Constraint-Based Motion Planning for Virtual Prototyping

Maxim Garber
Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/∼garber
garber@cs.unc.edu

Ming C. Lin
Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/∼lin
lin@cs.unc.edu

ABSTRACT

We present a novel framework for motion planning of rigid and articulated robots in complex, dynamic, 3D environments and demonstrate its application to virtual prototyping. Our approach transforms the motion planning problem into the simulation of a dynamical system in which the motion of each rigid robot is subject to the influence of virtual forces induced by geometric constraints. These constraints may enforce joint connectivity and angle limits for articulated robots, spatial relationships between multiple collaborative robots, or have a robot follow an estimated path to perform certain tasks in a sequence. Our algorithm works well in dynamic environments with moving obstacles and is applicable to challenging planning scenarios where multiple robots must move simultaneously to achieve a collision free path. We demonstrate its effectiveness for parts removal, automated car painting, and assembly line planning scenarios.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms

Algorithms, Performance, Design, Reliability, Verification

Keywords

Virtual Environments and Prototypes, Manufacturing and Assembly Planning, Computational Support for New Manufacturing Technologies

1. INTRODUCTION

The problem of achieving design flexibility and manufacturing automation using virtual environments provides a new set of challenges for computer aided design and manufacturing. Automatically planning the motion of parts or objects, i.e. motion planning, can be a significant aid in a rapid prototyping environment. Examples of the tasks that can be assisted by motion planning include virtual assembly for design verification, parts removal for maintainability studies, path generation for automated painting, etc. Some of the commonly used computer-aided manufacturing and simulation packages like IGRIP from Delmia Inc., CimStation from Adept Technologies, PDMS from CADCENTRE, ProductVision from GE Corporate R & D, and THOR Arc Weld from AMROSE include computational support for motion planning. These packages are already used for pipe routing in plant design, arc welding of complex assemblies, spot welding of car bodies, and other uses of robots to automate the manufacturing processes. The planning tasks are typically characterized by geometric goal regions, a variety of mechanical constraints, and often a partially known environment with uncertainties and moving obstacles.

Main Results: In this paper, we present a new motion planning algorithm for virtual prototyping. Our algorithmic framework is inspired by constrained dynamics [26] in physically-based modeling. We transform the motion planning problem into a dynamical system simulation by treating each robot as a rigid body or a collection of rigid bodies moving under the influence of all types of constraint forces in the virtual prototyping environment. These may include constraints to enforce joint connectivity and angle limits for articulated robots, constraints to enforce a spatial relationship between multiple collaborative robots, constraints to avoid obstacles and self-collision, or constraints to have the robot follow an estimated path to perform certain tasks in a sequence.

Our constraint-based planning framework has the following characteristics:

- It can handle both static environments with complete geometric information or dynamic scenes with moving obstacles whose motion is not known a priori.
- It is applicable to both rigid and articulated robots of arbitrarily high degrees of freedom, as well as multiple collaborative agents.
- It allows specification of various types of geometric constraints.
- It runs in real time for modestly complex environments.

We demonstrate the effectiveness of our framework for the problem of virtual assembly and electronic prototyping with applications in assembly line planning, automated car painting, and maintainability studies.

Organization: The rest of the paper is organized as follows. In section 2, we briefly discuss the problem of motion planning and survey related work. Section 3 presents an overview of our planning framework. We describe the constraints in our framework and how they can be used to represent planning scenarios in section 4. Section 5 presents a detailed discussion of the method we have implemented to solve the geometric constraints. We demonstrate our

SM'02, June 17-21, 2002, Saarbrucken, Germany.

framework by applying it to several virtual prototyping problems in section 6.

2. RELATED WORK

We first introduce the terminology used in this paper, provide some background information on motion planning, and survey related literature.

