Efficient Path Planning for Highly Articulated Robots using Adaptive Forward Dynamics

Russell Gayle[1], Stephane Redon[2], Avneesh Sud[1], Ming C. Lin[1], and Dinesh Manocha[1]

[1] University of North Carolina at Chapel Hill, U.S.A.
[2] INRIA Rhone-Alpes, France

We present an efficient path planning algorithm for highly articulated robots with many degrees of freedom (DoFs). Our approach takes into account physical and geometric constraints and formulates the problem as constrained dynamic simulation. We simulate the motion of the robot using a sub-linear time adaptive forward dynamics *algorithm. We also present fast collision detection and response computation algorithms for articulated models. Our planner computes an initial path for the articulated robot and refines the path by performing bounded-error dynamic simulation to ensure non-penetration. In practice, our planner scales well with the number of DoFs for highly articulated robots. We demonstrate its performance on models with hundreds and thousands of DoFs.*

1 Introduction

Highly articulated robots, such as snake or serpentine robots, with many degrees of freedom (DoFs) have received considerable attention recently [CB94b, HHC98, WJC+03]. Chirikjian and Burdick first introduced the term *hyper-redundant robots* to describe such robots with a very high number DoFs [CB90, CB94a]. Snake-like robots can serve as suitable alternatives over traditional robotic systems for difficult terrains and challenging scenarios. These include search and rescue missions in complex urban environments, planetary surface exploration, minimally invasive surgery, or inspection of piping and cabling. Highly articulated robots also have many applications in homeland security and national defense, as well as enabling inspection of ships, containers and other structures with narrow, tight workspace. Many computational biology algorithms also model the molecular chains as articulated models with hundreds or thousands of links.

Most of the prior work in motion planing for articulated models takes into account only the geometric constraints, like non-penetration, collision-free path computation and kinematics of manipulators. It is also important to account for the physical and dynamic constraints, and deal with the complexity of the hyper-redundant robots in terms of high number of DoFs. Current physically-based motion planning algorithms are limited to relatively simpler robots that can be represented as mass-spring systems and do not account

for all the physical constraints. Additionally, snake-like robots are often represented as 1D curves.

Main Results: In this paper, we present a novel path planning algorithm for highly articulated robots. We use the constraint-based planning framework [GL02] that treats both geometric and physical constraints and transforms the planning problem into a constrained dynamics simulation problem. In order to efficiently handle a large number of joints and modular components, we adapt a progressively refined forward dynamics algorithm [RGL05] and select the most important joints to simulate with bounded errors based on rigorous motion metrics. We also present algorithms for efficient collision detection between the articulated robot and the rest of the environment and collision response computation. Some of the main characteristics of our planning algorithm are:

- **Physically-based:** We take into account forward dynamics of articulated joints during motion planning, in addition to the geometric constraints (e.g. collision detection, contact handling, kinematic constraints, etc).
- **Efficiency:** We perform *lazy* dynamics update and achieve *sub-linear* running time performance in terms of DoFs when some of the joints do not move much.
- **Real-time:** Our algorithm *can* simulate the forward dynamics and plan the path for a robot with very high DoFs in real time, using a progressive refinement framework.
- **Error-bounded simulation:** Our tight approximation to a robot's dynamics and motion is based on well-defined motion metrics that compute the regions of simplification with bounded errors.

We have implemented our algorithm and tested the resulting system on Dell M60 Mobile Workstation, with a 2.1GHz Pentium-M processor and 1GB of main memory. On moderately complex planning scenarios, our algorithm is able to compute a path for an articulated robot consisting of up to $2,000$ DoFs at interactive rates. Our improved adaptive forward dynamics algorithm with contact resolution provides up to *10*x performance improvement in the resulting planner.

Organization: The rest of the paper is organized as follows. Section 2 summarizes related work in this area. We give an overview of our approach in Section 3. We present the forward dynamics algorithm and our extension to resolve contacts in Section 4 and 5, respectively. We describe our overall planning algorithm in Section 6 and discuss its implementation in Section 7.

2 Previous Work

In this section, we briefly review some of the related work in motion planning of the snake-like and flexible robots. We also give an overview on forward dynamics algorithms for articulated robots.

2.1 Motion Planning of Snake-like and Flexible Robots

Many of the existing work in motion planning can be applied to planning of general articulated robots [Lat99, CLH+05]. Most of the prior work in motion

planing for snake-like robots takes into account only geometric constraints and kinematics of manipulators [CB90, CB94a, CB94b, HHC98]. Most of the existing algorithms for deformable robots are specialized for snake-like geometry, such as pipes, cables, ropes, and flexible wires [SSL96, NK97, AOLK00, LK04, MK04, MK05, GSLM05]. However, most of these techniques either are geometric in nature or use optimization techniques that do not consider some of the physical constraints of highly articulated robots, such as forward dynamics for a high degree-of-freedom chain, joint limits, collision detection and contact resolution. A recent planning algorithm deals with more general deformable robot [RLA06], but does not consider friction and motion constraints (e.g. joint limits), which could be important for realistic modeling of interaction between the robot and its environment.

Fast specialized algorithms for tying knot [BLM04] and thin solids [Pai02] have also been proposed. But, they are designed for small-scale interaction and relatively focused domains. It is not clear how well they scale up for complex, lengthy snake-like robots.

