Physically-based Sampling for Motion Planning

Thomas Russell Gayle

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2010

Approved by:

Dinesh Manocha, Advisor

Ming C. Lin, Reader

Patrick G. Xavier, Reader

Mark Foskey, Committee Member

Nancy Amato, Committee Member

Abstract

Thomas Russell Gayle: Physically-based Sampling for Motion Planning.
(Under the direction of Dinesh Manocha.)

Motion planning is a fundamental problem with applications in a wide variety of areas including robotics, computer graphics, animation, virtual prototyping, medical simulations, industrial simulations, and traffic planning. Despite being an active area of research for nearly four decades, prior motion planning algorithms are unable to provide adequate solutions that satisfy the constraints that arise in these applications. We present a novel approach based on physics-based sampling for motion planning that can compute collision-free paths while also satisfying many physical constraints. Our planning algorithms use constrained simulation to generate samples which are biased in the direction of the final goal positions of the agent or agents. The underlying simulation core implicitly incorporates kinematics and dynamics of the robot or agent as constraints or as part of the motion model itself. Thus, the resulting motion is smooth and physically-plausible for both single robot and multi-robot planning.

We apply our approach to planning of deformable soft-body agents via the use of graphics hardware accelerated interference queries. We highlight the approach with a case study on pre-operative planning for liver chemoembolization. Next, we apply it to the case of highly articulated serial chains. Through dynamic dimensionality reduction and optimized collision response, we can successfully plan the motion of "snake-like" robots in a practical amount of time despite the high number of degrees of freedom in the problem. Finally, we show the use of the approach for a large number of bodies in dynamic environments. By applying our approach to both global and local interactions between agents, we can successfully plan for thousands of simple robots in real-world scenarios. We demonstrate their application to large crowd simulations.

Acknowledgments

When I first walked onto the campus at the University of North Carolina, I remember thinking it would be a great place for me to take the next step in my education. It certainly was a great place for this, but I never could have imagined that it would have such a significant impact on who I am and on my life today. The Department of Computer Science, the University, and even the town of Chapel Hill has been a wonderful and supportive environment. I met some amazing people, and made some of my best and closest friends while working towards my academic goals.

I really do feel fortunate to have had a great set of people supporting me over the years and in my endeavors. I would like to extend a great amount of thanks to my advisor, Dinesh Manocha. His support and persistence never faded, even through some fairly trying times. I owe him a great amount of gratitude for his help, his advice, and his support of my work. I could never have gotten this far without him, as both an individual and a researcher. I also want to thank my co-advisor, Ming Lin, whose guidance helped me through many difficult places and who always helped me reach out and network with the academic community. I want to extend a special thanks to my committee members. Many thanks to Patrick Xavier for being a collaborator and mentor during my summer at Sandia National Labs and for many interesting discussions and useful feedback on my work. Also, many thanks to Mark Foskey and Nancy Amato for their support and their feedback over the past few years.

I'm extremely thankful for the Department of Energy's High Performance Computing Fellowship, the Krell Institute, and its staff. The Fellowship and Krell opened my eyes to many opportunities and a wealth of interesting research projects. The summer I spent at Sandia National Laboratories was a wonderful experience and definitely helped to shape who I am today and what I want to do in the future.

Over the years, it has been a joy and honor to have worked with such a motivated and talented group of individuals. I owe a great deal of thanks to my co-authors Avneesh Sud, Sung-Eui Yoon, Brian Salomon, Naga Govindaraju, Erik Andersen, Ilknur Kabul, Kristopher

Klingler, Stephane Redon, Stephen Guy, and William Moss. Without them, much of this work may not have been possible or would have taken much longer. I'd like to thank the members of the UNC GAMMA Research group for all the feedback, support, and interesting discussions. I also want to thank many members of the Department of Computer Science staff, including Sandra Neely, Missy Wood, and Janet Jones, for all their friendliness and support as well.

One of the greatest joys of my time in Chapel Hill has been all the great friends I have made along the way. I'd like to thank two of my close friends in particular, Avneesh Sud and Sasa Junuzovic. I will cherish the good discussions, late night coffee runs, and especially all the fun times. To Karl Strohmaier, Jason Sewall, Sarah Kennedy, Vince Noel, Tynia Yang, Brian and Carol Begnoche, Jeff Terrell, Keith Lee, Stephen Olivier, Aaron Block, Luv Kohli, and Nico Galoppo, you all are some the best friends imaginable.

Many thanks go out to my parents, T. Vincent Gayle and Theresa Gayle, for their love, their care, their support, and mostly for believing in me and allowing me to grow into the person I am today.

Finally, to my wife Sarita I give my utmost gratitude for her love, her support and for being an inspiration in my life. Here's to the many great adventures to come!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The general problem of motion planning requires finding a path for an agent or agents through an environment, or otherwise reporting that one does not exist. Efficient methods for motion planning are essential to a wide variety of areas including robotics, computer graphics, animation, medical simulation, virtual prototyping. Automated agents, both in the real and virtual domain, are becoming more commonplace. From digital actors, automated vacuum cleaners, to robotic "mules" or packhorses, their ability to solve problems and adapt to new situations and difficult environments plays a significant role in their utility and the level to which they will be adopted.