2.1 Background and Terminology

We assume the robot(s) and obstacles are sets of closed and bounded geometry and whose locations at any time during the simulation are known and updated dynamically. The obstacles are free to move during the robot planning and motion execution stage, their motion acquired by sensory input is then fed back to the motion planner in real time. The robot $\mathcal{R}$ and a set of obstacles $\mathcal{O}$ move in a Euclidean space $\mathcal{W}$, called the *workspace*. The robot may be a free-flying rigid object or an articulated object. A free-flying object has no kinematic constraints that limit its motion. On the other hand, an articulated object $\mathcal{R}$ consists of several moving rigid parts $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$, called links, connected by joints. Each joint constrains the relative movements of the two links it connects.

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 from an initial configuration $\mathcal{I}$ to a goal configuration $\mathcal{G}$, such that $\mathcal{R}$ never collides with any obstacle $\mathcal{O}_i \in \mathcal{O}$ in $\mathcal{W}$ along the path P, if such a path exists. The path P is a continuous sequence of positions and orientations of $\mathcal{R}$.

Configuration space or C-space [20] is a powerful concept for the planning problem. This formulation represents the robot as a point in an appropriate space, i.e. the robot's configuration space, and maps the obstacles in this space. This mapping transforms the motion planning problem for an object into the problem of planning the motion of a point in a higher dimensional space. For a rigid body, the configuration is specified by six coordinates, three determining the position with respect to some fixed reference point (the *origin*) on $\mathcal{R}$, and three determining the robot's orientation. An articulated object can be interpreted as a set of m moving rigid objects connected by joints. By convention, each joint affords a single degree of freedom, and a physical joint that allows more than one degree of freedom is represented by multiple joints at a single location, so that there can be more joints than links. With this convention, a configuration $\mathbf{cf}$ for an articulated body is specified by six coordinates determining the position and orientation of a given link, the *base link*, along with an additional coordinate for each joint.

2.2 Global vs. Local Planning Methods

There have been two major approaches to motion planning: global and local methods [19]. Global planning methods, including the first Roadmap Algorithm [4], PRM [15] and other geometric or "criticality-based" methods [10, 19], are guaranteed to find a complete 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 widely used in many industrial applications [5, 23]. 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. For dynamic scenes, local, or reactive, planning techniques are often used. However, local methods have 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 staying maximally clear of nearby obstacles. At the same time, the framework can also take global geometric analysis into consideration while performing local planning, thus alleviating the local minima problem (Sec. *4.3.3*).

2.3 Planning for Industrial Applications

The basic motion planning problem can be enhanced by adding a model of position uncertainty to extend it to motion planning in a partially known environment, often encountered among industrial applications. But, it remains basically unchanged if compliant motion is not allowed.

Many specialized motion planning algorithms have been developed for different applications, including part orientation and positioning [9, 11], assembly sequencing [8, 21, 13], sensor-based planning [22, 6, 7] and maintainability studies [1, 5].

2.4 Constraint Solving

Geometric constraint solving has been extensively studied in many different fields, such as CAD/CAM, molecular modeling, and theorem proving [2, 3, 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 [2]. Our approach borrows a combination of ideas from some of these techniques, and specializes them for motion planning.

The constraint solving problem for motion planning can be NP-hard for arbitrarily high degree-of-freedom manipulators. Since we are interested in real-time performance for dynamic scenes with moving obstacles and multiple robots, we consider the intended solution by inferring 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. 5.

3. FRAMEWORK OVERVIEW

3.1 Framework Goals

Our planning framework is targeted to enable rapid prototyping in a 3D CAD/CAM environment. This goal imposes several design requirements. The framework must be:

- **Portable:** It should be able to plan paths for both rigid and articulated robots of any topology or arbitrarily high degrees of freedom, without any change to the underlying system.
- **Dynamic:** It should be able to plan collision free motion for an object in the presence of moving obstacles, and other collaborating robots, whose motion is not known a priori.
- **General:** The system must allow the user to easily specify a wide range of relationships and behaviors between the robots and other objects in the scene.
- **Interactive:** The system must run at interactive rates and allow the user to trade execution speed for accuracy, to enable rapid design and path verification.