2.2 Forward Dynamics

Multi-body systems and forward dynamics is central to simulation of an articulated robot. Due to the space limitation, we refer the readers to a recent survey [FO00] for more details. Our earlier work on adaptive forward dynamics [RGL05] also gives a brief overview of recent work in this area.

3 Overview

This section will provide the notation used throughout the paper and formalize the planning problem as constrained dynamic simulation of an articulated robot.

3.1 Notation

We simulate our articulated robot, R, as a sequence of m rigid bodies connected by $m-1$ joints, $j_1, \ldots, j_{m-1}$. The state or configuration C of the robot at time t is the collection of $m-1$ joint angles $Q(t) = \{q_1(t), \ldots, q_{m-1}(t)\}$ and $m-1$ joint velocities $\dot{Q}(t) = \{\dot{q}_1(t), \ldots, \dot{q}_{m-1}(t)\}$, or $C(t) = \{Q(t), \dot{Q}(t)\}$. We also keep track of the joint accelerations, $\ddot{Q}(t) = \{\ddot{q}_1(t), \ldots, \ddot{q}_{m-1}(t)\}$. Each joint can theoretically have up to six degrees of freedom (DoFs) (three revolute directions, three prismatic), but for articulated robots it is sufficient for joints to be 1-DoF. With this type of joint, we can construct an articulated robot that effectively has 2-DoF joints by attaching two joints to each other, one rotated to have an axis of rotation orthogonal to both the other joint and the axis of the robot. This construction simplifies the implementation of the articulated body. Also associated with each joint j_i is a pair of joint limits, $(lowerJointLimit, upperJointLimit)$, which defines the range of motion for a particular joint. Additionally, we refer to first link in a serial linkage, or the root node in a tree linkage as a the base link.

The motion equations for the articulated body are given using the *spatial notation* introduced by Featherstone [Fea99]. Essentially, spatial notation includes both translational and rotational components of motions and forces

in six-dimensional vectors. For example, the velocity field of a rigid body in a reference frame centered in O is represented by the six-dimensional vector $\hat{\mathbf{v}} = [\omega\ \mathbf{v}(O)]^T$, where ω is the angular velocity of the rigid body and $\mathbf{v}(O)$ is the linear velocity of the rigid body *at the origin of* O. Further information about spatial algebra, spatial kinematics, and spatial dynamics can be found in [Fea99, Kok04, Mir96].

The planning environment consists of a set of rigid, static obstacles $O = \{o_1, o_2, \ldots\}$ in the workspace W. With these, we can formulate our problem as follows:

Find a sequential set of robot configurations $\{C(t_o) \ldots C(t_f)\}$ *such that no* $C(t_i)$ *intersects any obstacle in* O*, and* $C(t_i)$ *satisfies environment boundary, non-penetration, joint-limit, and dynamics constraints, where* $C(t_o)$ *is the initial configuration of the robot and* $C(t_f)$ *is the final configuration.*

3.2 Constraint-Based Motion Planning

The core of our planning algorithm builds upon Constraint-Based Motion Planning (CBMP) [GL02]. CBMP effectively maps the motion planning problem into a constrained dynamics simulation. The planner links the initial and final configurations by sequentially computing intermediate dynamic states that satisfy the given set of constraints.

One key advantage of CMBP is that physical and mechanical properties of the robots and obstacles are automatically available along with geometric constraints. Furthermore, while CBMP can operate in high-dimensional configuration spaces, it can plan the path directly in the workspace by posing the planning problem as simulation of a constrained dynamical system. This transformation of the problem makes an intractably large configuration space much more manageable.

The core of the CBMP framework is a physically-based simulation in which constraints are satisfied by applying virtual forces to the system. The constrained dynamical system simulated is then used to guide the robot to a goal configuration, while ensuring that constraints are satisfied.

3.3 Planning for a hyper-redundant robot

Although multi-body dynamics has been widely studied, it is still computationally costly to simulate a system with a high number of DoFs. To efficiently simulate an articulated robot R, we take advantages of a recently introduced technique called **adaptive forward dynamics** for articulated bodies [RGL05]. This method enhances the performance of our planner by determining the joints undergoing most significant motion and thereby perform updated computations accordingly. The algorithm simulates either a fixed number of "active joints" or the number needed to reach the desired error tolerance. Based on motion metrics, it automatically updates the set of active joints that best represent the overall motion of the robots. This simulation handles the kinematic joint and articulated body dynamics constraints required for the robot.

However, the original algorithm does not handle contacts. We extend it to provide efficient collision detection and (possibly) sub-linear time contact response methods to satisfy the boundary, non-intersection, and non-penetration constraints. Additionally, our formulation can handle the collision and contact dynamics constraints effectively. The remaining constraints

enforce joint limits on the robot and encourage it to follow a guide trajectory that can be pre-computed for any given environment using one of the previously known methods [GL02, GSLM05].

Finally, we integrate the articulated body dynamics with contact handling with a simple planner, which is responsible for creating a guiding trajectory for the robot and locally altering the planned trajectory to avoid obstacles by enforcing the constraints.