Generating motions for real or virtual agents, which are coupled to a goal or task, is usually a complicated task. A wide variety of approaches and methodologies have been proposed to tackle the problem. In most cases, these solutions can be viewed as "search" algorithms, where we are seeking to find a way to the goal through some space representative for the problem. Each search space is tailored to the problem itself, which allows for an extremely wide variety of extensions, applications, and even interpretations of the basic motion planning problem.

For example, consider a team of robotic arms on a manufacturing assembly line whose task is to simultaneously paint an automobile. Each arm would need several joints in order to effectively articulate itself to be able to cover any part of the vehicle. The

individual arms must move themselves to precise locations along the automobile's body while also avoiding collisions with other arms, the car and other parts of the assembly line. Furthermore, if the vehicle or its parts are moving along the assembly belt, then the painting end of the arms, their end-effectors, must accurately move with the item in a prescribed manner. To automate this process, algorithms must be able to quickly determine the sequence, or collectively a path, of joint angles that each arm must follow to both reach the piece. Finally, it needs a motion controller to execute that sequence (See Fig. 1.1).

Figure 1.1: Car manufacturing plant, robotic assembly line, elevated view: Courtesy of Getty Images. http://www.gettyimages.com/detail/LA4074-003/Riser, 2009.

For a different viewpoint on generating motion, consider the task of animating a crowd of humans in an urban environment. Each human or virtual agent needs to be able move around while also avoiding other agents or obstacles in the scene. An animator could solve this task by painstakingly moving each agent through the scene. However, in many cases where the goal is only to have a crowd that moves around, motion planning

and simulation algorithms could be used to determine this locomotion and to provide goals for each agent. Simple equations for pedestrian dynamics, mathematical models which describe how humans to tend to move amongst each other, can model the agent's physical properties, how it interacts with its neighbors, and in which direction it should proceed next. (See Fig. 1.2).

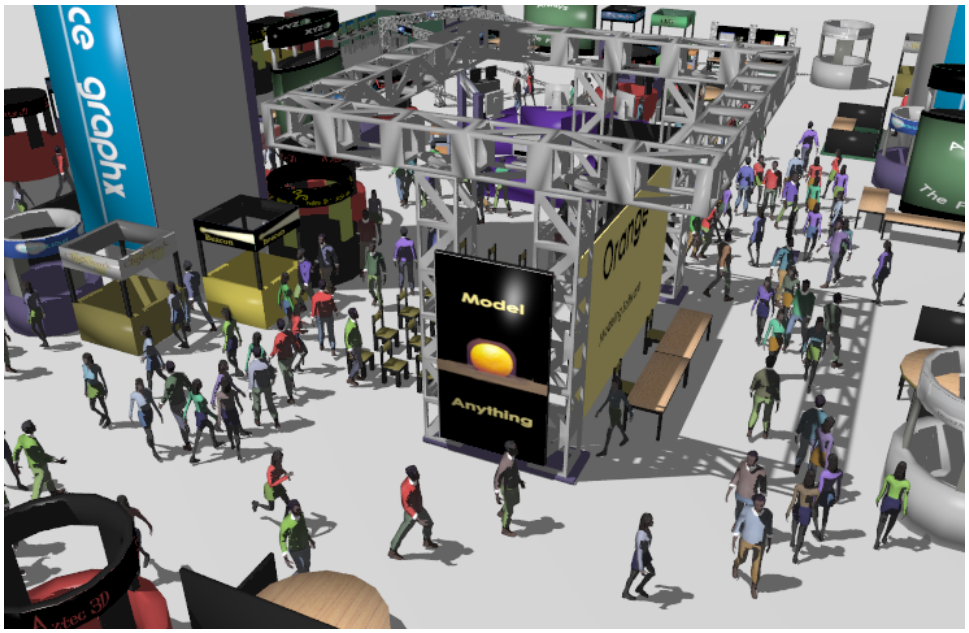

Figure 1.2: Trade show simulation: In the tradeshow scenario, each agent has a series of goals which it must visit. Realistic movements and interactions between humanoids is critical to creating a compelling and realistic simulation.

While these situations and their goals are mostly distinct, one generic similarity is that they share a set of rules govern their motion. For instance, in the robot arm example, every aspect of the generated motion is found and specified entirely by the "rules" of a search algorithm and its resulting path. As such, a geometric interpretation of this search only considers the geometry of the arm and disregards any masses or moments. On the other hand, in the crowd example, the general motion is prescribed by work on pedestrian dynamics and basic path planning. Instead of following a precise path, agents in a crowd follow a more general set of rules that govern their motion and interactions.

These two simple examples represent two ends of a spectrum on how motion can be considered and generated. An algorithm can precisely control all aspects of the situation, or it can employ a predetermined set of laws or rules which influence the motion and behavior. It is known that there are a wide variety of technical challenges at both ends of this spectrum, resulting in consideration attention from the related research communities. With precise control, it has been shown that the time required to find a solution grows intractably as the complexity of the planning problem, the underlying robot, or the scene increases. Additionally, the resulting motion is often considered 'un-natural' or 'un-intuitive' when observed by humans. In comparison, modern physically-based dynamics simulations (which are suitable for handling and applying sets of "rules" or motion equations) are capable of handling tens of thousands of objects or agents in complex environments (e.g., NVIDIA®PhysX®), but usually at the expense of fine-tuned control or control beyond some local region of the objects.