These design goals suggest a fusion of global and local planning techniques to capitalize on the benefits of both. We propose a

constraint-based planning approach that provides a local planning framework powerful enough to handle dynamic scenes, efficient enough to run at interactive rates, and general enough to incorporate many types of geometric constraints to govern an object's motion. These constraints can represent complex relationships between collaborating entities and also link collections of rigid objects to behave as articulated robots. This framework also allows natural extension to planning of flexible robots and incorporation of dynamics, as well as non-holonomic and other types of constraints.

3.2 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. We will return to the problem of defining constraints to solve a planning problem in Sec. 4.

Next we'll explain the simulation framework with the following notation:

- 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.

BEGIN LOOP

Compute Constraint Forces: Summing up all contributing forces, $F(S^t) = \sum_{j=1}^{n} F_j(S^t)$.

Update System State: Compute S^{t+h} from S^t subject to the force $F(S^t)$ [26].

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.

4. PLANNING SCENE FORMULATION

In this section we will describe how to render a planning scenario in the form of constraints for 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.

4.1 Constraint Classification

To achieve the desired results without robustness problems, we classify the constraints into two categories: soft and hard constraints.

Hard Constraints are those that absolutely must be satisfied at every time step of the simulation. Examples of the high level hard constraints include object non-penetration, articulated robot joint connectivity, and articulated robot joint angle limits.

Soft Constraints serve as guides to *encourage* or influence the objects in the scene to behave in certain ways. Some common examples of soft constraints include having an object move towards a goal configuration, avoid the nearest obstacles, and move along some predefined path. Soft constraints are the more difficult type to handle because there can be many competing ones acting on an object. We provide several methods to resolve such conflicts. First, the soft constraint penalty force scales with the degree to which the soft constraint is violated. Second, each soft constraint is given a priority, from $[0, 1]$, which scales the constraint force.

4.2 Hard Constraints

To ensure that the simulation enforces the high level hard constraints we use three atomic hard constraints:

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

4.2.1 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. 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.

Unlike the non-penetration constraints, the second and third categories of hard 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.

4.2.2 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.

4.2.3 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

- $\overrightarrow{axis} \in \mathcal{R}^3$, the axis of rotation for the joint, and its world transformation T_a^t
- $\theta_{min}, \theta_{max} \in \mathcal{R}^3$, 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 $[\theta_{min}, \theta_{max}]$, see Fig. 1.

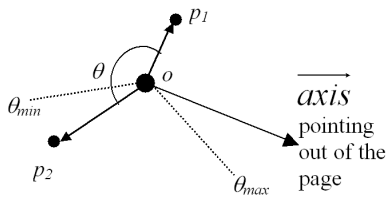

Figure 1: An angular constraint between two points.

4.2.4 Combining Point Constraints

In this section we will illustrate how point constraints can be combined to link rigid objects to form articulated objects. Our method uses 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 distances 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. At each frame of the simulation the constraints between all of these points are enforced, possibly changing their locations. From these new point locations we update the world state of the associated rigid object to a state that respects the constraints.

Example: A Ball Joint 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, see Fig. 2.

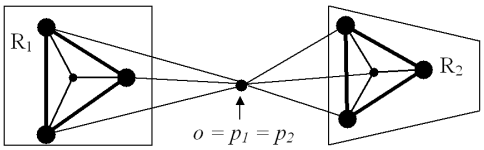

Figure 2: A ball joint, built from distance constraints linking the rigid structures of the two objects.

Example: A Revolute Joint We can extend the formulation for 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. 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 rigid structure of $\mathcal{R}_2$.

There has been previous work, in the field of robot control, on specifying joints using constraints [25]. 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 will show in Sec. *5.1.2*, this allows for a very fast and stable constraint solving method to be used.