4 Adaptive Articulated-Body Dynamics

We extend the *adaptive articulated-body forward dynamics algorithm* [RGL05] which adaptively recomputes the forward dynamics of a snake robot by only simulating the joints that best depict the overall motion of the robot with bounded errors. Precisely, we are able to select the *appropriate* number of simulated joints in the articulated body when determining a path for the robot. The adaptive dynamics algorithm only simulates these many joints, which results in faster dynamics computations and automatically evolves the set of simulated joints at runtime based on the forces applied to the articulated body. The set of active joints are selected based on *customizable motion metrics*, using an error-bounded approximation of the articulated body acceleration.

In the following, we briefly describe the basic components of the adaptive articulated-body dynamics. For more details, we refer the reader to the original paper by Redon *et al.* [RGL05].

4.1 Articulated-body dynamics

The adaptive dynamics algorithm is built upon Featherstone's *divide-and-conquer algorithm* (DCA) [Fea99]. Featherstone's algorithm is a linear time algorithm to compute the forward dynamics of an articulated body based on the forces applied to it. The algorithm relies on the following *articulated-body equation:*

$$\begin{bmatrix} \hat{\mathbf{a}}_1 \\ \hat{\mathbf{a}}_2 \\ \vdots \\ \hat{\mathbf{a}}_m \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}_1 & \mathbf{\Phi}_{12} & \cdots & \mathbf{\Phi}_{1m} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_2 & \cdots & \mathbf{\Phi}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Phi}_{m1} & \mathbf{\Phi}_{m2} & \cdots & \mathbf{\Phi}_m \end{bmatrix} \begin{bmatrix} \hat{\mathbf{f}}_1 \\ \hat{\mathbf{f}}_2 \\ \vdots \\ \hat{\mathbf{f}}_m \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{b}}_1 \\ \hat{\mathbf{b}}_2 \\ \vdots \\ \hat{\mathbf{b}}_m \end{bmatrix}, \tag{1}$$

where $\hat{\mathbf{a}}_i$ is the 6×1 spatial acceleration of link i, $\hat{\mathbf{f}}_i$ is the 6×1 spatial force applied to link i, $\hat{\mathbf{b}}_i$ is the 6×1 bias acceleration of link i (the acceleration link i would have if all link forces were zero), $\mathbf{\Phi}_i$ is the 6×6 inverse articulated-body inertia of link i, and $\mathbf{\Phi}_{ij}$ is the 6×6 cross-coupling inverse inertia between links i and j.

The DCA employs a recursive definition of an articulated body: an articulated body is a pair of articulated bodies connected by a joint. The sequence of assembly operations is described in an *assembly tree*: each leaf node of the assembly tree represents a rigid body, while each internal node describes an assembly operation, *i.e.* a subassembly of the articulated body. The root node of the assembly tree represents the complete articulated body.

The forward dynamics of the articulated body are computed in essentially two steps. First, the *main pass* recursively computes the inverse inertias, the inverse cross-coupling inertias, and the bias accelerations of each node in the assembly tree, from the bottom up. Then the *back-substitution pass* computes the acceleration and the kinematic constraint forces relative to the principal joint of each internal node in the assembly tree, in a top-down way. When the DCA completes, all joint accelerations and all kinematic constraint forces are known.

4.2 Adaptive articulated-body dynamics

Featherstone's DCA is linear in the number of joints in the articulated body: all nodes have to be processed in each pass of the algorithm, and each joint acceleration has to be computed. Determining a path or resolving contacts for a highly articulated body could be prohibitively slow using a typical forward dynamics algorithm. In order to improve the performance of the planner, we incorporate the adaptive dynamics algorithm by Redon *et al.* [RGL05] to lazily simulate the articulated body motion that best represents the overall motion of the robot with an error-bounded approximation. Essentially, this enhanced algorithm allows us to systematically choose the appropriate number of joints that are simulated in the articulated body, by automatically determining *which* joints should be simulated, in order to provide a high-quality approximation of the articulated-body motion.

Essentially, the adaptive algorithm relies on the proof that it is possible to compute an *acceleration metric value* $\mathcal{A}(C) = \sum_{i \in C} \ddot{\mathbf{q}}_i^T \mathbf{A}_i \ddot{\mathbf{q}}_i$, *i.e.* a weighted sum of the joint accelerations in the articulated body, *before computing the joint accelerations themselves*. Specifically, they show that the acceleration metric value $\mathcal{A}(C)$ of an articulated body can be computed from the forces applied to it:

$$\mathcal{A}(C) = \begin{bmatrix} \hat{\mathbf{f}}_1 \\ \hat{\mathbf{f}}_2 \\ \vdots \\ \hat{\mathbf{f}}_m \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Psi}_1 & \boldsymbol{\Psi}_{12} & \cdots & \boldsymbol{\Psi}_{1m} \\ \boldsymbol{\Psi}_{21} & \boldsymbol{\Psi}_2 & \cdots & \boldsymbol{\Psi}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Psi}_{m1} & \boldsymbol{\Psi}_{m2} & \cdots & \boldsymbol{\Psi}_m \end{bmatrix} \begin{bmatrix} \hat{\mathbf{f}}_1 \\ \hat{\mathbf{f}}_2 \\ \vdots \\ \hat{\mathbf{f}}_m \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{f}}_1 \\ \hat{\mathbf{f}}_2 \\ \vdots \\ \hat{\mathbf{f}}_m \end{bmatrix}^T \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_m \end{bmatrix} + \eta, \quad (2)$$

where $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Psi}_{ij}$ are 6×6 matrices, $\mathbf{p}_i$ is a 6×1 vector, and η is in $\mathbb{R}$. The coefficients $\boldsymbol{\Psi}_i$, $\boldsymbol{\Psi}_{ij}$, $\mathbf{p}_i$ and η are called the *acceleration metric coefficients* of the articulated body.