A solution in the middle of the "control versus predetermined laws" spectrum would be beneficial and help to overcome of the drawbacks of each. A robotic arm empowered with control criteria, goals, and a physical "state" no longer requires explicit control of all of its joints. By modeling the joints as physical bodies, a path could be generated where these joints move as needed based on motion equations as the end effector moves. With additional control, the artist would be able to provide specific goals for individual members of the crowd to better direct its motion while the general crowd moves and adapts appropriately. In general, realistic extensions to many planning problems can utilize these ideas and improvements.

In this thesis, we focus on integrating precise control with generalized rules of motion for motion planning of single (rigid, articulated, or non-rigid) or multiple agents. Our goal is to utilize physics simulations to relax the requirement of precise control of many degrees of freedom, and instead allow the agents or their parts to move toward the goal through the use of artificial physical forces acting on the agents. We call this process

physics-based sampling, because a system of forces and constraints are applied to the the agents in a physically-based manner to bias the search toward the goal. Control, when needed, is gained through integration of these forces with simplified paths through the search domain. We show that a physics-based sampling framework can overcome many of the limitations of prior approaches and show how it can be used in a variety of application including medical simulation, hyper-redundant robots, and crowd or social simulations.

1.1 Motion planning background

Since motion planning fundamentals are used throughout this work, this section first introduces key concepts and ideas and traditional approaches in motion planning. Then, we discuss extensions to the basic planning problem as related to this work and their inherent challenges.

1.1.1 Definitions and Notation

Consider a robot r in an environment or *workspace* $\mathbf{D}$ embedded in $\mathbb{R}^2$ or $\mathbb{R}^3$. Within the workspace is a set of obstacles, $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n\}$. Traditionally, robot r is a rigid or an articulated body composed of rigid parts and is the only object allowed to move in $\mathbf{D}$, the remaining obstacles and environment are completely *static*. A robot's *configuration* $\mathbf{q}$ is the set of variables or parameters which fully determine all points on the robot's body. This is often the position and orientation of robot as well as that of its sub-parts, but any parameterization that completely models the agent can be used. The number of configuration parameters is the number of *degrees of freedom*, or DOFs, of the robot. The set of all possible configurations defines a robot's *configuration space*, $\mathcal{C}$ [LP83]. In many cases, there are constraints which prevent a robot from moving into a particular configuration, such as collisions with an obstacle or limits on the range of

motion. These constraints are mapped into $\mathcal{C}$ as illegal or forbidden regions. The space outside of these regions makes up the robot's *free space*, $\mathcal{F}$. A path through $\mathcal{C}$ is a function $\tau : [0, 1] \rightarrow \mathcal{C}$ between configurations $\tau(0)$ and $\tau(1)$. A path is *collision-free* if for every configuration $\mathbf{q}_t$ on τ, $\mathbf{q}_t \in \mathcal{F}$ (See Fig. 1.3).

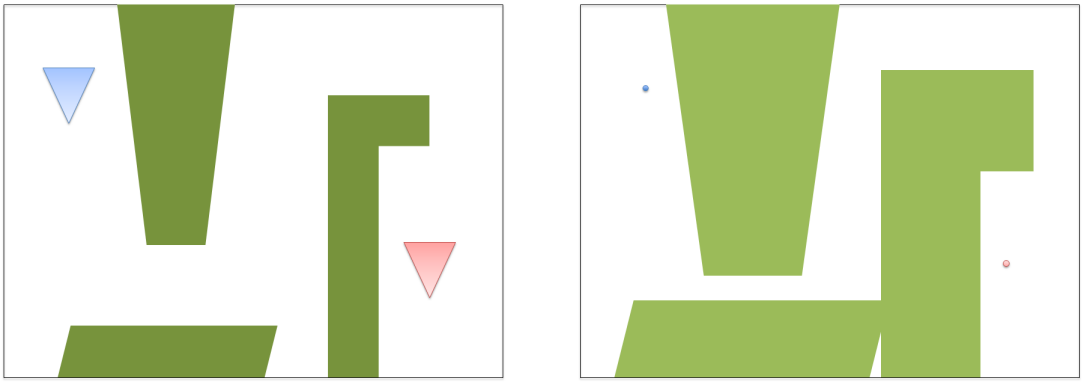

Figure 1.3: Workspace vs Configuration space: This planning situation requires the triangle to move translationally (no rotation) from the blue to the pink position. The left image contains the start and final configurations, while the right image maps it to its configuration space. The initial and final configurations can be represented as a point in $\mathcal{C}$ while the $C-$obstacles have been enlarged. Here, the $\mathcal{F}$ is the white area on the right image.

Using these definitions, we pose the basic motion planning as follows: Given a robot r in workspace $\mathbf{D}$ and its initial and goal configurations, $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, respectively, find a collision-free path τ such that $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$, or report that such a path does not exist.

1.1.2 Prior Approaches

A variety of approximations have been proposed and can usually be classified into a few categories.

