appropriate to speed up the proximity queries. This approach works well, unless the scene, as in the case of Scene 1, has highly non-convex complex geometry that is poorly approximated by the bounding boxes. In such cases, hierarchical bounding box culling could be used to further limit the application of the distance field computation to increase runtime performance. We are currently working on this optimization, as well as accelerating the 3D distance field computation. With the new optimized implementation, we expect at least an order of magnitude performance improvement.

The constraint solver we have developed for the current system uses an iterative relaxation method that is specialized to provide interactive performance when planning the motion of rigid and articulated robots in dynamic

scenes. It works well for overconstrained and consistent systems, such as those produced by our method of modeling robot joints using constraints, and in our planning framework where the dynamic simulation typically advances in small time steps allowing it to take advantage of temporal coherence to achieve performance and stability. However, it is possible for our framework to incorporate other efficient constraint solvers based on the non-trivial extension of [3,4,17], as we extend this work to plan the motion of flexible bodies and to also include different types of geometric and dynamics constraints in our system.

7 CONCLUSION AND FUTURE WORK

We reformulate the motion planning problem into a physical simulation where constraints on the robot's motion and the GVD of the workspace guide the robot from its starting to its goal configurations. We have demonstrated its effectiveness and real-time performance on several challenging planning scenarios. The flexibility of our framework offers the possibility of natural extension, including: (a) additional constraints between robots, including constraints for maintaining line of sight contact or other complex geometric relationships, as well as dynamic and non-holonomic constraints; (b) extension to flexible geometry that deforms over time due to contact; and (c) incorporation of direction human interaction and control.

Acknowledgments: This research was supported in part by by ARO DAAG55-98-1-0322, NSF DMI-9900157, NSF IIS-9821067, NSF ACR-00118743, ONR N00014-01-1-0067 and Intel.

References

1. N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. *Proceedings of WAFR98*, pages 197–204, 1998.
2. O. B. Bayazit, J.M. Lien, and N. M. Amato. Probabilistic roadmap motion planning for deformable objects. *Proceedings of International Conference on Robotics and Automation*, pages 2126–2133, 2002.
3. W. Bouma, X. Chen, I. Fudos, C. Hoffmann, and P. Vermeer. *An Electronic Primer on Geometric Constraint Solving.* http://www.cs.purdue.edu/homes/cmh/electrobook/intro.html, 1990.
4. B. Bruderlin and D. Roller (eds). *Geometric Constraint Solving and Applications.* Spring Verlag, 1998.
5. J. F. Canny and B. Donald. Simplified voronoi diagrams. *Discrete and Computational Geometry*, 3:219–236, 1988.
6. J.F. Canny. *The Complexity of Robot Motion Planning.* ACM Doctoral Dissertation Award. MIT Press, 1988.
7. H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proceedings of International Conference on Robotics and Automation*, 1995.
8. H. Choset and J. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. *IEEE Conference on Robotics and Automation*, 1995.

9. H. Choset and J. Burdick. Sensor based planning: The hierarchical generalized voronoi graph. *Workshop on Algorithmic Foundations of Robotics*, 1996.
10. M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.
11. S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.
12. K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.
13. K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. *IEEE Conference on Robotics and Automation*, pages pp. 2931–2937, 2000.
14. L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. *IEEE Conference on Robotics and Automation*, pages 2138–2145, 1994.
15. L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580, 1996.
16. O. Khatib. Real-time obstable avoidance for manipulators and mobile robots. *IJRR*, 5(1):90–98, 1986.
17. G. Kramer. *Solving Geometric Constraint Systems: A case study in kinematics.* MIT Press, 1992.
18. E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, 2000.
19. J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.
20. T. Lozano-Perez and R. Wilson. Assembly sequencing for arbitrary motions. *Proc. IEEE International Conference on Robotics and Automation*, 1993.
21. K. Ahrentsen N. Jacobsen, R. Larsen, and L. Overgaard. Automatic robotwelding in complex shipstructures. *J. Applied Artificial Intelligence*, 1997.
22. C. Ó'Dúnlaing, Micha Sharir, and C. K. Yap. Retraction: A new approach to motion-planning. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 207–220, 1983.
23. J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, 1970.
24. L Overgaard, H. Petersen, and J. Perram. A general algorithm for dynamic control of multilink robots. *Int. J. Robotics Research*, 14(3), 1995.
25. C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.
26. Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, 1999.
27. A. Witkin and D. Baraff. *Physically Based Modeling: Principles and Practice.* ACM Press, 1997. Course Notes of ACM SIGGRAPH.