The acceleration metric is used to predict which nodes have the largest overall acceleration during the top-down back substitution pass, and to restrict the computation of the joint accelerations to a *sub-tree* of the assembly tree only, while implicitly assuming that the other joint accelerations are zero. In effect, this allows us to determine an *error-bounded approximation* of the articulated-body acceleration, and evolve the set of active joints accordingly.

In summary, the adaptive algorithm is able to automatically determine which joints move the most, according to the acceleration and a similar velocity motion metric[1], based on the forces applied to the articulated body.

[1] In our implementation, we use identity weight matrices.

5 Contact, Collision, and Joint Impacts

This section describes a number of constraints for the articulated body. We first describe how contact, collision, or joint impact situations are detected. Then, we describe how to add non-penetration, non-intersection, and boundary constraints, as well as dynamics-based contact response.

5.1 Impact Determination

Joint-Impact Determination

A joint impact occurs when a joint position reaches its limits, to ensure that the robot does not bend too much in any particular direction. In general, it is only necessary to compare the current position of each joint with its predefined limits. In order to maintain efficiency with adaptive articulated-body dynamics, this generalizes to only checking joints which are currently active. Thus, joint impacts can be found by traversing the assembly tree and terminating at rigidified nodes. All joints which violate the constraint are reported to the resolution system.

Collision and Contact Determination

Contacts and collisions are impacts that can occur between the body and the environment, due to an intersection or penetration between the robot and the environment. Contacts are distinguishable from collision by comparing the relative velocity of the bodies at the contact location. This information is computed during the contact and collision resolution phase. Thus, the remainder of this section focuses on finding the impact events. Since the geometric aspects are identical, collision detection and contact determination are treated as the same operation with the two terms used interchangeably.

Collision detection has been widely studied for decades. Although collision detection for snake robots can be handled using a naive application of existing methods, inefficient collision checks can result in sub-optimal performance for the overall planning algorithm, as contact queries can take up to 95% or more of the overall running time for the planner.

We perform two culling steps, based on *axis-aligned bounding boxes* (AABBs) and *oriented bounding boxes* (OBBs), to help localize potential collisions, before performing intersection tests at the triangle level. We pre-compute and store one hierarchy of oriented bounding boxes for each rigid robot link and each rigid environment obstacle. We also precompute one axis-aligned bounding box for each obstacle in the environment. The OBB hierarchies and AABBs of environment obstacles do not have to be updated during planning.

At runtime, we determine the intersections between the robots and the environment obstacles using the following collision detection algorithm:

- **AABB hierarchies update:** For each mobile rigid link, we determine a bounding AABB using the root OBB of the OBBTrees [GLM96]. We then compute an AABB for each node of the assembly tree (*i.e.* for each sub-assembly of the articulated body) using a bottom-up pass. We thus obtain one AABB hierarchy per articulated body, whose structure is *identical to the assembly tree of the articulated body.*

- **AABB culling:** Using the AABB hierarchies, we detect potentially colliding rigid links and objects.
- **OBB culling:** When two rigid objects are found to potentially intersect after the AABB culling step, we recursively and simultaneously traverse their OBB hierarchies to help localize potential collisions between pairs of triangles [GLM96].
- **Triangle/triangle intersection tests:** Whenever two leaf-OBBs are found to overlap, a triangle/triangle intersection test is performed to determine whether the triangles contained in the leaf-OBBs intersect. When they do, we report the the corresponding intersection segment.

All pairs of intersecting triangles and the corresponding intersection segments are reported. Note that this algorithm handles both robot/robot collision detection and robot self-collision detection, since all steps can operate on pairs of links that belong to distinct or identical robots.

5.2 Collision, Contact, and Joint-Impact Resolution

This section describes the methodology used in responding to collisions, contacts, and impacts events. In general, we formulate the problem as determining what virtual force or virtual impulse must be applied to the body in order to achieve a desired response and how to apply the force or impulse to ensure the robot is in a valid state. Additionally, these methods must be efficient; otherwise we would not be able to achieve the sub-linear runtime complexity as the original adaptive forward dynamics.

We first describe a unified impulse-based approach suitable for handling both collision and contact. This formulation inherently handles dynamics constraints such as static and sliding friction. Then, we describe a method for resolving joint impacts.

Impulse-Based Dynamics

Impulse-based dynamics provides a unified framework for handling both contacts and collisions. In particular, it integrates physical concepts for both collision and friction into a single method. It is derived from the following assumptions: all colliding objects are perfectly rigid, Stronge's hypothesis, and the Coulomb friction law. In practice, impulse-based dynamics are a local solution as opposed to constraint-based dynamics, which provide a global solution.

For both rigid and articulated bodies, impulse-based dynamics consists primarily of three steps.

1. **Collision Matrix Determination:** The collision matrix captures the dynamics properties of either a rigid or an articulated body for a given contact or collision.
2. **Collision Integration:** Given a collision matrix and some other dynamics information, this step outputs an appropriate impulse that satisfies collision and contact constraints.
3. **Impulse Application:** This step applies the impulse to the physical model. While this is simple for rigid bodies, the process of applying impulses to articulated bodies is more involved.