- **Complete motion planning and cell decompositions**: Many of the early approaches in motion planning compute an exact representation of the freespace, $\mathcal{F}$. These solutions are complete, in that they return a path if one exists or otherwise

report that no path exists. One of the more popular approaches for this is cell decomposition planners [SS83, BLP85]. The idea of cell decomposition planners is that the configuration space can be divided into regions of non-overlapping cells. The size and type of cell is usually chosen such that a path exists between any configurations within a cell and it can be determined whether or not the cell is in $\mathcal{F}$. After determining all of the "free" cells, a connectivity graph is generated by using the adjacency relationships between cells. To solve a planning query, the algorithm first identifies which cells contain the initial and goal configurations and then performs a graph search through the connectivity graph between these cells. The final path is extracted from the returned sequence of cells.

Other complete motion planning approaches, often also called criticality-based algorithms, include computing the free space for a specific class of robots [LPW79, Don84, Hal02] and exact roadmap methods [Can88a]. However, due to the complexity of exact geometric queries for higher degree of freedom problems, there are no known practical and efficient implementations of these complete approaches [HH03]. Several hybrid planners which include cell decompositions have been proposed as well, many of which are complete [VM05, ZKM07]. Others are considered resolution-complete and work by subdividing cells which are known to be both "free" and contain a configuration space obstacles. Thus, for a sufficiently small cell, they can report whether or not a path exists (See Fig. 1.4)

- **Roadmaps**: Roadmap approaches represent the connectivity of $\mathcal{F}$ by a graph or network of configurations and $1D$ collision-free curves connecting these configurations. A roadmap is first constructed by sampling configurations in $\mathcal{F}$ (usually in a randomized or otherwise prescribed manner) and then determining a local path between these configurations. Given an initial and goal configuration, the planner first determines how to connect each of these configurations to the roadmap. Then, a graph search is performed on the roadmap to return the final path. The success

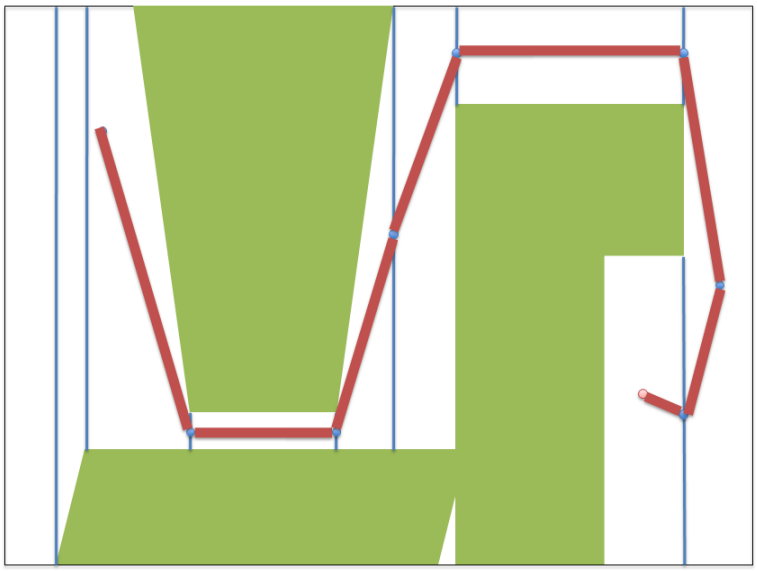

Figure 1.4: Cell decomposition: A vertical cell decomposition of the $\mathcal{C}$ is computed by extending the vertices of the $\mathcal{C}$ obstacles vertically to the environment boundary. A path is created by connecting the midpoints of adjacent cells.

of a planning query relies on the quality of the roadmap: how well it captures the global connectivity of $\mathcal{F}$ and on how easily any configuration can be connected to the roadmap. Recently, randomized approaches of roadmap generation (Probabilistic Roadmap Methods [KSLO96], Rapidly-exploring Random Trees [LK00]) have been very successful at solving a wide range of complex problems and have also been shown to be probabilistically-complete (See Fig. 1.5).

- **Potential fields**: These planners exploit the idea that *artificial potential functions* defined over the configuration space can be used as a heuristic to direct the path search. Navigation potential functions are usually designed as a summation of an attractive potential (or a sink) at the goal configuration, and repulsive potentials (or sources) for each obstacle in the environment. For example, a simple potential