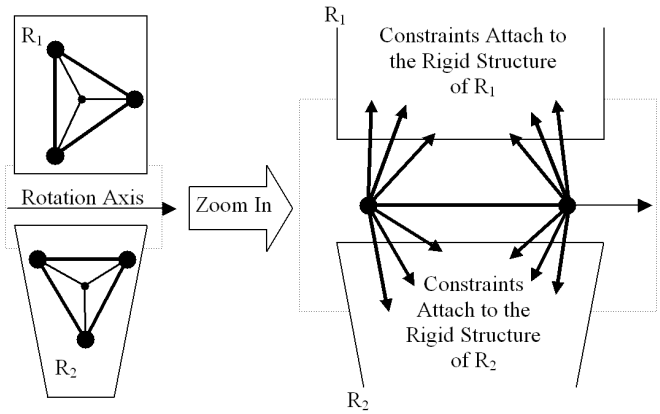

Figure 3: A revolute joint, built from distance constraints linking the rigid structures of the two objects.

4.3 Soft Constraints

Soft constraints in our framework are user specified 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 are:

- Goal Attraction
- Surface repulsion
- Path following

4.3.1 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.

4.3.2 Surface Repulsion

These constraints are used to have the robot be repelled from the surfaces of nearby obstacles. To define a surface repulsion constraint for a rigid body, $\mathcal{R}$, relative to a second rigid object O, we specify a distance threshold, δ, and a force coefficient $k \in [0, 1]$. The distance threshold is the minimum distance allowed between the objects before the $\mathcal{R}$ acts to move away from O. If $\mathcal{R}$ is following an estimated path, often the path will have an associated minimum distance tolerance for static obstacles, which can be used

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

4.3.3 Estimated Paths

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 ahead of time by a high-level task planner [21], 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, 14], a Probabilistic Roadmap Planner [15], a binary space partitioning of the workspace [23], or simply by taking input from a user [5]. A path from any of these can be integrated into the simulation using a path following constraint.

To specify a path constraint for some rigid body, $\mathcal{R}$, we require an ordered list of points $p_1, p_2, p_3, ..., p_n$, which represent the milestones along the path, a distance threshold δ, and a force coefficient $k \in [0, 1]$. In our system, it is assumed that the last point on the path is the goal location that we wish the object $\mathcal{R}$ to reach, and all other points are only landmarks guiding the object as to how this goal may be achieved. This assumption could be relaxed if we add weights to every path location indicating how important it is for $\mathcal{R}$ to reach that location. The distance threshold, δ, is used to determine if $\mathcal{R}$ is close enough to a milestone to consider that milestone reached. The default value for δ is the radius of the smallest sphere enclosing $\mathcal{R}$. As in the case of surface repulsion constraints, Sec. *4.3.2*, the default value of k is 1, which can be changed if some particular priority ranking between forces is desired. In our current system, the object's orientation along the path is left arbitrary, so that it can be determined by other soft constraints, such as surface repulsion. For a scene with an articulated robot, we can assign a part of the robot, such as the end effector, to follow the high-level path sequence.

5. CONSTRAINT IMPLEMENTATION

5.1 Solving The Constraints

We use two main constraint solving techniques in our current framework: penalty-based methods, used to represent soft constraints, and iterative relaxation, used to enforce hard constraints. The algorithm for updating the states of objects in our simulation is quite simple. We first apply the penalty forces due to the active soft constraints and update the simulation state. Next we apply an iterative relaxation technique which modifies the object states to ensure that all hard constraints are satisfied at the end of the time step.

5.1.1 Applying Penalty Forces

We require that 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 [26]. We use this method because some of the penalty forces are computationally intensive to evaluate, and the midpoint method provides stable integration with only two force evaluations per time step.

5.1.2 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. To efficiently ensure that these constraints are satisfied, we use the well known Nonlinear Gause-Siedel iterative relaxation method [24]. 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 system is in state S^t. For each type of hard constraint $\mathcal{C}_h$, we require an instance solver $Relax(\mathcal{C}_h, S^t)$, which returns a new state S in which $Res(C_h, S) = 0$. The specific methods used to relax each type of hard constraint, will be explained in Sec. 5.2.