From the current impulse-based dynamics for articulated bodies, the first step has a worst case $O(m)$ run-time complexity for m joints, and the third step has a linear, $O(m)$ runtime complexity, since it must propagate the velocity resulting from the impulse to the remaining joints on the body. The second step is performed independently from the body, and is bounded by the time to perform numerical integration over a pair of intervals representing the compression and restitution phases.

Since efficiency is a goal of our planner, we use a modified impulse-based dynamics scheme which fits within the adaptive dynamics framework. Thus, impulses are determined and applied to the hybrid state of the articulated body. Next we briefly describe the necessary changes to the impulse-based dynamics algorithm. For more details, we refer the reader to [XXX06].

Collision Matrix Determination

For articulated bodies (and similarly for rigid bodies) the collision matrix, K, is a 3×3 matrix and is constant for each contact or collision. It can be shown that for a single contact, there is a linear relationship between the applied contact impulse and the resulting change in velocity, with K being the linear coefficient. Thus, K can be determined by applying unit test impulses in each of the basis directions and observing the result.

In terms of a serial linkage, this amounts to one update from the link in contact to the base link to propagate the impulse, and another update from the base to the link in contact to determine the change in its velocity. By taking advantage of the error metrics established for adaptive articulated body dynamics, we can limit this traversal to only the joints that must be simulated given an user-defined error tolerance. Since the robot body is organized into an assembly tree, this amounts to performing in-order traversals in either direction, and not proceeding past sub-trees that are marked as inactive.

This change ensures that each collision matrix determination asymptotically does as much work as the adaptive forward dynamics computation, since each one operates only over active joints. It should also be noted that a collision matrix must be determined for each contact point.

Impulse Response

By this stage, we have determined what impulse needs to be applied in order to properly simulate and satisfy the contact or collision dynamics. This task involves appropriately applying the impulse to the body and propagating the resulting change in velocity.

This requires one pass from the contact link to the base link, and then a complete pass from the base to all the links. Like collision matrix determination, this too ordinarily requires a linear run-time complexity in the number of DoFs. However, as in the finding the collision matrix, the pass up the tree can be easily reduced to only the active joints through the modified in-order traversal. And, the top-down pass similarly terminates at the inactive nodes.

5.3 Joint-Impact Resolution

Here we describe an efficient method for satisfying the joint limit constraint. While a simple option is through the application of stiff spring-like penalty

forces, such an approach can lead to instabilities in simulation. Instead, we modify the acceleration-based constraint [Kok04] in order to utilize our adaptive framework.

This work relies on the fact that at any particular instant in time, there exists a linear relationship between the magnitude of the generalized forces acting upon the body and the magnitude of the acceleration of either the body's joints or links. Here, a generalized force refers to the aggregation of all forces and torques being applied. This can be expressed as either

$$a^1 = kf + a^0 \quad \text{or} \quad \ddot{q}^1 = kf + \ddot{q}^0$$

where a refers to spatial acceleration at a point, f is the generalized force (a collection of external forces and torques) or a joint force, $\ddot{q}$ is the joint acceleration, k describes a scalar inverse inertia, and the super-scripts 1 and 0 refer to the acceleration after application of the force, and before it, respectively. It follows that for linear constraint functions on joint accelerations, $h(\ddot{q})$,

$$h(\ddot{q}^1) - h(\ddot{q}^0) = kf$$

for some constant k. Thus, we can determine k by evaluating the constraint function before and after application of a unit test joint force.

Given k, we can solve for f (as a joint force) in order to satisfy the constraint. This can be easily generalized to solve for multiple, simultaneous constraints as described in [Kok04]. Computationally, this step requires inverting a matrix whose dimensions are equal to the number of constraint functions. In practice, this number has been fairly low.

Application of the resulting joint force will take advantage of our adaptive framework. Therefore, little extra work is done by the forward dynamics algorithm. It remains to show how we apply this formulation to joint-impacts. [Kok04] gives a treatment of this type of constraint, requiring one acceleration and one impact constraint for each event. The acceleration constraint can easily be derived from the above discussion. When a joint reaches a limit, we want to ensure that it will no longer move in a direction that will violate the joint limit. For acceleration, setting the joint's acceleration to zero will ensure that the joint no longer moves against its limitations. We can use the simple constraint function,

$$h(\ddot{q}) = \ddot{q}$$

which is satisfied when $h(\ddot{q}) = 0$, or $\ddot{q} = 0$. So, for each impact, we add one of these constraints to the constraint set.

The impact constraint is formed in a similar manner. However, instead of affecting the acceleration, it requires a direct change in joint velocities, much like the prior impulse-based method. The key concept is that over a discrete time interval δt, an acceleration is seen as a change in velocities, with magnitude

$$\ddot{q}^I = \frac{(\dot{q}^1 - \dot{q}^0)}{\delta t}$$

where $\ddot{q}^I$ is the acceleration due to an impact. Without loss of generality, δt can be set to 1, and this can be placed in an acceleration constraint,

$$h(\ddot{q}) = \ddot{q} - \ddot{q}^I$$

At this time, when the constraint function evaluates to zero, or our constraint is satisfied, the joint acceleration will be $\ddot{q} = \ddot{q}^I$. Once we determine the force necessary to satisfy the constraint and determine accelerations after its application, the joint velocities can be computed and directly applied to the body. Since this is all based on acceleration updates, it fits nicely within the adaptive forward dynamics framework. Thus, the run-time complexity is no worse than the original adaptive forward dynamics algorithm.