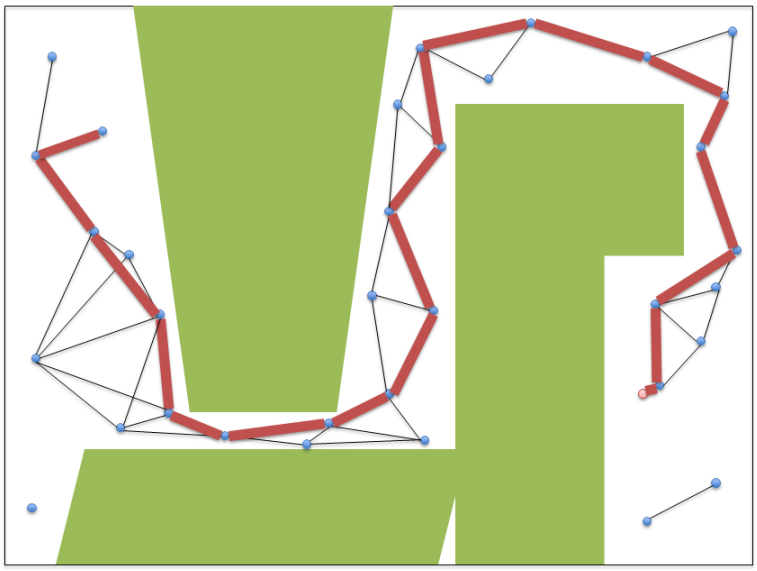

Figure 1.5: Roadmap: A roadmap is created by randomly sampling the $\mathcal{C}$. Each sample in $\mathcal{F}$ is then connected to the nearest neighbors on the roadmap. To find a path, the initial and goal configurations are connected to the roadmap. Then, the shortest path between the configurations is found and used as the path.

function for a robot with a configuration $\mathbf{q}_t$ at time t may be:

$$U_r(t) = U_{\mathbf{q}_{goal}}(t) + U_{\mathbf{O}}(t)$$

where

$$U_{\mathbf{q}_{goal}}(t) = \frac{1}{2} k_p (\mathbf{q}_t - \mathbf{q}_{goal})^2$$

and

$$U_{\mathbf{O}} = \sum_{\mathbf{o}_i \in \mathbf{O}} \frac{1}{2} k_o \left(\frac{1}{d(\mathbf{o}_i, \mathbf{q}_t)} - \frac{1}{\rho}\right)^2$$

where $U_{\mathbf{q}_{goal}}$ is an attractive potential to the goal, and $U_{\mathbf{O}}$ is the total potential away from obstacles, k_p is a potential gain for the goal, k_o is a potential gain for the obstacles, ρ is a limit on the distance of the potential field influence, and $d(\mathbf{o}_i, \mathbf{q}_t)$ is the shortest distance between $\mathbf{q}_t$ and $\mathbf{o}_i$. The gradient at any configuration in

the potential field points toward lower potentials and thus away from obstacles and toward the goal. In many cases, the potential function can be sampled locally and does not need to be entirely precomputed. Similarly, this often lends itself to real-time executions since the potential field at a point can usually be determined very quickly [Kha86]. However, one well known drawback of this approach of potential field motion planning is that the field could contain local minima. If a robot moves into one of these regions, it effectively becomes stuck. A variety of approaches help reduce the likelihood of this situation such as construction of a potential field which does not contain any local minima [RK92] or by adding Brownian motion or a random walk to the motion at each stage [BL91]. While the exact approach is limited by the number of degrees of freedom in the problem, the Brownian motion solution has been shown to work effectively for tens of degrees of freedom. Due to the randomization, it is also *probabilistically-complete*, or will find a path if one exists given sufficient time. In general, probabilistic completeness relaxes the requirement of path existence. (See Fig. 1.6).

While most work falls into the previously described categories, there are a few approaches which try to leverage the advantages and disadvantages of several of these ideas. For instance, a coarsely defined roadmap does a good job of capturing the general connectivity of an environment, while a potential field can help overcome difficulties in navigating between the connected components. A roadmap could also be extracted from a potential field by identifying the local minima checking if they can be connected by a simple "link".

The true success of any of these approaches is often measured by how large or complex of a problem it can solve, and how quickly a solution can be found. While each of these classes of planners or solvers have been extended to solve a wide variety of problems of varying complexity, each has some inherent drawbacks. For instance, while the potential field approach generally cannot easily find a shortest path, it often picks a path which

Figure 1.6: Potential field: Since all obstacles here are given a repulsive or high potential value, this potential field path causes the robot to try to stay optimally far from obstacles as it navigates toward the goal.

is further away from obstacles. Furthermore, it is often perceived as being smoother or more natural since the resulting path is smooth. The next two subsections discuss additional topics: the range of problems which planning approaches can be used to solve and the related complexity of these planning problems.

1.1.3 Extensions to basic planning

In many realistic or modern planning scenarios, the problem's assumptions do not match those of the basic motion planning problem. A wide variety of extensions to the basic problem have been proposed. Since this work relates to several of them, we provide an overview of the extensions related to our work here.

- **Dynamic environments**: In many scenarios, it is more realistic if the robot is not the only entity moving in a scenario. For this extension, obstacles $\mathbf{o}_i \in \mathbf{O}$ are no longer static and can either move on their own or be moved by a robot.

If the motion of a dynamic $\mathbf{o}_i$ is known a priori, the configuration space can augmented to include a temporal dimension t where $\mathbf{o}_i(t)$ describes the state of the obstacle at any point in time. On the other hand, if the motion of $\mathbf{o}_i$ is not predetermined or is influenced by interactions with the robot, then the planning algorithms must be able to quickly adapt or replan as time proceeds to account for changes in the environment. If the obstacles can be moved by the robot, the planner could consider how an obstacle must move or how the moved obstacle changes the connectivity when planning a path.