The iterative relaxation method, described in Algorithm. 5.1, 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. Particular care must be taken to ensure that point distance constraints described in Sec. *4.2.2* and point planar angle constraints described in Sec. *4.2.3* are satisfied first, because only when these constraints are satisfied can we read back the resulting transformations on the rigid objects to be used in the remainder of the constraint solving iteration. We call the procedure which updates the states of the rigid objects in the system from the locations of the points that make up their associated rigid structures $Update\_Object\_States\_From\_Points(S)$ in Algorithm. 5.1.

5.1.3 Convergence of the Relaxation Method

The reason that we use the iterative relaxation method, Sec. *5.1.2*, instead of other techniques (e.g. Lagrangian formalism) for satisfying all the hard constraints, is because of its simplicity and because it allows our implementation to achieve interactive performance in most practical scenarios, even with a large number of active constraints. 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 when the iterative method begins the objects are not far from a configuration that satisfies the hard constraints. 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 occur in practice because the hard constraints are typically defined so that they are compatible with each other. Furthermore, in practical situations the hard constraints are satisfied before the planning simulation starts, allowing the system to benefit from temporal coherence from the beginning.

One of the goals of our planning framework was to allow the user to trade performance for accuracy to enable both very rapid prototyping and exact path computation. We can achieve this goal by allowing an upper bound to be imposed on the number of iterations used in relaxation, to guarantee that the simulation runs at the rate desired. If the system fails to converge within the specified number of iterations, the simulation continues as if it had converged. The result is error that manifests as small violations in the hard constraints. Despite of this problem, we have found that in practice the computed solution is a good approximation of the error free path obtained when no limit is placed on the number of constraint solving iterations. This can be considered as a useful feature for

rapid prototyping of CAD designs, when a quick estimation of the planned motion is required to provide real-time user feedback in the design process.

Relax_Constraints

Input The state S^t, at time t, of all rigid objects, points and vectors in the simulation, the set $\mathcal{C}_p$ of point hard constraints, and the set $\mathcal{C}_r$ of rigid object hard constraints.

Output New state vector S which satisfies all hard constraints.

Let $S \leftarrow S^t$.
While $\sum_{i=0}^{|C_p|} |Res(C_i, S)| + \sum_{j=0}^{|C_r|} |Res(C_j, S)| > 0$:
{
  While $\sum_{i=0}^{|C_p|} |Res(C_i, S)| > 0$:
  {
    for each point hard constraint $\mathcal{C}_p$:
      $S \leftarrow Relax(C_p, S)$.
  }

  $S \leftarrow Update\_Object\_States\_From\_Points(S)$

  for each hard rigid object constraint $\mathcal{C}_r$:
    $S \leftarrow Relax(C_r, S)$.
}
return The state S.

ALGORITHM 5.1: Relax Hard Constraints

5.2 Solving Hard Constraints

To satisfy each of the hard constraint types of Sec. 4.2: point distance, point planar angle and non-penetration, we have a corresponding $Relax$ solver that is used in the iterative algorithm of Sec. *5.1.2*. For the constraints that act on rigid objects the solver updates the state, position and orientation, of the objects, while for point constraints the solver modifies the positions of the points.

5.2.1 Point Distance Constraints

Given the formulation of a point distance constraint (Sec. *4.2.2*) between two points, p_1 and p_2, with corresponding world transformation, T_1^t and T_2^t, and a distance threshold d, we define the residual of the distance constraint:

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

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.

5.2.2 Point Planar Angle Constraints

Given the formulation of a point planar angle constraint between points p_1 and p_2, with angle limits θ_{min} and θ_{max}, we compute the angle θ as defined Sec. *4.2.3*. The residual of the angle constraint is then defined as:

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

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, $T_a^t(\overrightarrow{axis})$.