6 Path Planning

In this section, we describe the planning algorithm that combines kinematic, non-penetration, joint-limit, and dynamics constraints in order to plan the motion for a highly articulated, snake-like robot. It should be noted that while the emphasis is on serial linkages, these ideas are applicable to tree linkages such as legged robots as well as more complex linkages with loops.

The complete planning algorithm proceeds in two phases; (1) an initial trajectory or guiding path computation phase followed by (2) the simulation phase.

6.1 Initial Trajectory Computation

The planner first computes an initial trajectory $p_o, p_1, p_2, \ldots, p_f$ for the robot R to take, where each p_i is a milestone along the path, and p_o and p_f represent the initial and final configuration. This trajectory does not need to be collision free along the entire path. By satisfying all geometric and physical constraints, the simulation phase will locally correct the motion and subsequent path to stay in a valid, collision free state.

Rather than simply generating random samples in the workspace, several other options are available. A simple, and effective, approach takes the cross-sectional diameter, d, of the robot into account. Following well-known roadmap generation methods, we compute random samples in the work space such that no sample is within a distance d from any obstacle in O. Then, nearby samples, or milestones, are connected to construct a roadmap. By connecting the initial and goal position of the root to the roadmap, an initial guiding path can be generated.

It should be noted that for this robot, the initial paths do not likely guarantee collision-free paths. These guiding paths only serve as an initial guess at how the robot should move through the environment. As mentioned earlier, the constrained dynamic simulation will locally adapt the robot to ensure a valid path is generated.

6.2 Simulation

The simulation phase is responsible for maintaining the state while allowing the robot to progress through the environment to the goal. The result of the simulation is a sequence of robot configurations that constitute the complete path of the robot. This phase proceeds either until the goal is reached, or some maximum time has lapsed for which to assume the robot cannot make it to the goal.

At the core of the simulation is the current state $C(t_i)$ of the robot along with a list of constraints and constraint solvers. At each time step of the

simulation, the constraints must be satisfied. These constraints include non-penetration constraint, collision avoidance with the obstacles, staying within the environment, and maintaining joint limits. The previous section described how to solve these constraints. Then, through numerical integration, the system is updated to give the next state, $C(t_{i+1})$.

But, up to this point, all the constraints mentioned serve to ensure the robot properly interacts with the environment. We also provide another constraint, in the form of a penalty force, which works to move the robot along the guiding path toward the goal. We use a local planner similar to that of Gayle et. al [GSLM05], but additionally include extra repulsive and attractive forces whose goal is to help the robot's body stay closer to the path while also moving along it.

Since the guiding path is essentially in the workspace, there is still the possibility the robot is placed into a configuration which will break a constraint. For instance, this could occur when the robot must turn around a sharp corner, that forces some joint to go beyond its joint limits.

To remedy this situation, once the robot reaches a milestone, we store its configuration. Then, if the robot reaches such a state, we remove the edge currently being traversed from the roadmap and recompute a trajectory. Once a new trajectory is found, we set the robot's state to the milestone farthest along the new trajectory and resume the simulation. This effectively enables the robot to backup while ensuring it can reach a valid state along the new path. This approach is essentially a depth-first search for valid paths.

6.3 Planning Step

Here we show a summary of steps in the simulation. The simulation phase proceeds as follows:

1. Perform collision detection and check joint limits
2. Update the constraint set to include collisions, solve, and apply the constraint forces f^c.
3. Propagate the forces, f^c in the body (i.e. update the bias and joint accelerations for each active region)
4. Integrate to find new joint positions ($q(t)$) and new velocities ($\dot{q}(t)$) for this step
5. a) Check to see if this is a valid state
   b) If not a valid state, recompute the path and set the positions and velocities to their value at the milestone that was most recently visited and on the new path.
6. Update current joint positions and velocities with the new joint positions and velocities.

7 Implementation and Results

We have implemented this algorithm on a Dell M60 Mobile Workstation, with a 2.1 GHz Pentium-M processor and 1 GB of main memory. To test the effectiveness and feasibility of the approach, we applied the planner to several different scenarios.

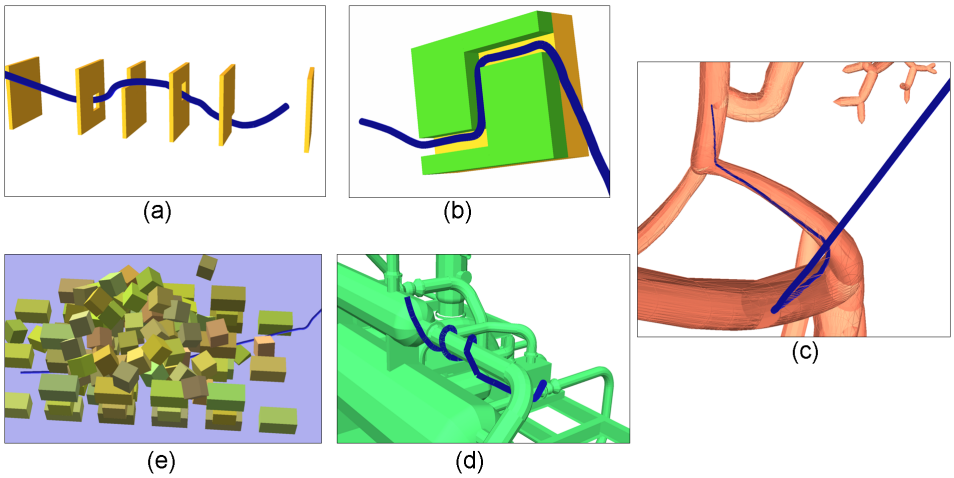

Fig. 1. Benchmark scenarios: (a) Serial Wall; (b) Tunnel; (c) Liver Catheterization; (d) Pipes; (e) Debris.