- **Differential constraints**: The basic planning problem requires finding a path based only on the geometry of the robot. Other factors can influence or otherwise restrict the way a robot can move. For instance, a wheeled robot typically has a limited turning radius and moves based on the directions of the wheels. In other cases limits on velocity and acceleration require second order constraints (over time) to be maintained. These types of constraints are often generically classified as differential constraints. A planner needs to consider a different parameterization of space which includes the robot's *state* rather than just its geometric configuration, which includes additional information such as a robot's velocity or acceleration. Planning algorithms must then consider the robots state-space $\mathcal{S}$ and any state-space obstacles. Finally, other factors such as avoiding *inevitable collisions states*, or states in which the momentum of the body will cause a collision in the future, should be considered. This category is often broken down into three subcategories: nonholonomic planning, kinodynamic planning, and trajectory planning. Briefly, nonholonomic planning is primarily mentioned in regards to wheeled robots, and involves constraints which cannot be completely integrated. Kindoynamic planning usually requires velocity and acceleration bounds to be satisfied while also avoiding obstacles, thus the second order dynamics of the motion are included in the resulting path. Finally, trajectory planning often refers to

determine a sequence of positions along with velocities (or velocity profiles) for a robot that satisfy the dynamics constraints.

- **Multiple agents**: In many real and virtual world scenarios, a team of robots must work together to solve a problem, or several agents will move in close proximity to each other to individually reach their goals. For both of these, the basic problem needs to be extended to include several robots. One view of this extension is that it is very similar to unpredictable obstacles, each robot is an unpredictable obstacle to every other robot. However, unlike obstacles, each agent or robot can be controlled. In another view, groups of robots or the entire population of robots can be treated much like a single robot in a "combined" or "aggregated" configuration space. Then, motion for the multiple agents can be generated in a similar manner to the standard planning approach.

- **Deformable or non-rigid robots**: With recent improvements in simulation and robotics technology, it is no longer a requirement to have rigid agents or robots. For instance, consider motion planning for a flexible fire hose or rubber tube, a bendable needle, or any other soft-bodied or otherwise pliable object. Also, consider a situation where a robot needs to move or adjust a deformable object. Since any point on the surface can be moved or deformed relative to any other point, the parameterization of the space is often much more difficult. In practical implementations, the bodies are discretized as a set of volumes or control points. For a higher resolution deformable body, this leads to a very large number of control points and extremely large configuration spaces (and intractably long planning times). Furthermore, planners for general deformable robots must consider soft-body constraints such as volume and total or degree of deformation as part of the planning and simulation process.

1.1.4 Motion Planning Challenges

Advances in motion planning methods and algorithms have had a difficult time keeping up with growth of complexity of many current systems or situations. For instance, medical practitioners could make use of a deforming catheter-like robot to aid in performing various procedures. Related simulations could also help as a teaching tool or for preoperative planning. Hyper-redundant robots such as snake-like robots or self-reconfigurable modules may contain hundreds or possibly thousands of links and joints. Fleets of mobile robots sizing from the tens to hundreds are already used for managing warehouses. Autonomous vehicles will need to know how to avoid and navigate among many other moving obstacles. Virtual reality developers can populate their environments with many agents each of which have their own goal, task, or behaviors. All of these examples are very difficult to handle in the current state of motion planning algorithms.

The complete robot motion planning problem has been shown to be exponential in number of DOFs of the robot [Rei79, Can88b]. As previously mentioned, many modern and popular approaches employ randomization to quickly generate an approximate and hopefully representative mapping of the $\mathcal{C}$, and they also relax the completeness requirement in the process. These algorithms have enjoyed a great deal of success over the last decade due to their ability to solve a wide variety of problems in higher dimensional, or high DOF, configuration spaces in a more practical amount of time. Moreover, these randomized algorithms are generally intuitive and relatively easy to implement.

As the complexity of the robot or its environment increases, motion planning becomes more and more difficult. When considering problems with deformable robots, highly articulated robots, or multiple, complex robots such as those previously mentioned, the effectiveness of most motion planning algorithms rapidly deteriorates. By nature of randomized algorithms, the motion itself may not be even be realistic. Most of these approaches generate motion by linearly interpolating between states rather than preserving any kinematics, dynamics, or differential constraints. As a result, the motion

is not necessarily physically plausible, is not smooth, or may contain motion paths which are un-intuitive. There are some approaches which plan in the "state" space of a robot, sometimes referred to as "kinodynamic" planning [HKLR02]. However, since kinematic and dynamic constraints increase the dimensionality of the space which needs to be searched, they typically do not scale well to higher DOF planning scenarios.

Based on these considerations, we identify three important sources of complexity in motion planning for realistic, and more generic, scenarios:

1. **Agent complexity**: The complexity is inherent to the robot itself or to its geometry. For instance, for a highly articulated robot with thousands of joints and possibly many branches or kinematic loops, the $\mathcal{C}$ has an extremely high dimensionality. As mentioned in Sec. 1.1.3, for a completely deformable robot the $\mathcal{C}$ may be intractably large. In both cases, the complexity of the robot is typically too great for standard planning approaches. Finally, to ensure smooth and realistic motions, dynamics and kinematics for these robots should be considered. Since each parameter in the "state" space adds additional DOFs to the problem, these constraints can also greatly increase the overall problem complexity.