5.2.3 Non-Penetration Constraints

In our current implementation, these are the only constraints for which the $Relax$ method directly modifies the transformations of rigid objects in the scene. We use an in-house proximity (collision) query package, PQP [18, 12], 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 [26]. The approach that we use to implement the $Relax$ solver for non-penetration constraints is the impulse-based rigid body dynamic simulation [26]. 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.

5.3 Solving Soft Constraints

For each soft constraint we require a method, $Get\_Penalty\_Force$, which produces a force along the gradient vector of the constraint.

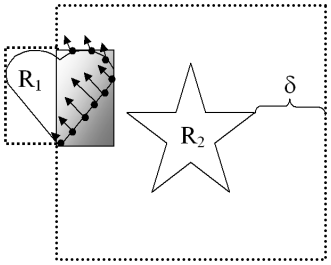

Figure 4: A portion of the distance field of R_2 which generates forces that act on R_1 pushing it away from R_2.

5.3.1 Surface Repulsion

To apply a surface repulsion constraint between object $\mathcal{R}_1$ and a second object $\mathcal{R}_2$, as described in Sec. *4.3.2*, 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 (Fig. 4). 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 with no penalty force applied. If the test fails then we compute the intersection, call it I, of the two bounding boxes. We then use a hardware accelerated distance field computation [14], to generate the distance field for the surface features of object $\mathcal{R}_2$ in the region I. 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. As a preprocess, a sampling of the surface of object $\mathcal{R}_1$, at some user specified resolution, is computed. The default resolution is the pixel resolution. For each sample point on $\mathcal{R}_1$ 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. For each sample point we generate a force in the direction of the gradient of the distance field, proportional to the distance between that sample point and $\mathcal{R}_2$, as seen in Fig. 4. 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 system the force for each sample point, p_i, is:

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

And, 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.

5.3.2 Goal Attraction

As described in Sec. *4.3.1*, if the robot has not reached its goal configuration, this constraint generates a penalty force that attracts the robot towards its goal. The strength of this force is directly proportional to the distance between the robot and the goal, and its direction is simply the direction between the center of mass of the robot and the location of the goal.

5.3.3 Path Following

Currently we implement a path following constraint, Sec. *4.3.3*, by first finding the closest point on the path, p_i, to the center of mass of the robot $\mathcal{R}$. If this point is not within the path distance threshold, δ, from the center of mass, then we consider that $\mathcal{R}$ has not reached p_i and we apply a force at the center of mass of $\mathcal{R}$ to push it towards p_i . If $\mathcal{R}$ is within the distance threshold of p_i, we apply a force at the center of mass of R in the direction of $p_{i+1} - p_i$ to push $\mathcal{R}$ along the path.

6. APPLICATIONS TO PROTOTYING

6.1 Implementation

Our system was implemented in an object-oriented framework using C++. We use the Proximity Query Package [12, 18] for collision detection, to enforce non-penetration constraints, and our in-house library HAVOC3D [14] to generate distance fields for surface repulsion constraints.

6.2 System Demonstration

We have tested our motion planning system in the following virtual prototyping 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 [5]. In our example, shown in Fig. 5(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. 5(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. 5(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 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 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.

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, very efficient, constraint solvers based on the *extension* of [2, 3, 17], as we extend this work to plan the motion of flexible bodies and also include different types of constraints in our system.

7. CONCLUSION AND FUTURE WORK

We have presented a novel framework for motion planning in virtual prototyping applications. We reformulate the motion planning problem into a physical simulation where constraints on the robot's motion guide it from its starting configuration to its goal. These constraints can enforce non-penetration constraints among objects, the angle limits and connectivity of articulated robot joints, the avoidance of collision, the following of estimated paths, and many other possible relationships between the robots and objects in the scene. The flexibility of our framework offers the possibility of natural extension in the following areas:

- **Inclusion of Additional Constraints:** such as non-holonomic constraints on object motion, as well as constraints that enforce more complex interactions (e.g. maintaining line of sight) between collaborating robots.
- **Extension to Flexible Geometry:** since our planning framework assumes no fixed or rigid geometry throughout the planning simulation.
- **Incorporating Direct Human Interaction:** to allow the user to directly control the motion of parts of the robot or obsta-