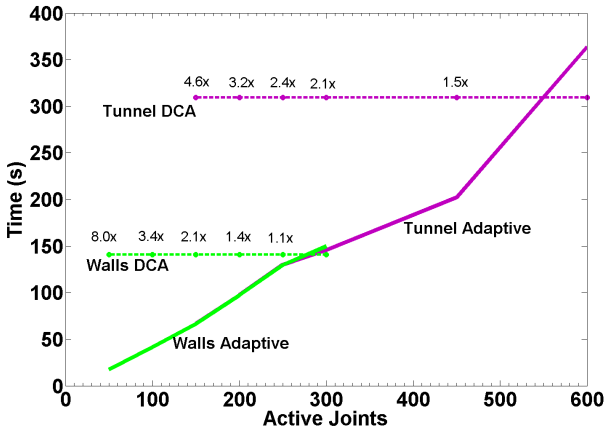

Fig. 2. Simulation time vs. number of active joints. This graph shows the time spent in dynamics computations for a set number of active joints. The dashed horizontal lines show represents the time if DCA was used for forward dynamics. The numbers above this line represent the speedup observed for that number of active joints.

- **Serial Walls** - This scenario is based upon a Texas A&M Parasol Motion Planning benchmark. The robot must travel through each of the holes in order to reach the goal. (See Fig. 1(a))
- **Tunnel** - Our Tunnel scenario is also based upon a Texas A&M Parasol motion planning benchmark. It is composed of a simple tunnel through a block with two right angle bends about half way through the block. This scene requires the planner to quickly solve many simultaneous contact constraints. (See Fig.1(b))

Scene	Total Joints	Active Joints	Env. (tri)	Robot (tri)	Sim. Time (s)	Avg. Step Time (s)
Walls	300	50	216	6000	17.5	0.0008
Tunnel	600	150	72	12000	66.92	0.003
Catheter	2500	200	80086	50000	1821	0.0071
Pipes	2000	200	38146	40000	193.6	0.0064
Debris	2000	175	1296	40000	157.3	0.0059

Fig. 3. This table shows the complexity of the robot, scene, and the time spent in the dynamics stages of planning.

- **Catheterization** - Based on a simulated medical procedure, a thin tube-like catheter must navigate through a sequence of arteries. After entering through the femoral arteries, the goal is to find a tumor in the liver for treatment. (See Fig. 1(c))
- **Pipes** - This situation represents an application of snake robots for pipe inspection. A snake-like robot coils around piping while searching for a leak in the pipes. (See Fig. 1(d))
- **Debris** - Modeling a search-and-rescue scenario, the snake-like robot searches for an opening in a pile of debris where a survivor or important artifact may be found. Then, the snake must find a way out of the debris. (See Fig. 1(e))

In all of these scenarios, the robot is modeled as a sequence of connecting cylinders. The geometric representation of each cylinder contains 20 triangles. As previously described, each joint is a 1-DoF revolute joint.

The results of each planning algorithm can be seen in their associated figure. The performance of our planner is highlighted in Figure 3. This table shows the complexity of the robot in terms of joint and geometric complexity. The active joint column refers to the number of joints simulated during the scenario. This number was chosen such that the robot could complete the task with bounded motion error and to be fairly efficient. Finally, the total time spent in simulation, as well as the average time step, is shown for each task.

The table highlights the performance of our planning algorithm. Compared to previous approaches for catheterization, our total planning time was about 7 to 8 times faster than reported results [GSLM05]. For the other scenarios, the reported time is usually fairly small, with dynamics computation taking less than 4 minutes in many of the scenarios. It should be noted that the overall dynamics time is highly dependent upon the length of the guiding path. This explains why the catheterization benchmark takes much longer while having a time step comparable to the other complex benchmarks.

The effectiveness of the adaptive framework is explored in Fig. 2. In the graph, the horizontal bars represent timings using the standard Featherstone DCA. Since DCA is not adaptive, it runs linearly with the number of joints. As can be seen, the adaptive framework enables much faster computation. In practice, we have observed up to one order of magnitude speedup with about 10% to 15% of the joints being active.

7.1 Analysis

We discuss some of the comparisons with other planners here, with regard to the more complex debris, catheterization, and pipes benchmarks. Note that

while many of the DoFs are fairly constrained, the configuration space (C-space) still has over 2000 DoFs.

A standard Probabilistic Roadmap (PRM) planner is not feasible at this point since it would require an extraordinarily large number of samples and does not easily model the dynamics of the scene. Similarly, the sequential framework [GG95] does not consider dynamics. While some variants of Rapidly-exploring Random Tree (RRT) planners [LK00] have handled contacts, their performance also would suffer from the size of the C-space.