2. **Numerous agents**: Many real world and virtual planning situations include multiple moving bodies, or multiple agents. A complete, *centralized* approach to this situation treats the DOFs of all the robots as a single robot, i.e. a single configuration describes the complete position and orientation of all robots and their sub-parts. Under this formulation, the problem is identical to the single-robot planning problem and can be handled as such (via any of the approaches mentioned earlier). Since the number of DOFs grows linearly with the number of robots, the inherent complexity of the planning problem becomes intractable with just a few robots. *Decentralized* approaches treat each robot individually; they plan for each robot individually and then perform a coordination planning step to try to ensure that robots do not collide as they execute their paths [KZ86, OLP89].

The latter solution is usually faster, but gives up the completeness property of a centralized algorithm and in practice has been shown to have a much lower success rate [SI06].

3. **Dynamic environments**: Finally, in many cases, there are other obstacles or items which are moving or are otherwise changing during the execution of the motion. These dynamic events and environments add complexity to the planning situation itself, and present additional challenges. In cases where the motion of obstacles is unpredictable, it adds a real-time constraint. The robot has a finite amount of time to decide how to avoid or adapt to a changing environment. In many cases, this requires the robot to quickly re-plan or otherwise adapt its current plan. Since planning is already a complex task, this poses a large challenge for even moderately complex environments. When the obstacle motion is known a priori, this adds a temporal dimension to $\mathcal{C}$ which makes planning more computationally expensive due to the larger search space.

In many real and virtual world situations, many of these constraints may exist even in a single situation or scenario. For instance, a city scene may include navigating agents which must also avoid moving vehicles. While algorithms do exist which can appropriately handle for each of them individually, the ability to do this successfully and in a reasonable amount of time diminishes as the problems become more complex. Even for moderately "realistic" problems, most modern motion planning algorithms tend to take a long time or do not respond in a practical amount of time.

1.2 Constrained Simulation

Motion planning has enjoyed a great deal of success in providing specific, goal-oriented automated control. However, the computational cost of that degree of control is very

high. On the other hand physics simulators, systems or software which can adequately model physics and physical interactions, have seen successes in handling large numbers of objects in a general, unified manner, in a physically realistic manner, and in a relatively faster amount of time.

With regards to robots, agents, and environments, many simulation environments provide a uniform set of "motion equations," or predefined rules or laws which dictate the motion of a body, based on well established laws such as those of classical dynamics. This permits any object in a scene to behave in a physically-plausible way: they accelerate, torque, recognize collisions, and respond to collisions much like one would expect it to respond. *Constrained simulation* augments standard simulation environments by restricting or otherwise influencing the way an object can move. For example, consider a hinged joint with two pieces, such as one that allows a door to open and close. One solution to generate motion in this situation is for all the interactions between both parts of the joint to be modeled. Since they are always in contact, there would be an infinite number or extremely high number of contacts each of which would need to be resolved. Instead, since we know by design that both parts of the hinge will always be in contact, we can ignore these complex interactions and instead constrain the motion of the two pieces to behave in this manner, like an articulated body. This greatly reduces the computational complexity while still allowing for realistic simulations.

In this sense, constrained dynamics allows a body to follow and obey the established motion laws and equations while also requiring it to obey the defined constraints and restrictions. In a way, it gives more "control" over a simulation but still benefits from the generalized simulation framework. This aspect of constrained simulation makes it a good fit to aid in motion planning.

In this section we give a brief overview of a general simulation framework, describing the primary tasks which a simulator needs to implement. A more complete treatment with respect to this work is given in Chapter 3. Finally, we discuss the challenges and

limitations in using constrained dynamics specifically for the task of motion planning.

1.2.1 Physically-based simulation

Most physics and dynamics simulators share common elements, regardless of what they are attempting to simulate. This includes a set of motion equations which dictates how the body moves given a set of external (and sometimes internal) forces, a collision or interference query system which finds the set of overlapping geometry, a collision response system which determines how to respond or react to collisions, and finally a numerical solver which is used to evolve the state of the environment in time.

For example, consider a simple particle: a point in space with a mass. Its motion can be most easily described via the Newton-Euler laws of motion. Collisions are detected when the particle crosses any surface or plane in the environment, and a response can be calculated by simply reflecting its currently velocity (and eliminating any acceleration in the direction of the surface with which it collided). Finally, Euler's numerical integration, or any other numerical integration technique, can be used to solve the motion equations and to update the state of the particle.

While this is a simple example, nearly all physical simulators have a similar framework. The motion of rigid, articulated, and even deformable bodies can be described by some equations. A wide variety of collision detection and response methods have been studied for each of these classes of bodies, and a wide range of methods are available to update the state. Regardless of the choice of components, the end result is a simulator which reflects whatever behavior is encoded into the system.

1.2.2 Constraint-driven simulation

Standard physics simulators have been successful used in numerous applications, mainly when the emphasis is on the evolution of the motion or the trajectory of objects in a system rather than directly controlling the objects themselves. In general, the idea of

constraint-driven or constrained simulation is that we want to be able to do more than apply external forces, we want to be able to actually restrict the way the bodies are allowed to move in a scene or environment. This is an important concept in general, as it paves the way for more compelling and efficient simulations. Constraints allow us to connect two bodies via hinge, require that certain bodies stay a fixed distance apart, ensure that no pair of bodies are overlapping or interesting, and many more.