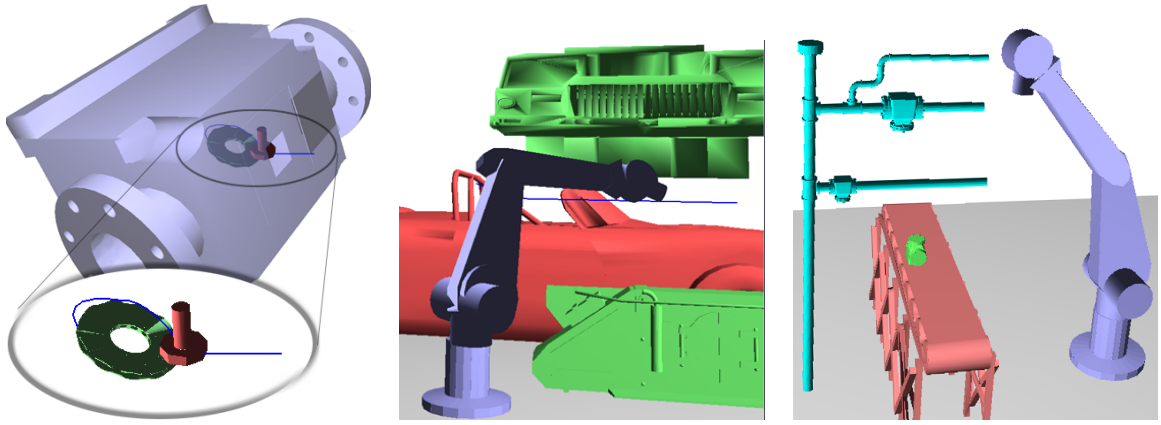

Figure 5: From left to right, *Maintainability Study Scene:* the planner must extract the bolt from the pump assembly. Both the bolt and the washer must be moved simultaneously around each other to avoid collision; *Automated Car Painting Scene:* the robot arm follows a path over the car body while avoiding obstacles; *Assembly Line Planning Scene:* the robot arm avoids the moving pipes to reach a moving part passing on the conveyer belt.

cles in the scene, thus better enabling interactive prototyping of CAD designs and faster verification of the design.

Acknowledgements

This research is supported in part by NSF DMI-9900157, NSF IIS-9821067, ONR N00014-01-1-0067 and Intel.

8. REFERENCES

[1] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robotic Research*, 1991.

[2] 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.

[3] B. Bruderlin and D. Roller (eds). *Geometric Constraint Solving and Applications*. Spring Verlag, 1998.

[4] J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.

[5] H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proceedings of International Conference on Robotics and Automation*, 1995.

[6] H. Choset and J. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. *IEEE Conference on Robotics and Automation*, 1995.

[7] H. Choset and J. Burdick. Sensor based planning: The hierarchical generalized voronoi graph. *Workshop on Algorithmic Foundations of Robotics*, 1996.

[8] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 7(2):228–240, 1991.

[9] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Tr. on Robotics and Automation*, 4:369–379, 1988.

[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] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[12] 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.

[13] D. Halperin, J. Latombe, and R. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 1999.

[14] 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.

[15] 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.

[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-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[21] T. Lozano-Perez and R. Wilson. Assembly sequencing for arbitrary motions. *Proc. IEEE International Conference on Robotics and Automation*, 1993.

[22] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[23] K. Ahrentsen N. Jacobsen, R. Larsen, and L. Overgaard. Automatic robotwelding in complex shipstructures. *J. Applied Artificial Intelligence*, 1997.

[24] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[25] L Overgaard, H. Petersen, and J. Perram. A general algorithm for dynamic control of multilink robots. *Int. J. Robotics Research*, 14(3), 1995.

[26] A. Witkin and D. Baraff. *Physically Based Modeling: Principles and Practice*. ACM Press, 1997. Course Notes of ACM SIGGRAPH.