Another recent related work plans for curves of constant length [MK05], but again this does not handle dynamics with the environment. Gayle et. al [GSLM05] plan for a similar catheter scenario, but the reported time is about 8 times slower than that of our planner for snake-like robots. The planner by Rodriguez et. al [RLA06] would have difficulty with these scenarios due to the high aspect ratio of the robot, as the robot plans around its center of mass.

Our approach also has some limitations. Namely, it currently cannot handle tree-linkages or loops. Its performance is dependent upon the error tolerance and the number of active joints. There is no guarantee that the motion is accurate in all cases.

8 Conclusion and Future Work

In this work we have introduced a novel planning framework for snake-like robots. It uses an adaptive approach to simulating multi-body dynamics by using error-bounded motion metrics. It incorporates collision and contact constraints to model dynamics with the environment. The preliminary results are promising, generating paths with bounded errors in motion metrics very efficiently.

There are many potential directions for future research based on this planning algorithm. As mentioned in the limitations, we would like to generalize the method to handle more complex articulated topologies. Since the focus is on snake-like robots, it would be useful to update the model to include snake-like motion. We would also like to validate our results where possible, such as on clinical trials for the catheterization procedure.

References

[AOLK00] E. Anshelevich, S. Owens, F. Lamiraux, and L. Kavraki. Deformable volumes in path planning applications. *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2290–2295, 2000.

[BLM04] J. Brown, J.-C. Latombe, and K. Montgomery. Real-time knot-typing simulation. *Visual Computer*, 20, 2004.

[CB90] G. S. Chirikjian and J. W. Burdick. An obstacle avoidance algorithm for hyper-redundant manipulators. *IEEE Transactions of Robotics and Automations*, 1990.

[CB94a] G. S. Chirikjian and J. W. Burdick. A modal approach to hyper-redundant manipulator kinematics. *IEEE Transactions on Robotics and Automation*, pages 343–354, 1994.

[CB94b] H. Choset and J. Burdick. Extensibility in local sensor based planning for hyper redundant manipulators (robot snakes). *AIAA/NASA CIRFFSS*, 1994.

[CLH+05] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[Fea99] Roy Featherstone. A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 1: Basic algorithm. *International Journal of Robotics Research 18(9):867-875*, 1999.

[FO00] R. Featherstone and D. E. Orin. Robot dynamics: Equations and algorithms. *IEEE Int. Conf. Robotics and Automation, pp. 826-834*, 2000.

[GG95] K. Gupta and Z. Guo. Motion planning for many degrees of freedom: Sequential search with backtracking. *IEEE Transactions on Robotics and Automation*, 11(6):897–906, 1995.

[GL02] M. Garber and M. Lin. Constraint-based motion planning for virtual prototyping. *UNC Technical Report*, 2002.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[GSLM05] R. Gayle, P. Segars, M. Lin, and D. Manocha. Path planning for deformable robots in complex environments. *Proc. of Robotics: Science and Systems*, 2005.

[HHC98] W. Henning, F. Hickman, and H. Choset. Motion planning for serpentine robots. *Proc. of ASCE Space and Robotics*, 1998.

[Kok04] V. Kokkevis. Practical physics for articulated characters. *Proc. of GDC*, 2004.

[Lat99] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, pages 1119–1128, 1999.

[LK00] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Robotics: The Algorithmic Perspective (Proc. of the 4th Int'l Workshop on the Algorithmic Foundations of Robotics*, 2000.

[LK04] A. Ladd and L. Kavraki. Using motion planning for knot untangling. *International Journal of Robotics Research*, 23(7-8):797–808, 2004.

[Mir96] B. V. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California, Berkeley, 1996.

[MK04] M. Moll and L. Kavraki. Path planning for minimal energy curves of constant length. *Proceedings of International Conference on Robotics and Automation*, 2004.

[MK05] M. Moll and L. Kavraki. Path planning for variable resolution minimal-energy curves of constant length. *Proceedings of International Conference on Robotics and Automation*, pages 2143–2147, 2005.

[NK97] H. Nakagaki and K. Kitagaki. Study of deformation tasks of a flexible wire. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1997.

[Pai02] D. Pai. Strands: Interactive simulation of thin solids using cosserat models. *Computer Graphics Forum (Proc. of Eurographics)*, 2002.

[RGL05] S. Redon, N. Galoppo, and M. Lin. Adaptive dynamics of articulated bodies. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 24(3), 2005.

[RLA06] S. Rodriguez, J. Lien, and N. Amato. Planning motion in completely deformable environments. *Proc. of IEEE Int. Conf. Robot. Autom. (ICRA)*, 2006.

[SSL96] D. Sun, X. Shi, and Y. Liu. Modeling and cooperation of two-arm robotic system manipulating a deformable object. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2346–2351, 1996.

[WJC+03] A. Wolf, H. Brown Jr., R. Casciola, A. Costa, M. Schwerin, E. Shammas, and Howie Choset. A mobile hyper redundant mechanism for search and rescue tasks. *Proc. of IROS*, 3:2889–2895, 2003.

[XXX06] Author XXX. Adaptive dynamics with efficient contact handling for articulated robots. *Technical Report*, 2006.