In the scope of this work, constraints generally fall into two different categories. *Hard* constraints are firm restrictions on the motion or state and must be enforced at all times whereas *soft* constraints are used to influence the motion or behavior at a given state, but are not required to be maintained at all times. Thus, hard constraints directly remove or changes properties of the body or its forces whereas soft constraints act alongside (or in opposition to) external forces. When combined, these constraints greatly enhance the capabilities of a simulation.

For example, consider a "point-distance" constraint between a pair of points $\mathbf{x}$ and $\mathbf{y}$. This *hard* constraint could be useful for fixing two points on a robot or ensuring that the robot stays a fixed distance from some other object. Mathematically, this constraint can be defined as $\mathcal{C}_{dist}(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}|| - d$ for some specified distance d. During execution, if $\mathcal{C}_{dist} \neq 0$, then let δ_d be the difference between d and the actual distance. We can move each body a distance of $\frac{\delta_d}{2}$ toward each other to enforce the constraint or apply a force that would cause the same change in position.

Alternatively, consider a "goal attraction" *soft* constraint. This constraint can be defined such that it will only be satisfied once the robot, or whatever part of the robot the constraint is defined on, reaches its goal. A penalty force for this constraint would point directly toward the goal or along a path, and could be applied to the entire robot or to whatever part of the robot needs to reach the goal.

Many more details and specific examples of hard and soft constraints will be developed and described throughout this work.

1.2.3 Simulation loop

To this point, we have discussed the main concepts for a constrained simulation. The simulation loop ties these components algorithmically in order to generate physically-plausible motion.

Note that we have not yet put all the pieces together into a single, unified system. Many related works describes a simulation core or a simulation loop to achieve this. The following are the general steps in a single step of a simulation core:

1. **Clear the force accumulators**: Each body or its components maintains a total of all forces on it. At the beginning of each step, we clear the forces from the previous step.

2. **Detect collisions**: Loop over all bodies and determine any contacts or collisions in the scene and prepare them to be resolved.

3. **Compute external forces**: Loop over all external forces, including contact or collision resolution forces, and add them to the force accumulators.

4. **Compute constraint forces**: At this point, each body or component knows the total force acting upon it (its in force accumulator). To process the constraints, first add any soft constraints to the accumulators. Finally, compute and apply the hard constraint forces.

5. **Compute derivatives**: Gather derivatives as needed to prepare to update the system.

6. **Integrate and update the state of the simulation**: Use a numerical integration method to update the state of the system by some small time increment.

1.2.4 Constrained simulation challenges

Constrained simulation is a powerful framework for generating motion of many bodies. A simple set of rules and constraints can be defined which will handle an extremely wide variety of situations. As such, it has become an extremely popular tool in interactive computer graphics, animation, and even in robotics. However, there are several challenges and complications inherent to physics simulation with respect to the goal of this work.

- *Computational bottlenecks*: A wide variety of geometric and numerical methods are used to determine the state of the bodies in the environment, interactions between the bodies, their derivatives, and finally to advance the simulation in time via numerical integration. Since these steps must be performed at every step, the performance of a simulation dependent on the complexity of these methods. Algorithms for the simulation core must be chosen to not only be accurate, but also efficient, in a wide range of applications.

- *Motion per step*: Even if the simulation core is efficient, the overall simulation can still take a great deal of time. For instance, if our time step is extremely small or if the simulation's goals require a the body to travel a great distance (relative to the maximum amount it can travel per iteration) then the overall simulation can still take a great deal of time. Thus, simulation core algorithms should try to allow for large steps either in time or motion available.

- *Stability*: A well known drawback of several numerical integration techniques is their instability. In particular, many integration methods suffer from *stiff* systems and when the time step is too large. Compelling simulations need to be aware of the complete range of interactions possible in order to try to avoid this state. While higher-order explicit, semi-implicit, implicit numerical integration and tuning of

simulation parameters can help overcome some of these drawbacks, these can also significantly impact performance.

- *Constraints*: In general, there is a standard set of constraints like non-penetration and collision response we can be included in any simulation. However, many simulations, such as hinges or joints as mentioned earlier, require specific additional constraints. Thus, to some degree, every problem must be evaluated for which constraints are necessary. Furthermore, many of these constraints include constants which must be tuned depending on the range of interactions and other dynamics of the simulation.

- *Accuracy and error*: To help improve performance, many portions of a simulation core are often approximated. The time-integration methods and computation of derivatives are examples of methods that are often approximated or that can include some additional error. Given the simulation loop, these errors can accumulate over the duration of a simulation. Thus, components of the simulation core must be precise enough to either correct for errors or keep them at a reasonably small level, but fast enough to be able to solve problems within a reasonable amount of time.

While these topics each present challenges for simulation, they are common across most simulation environments. Due to the popularity of simulation or simulation-related techniques, a great deal of attention has been given to these topics to help alleviate or reduce the impact of them. Nonetheless, they are important considerations in using simulation as part of general motion planning.

1.3 Goals

In the previous sections, we describe two variations on ways to control motion of bodies as they move between points or configurations in a workspace. Robot motion planning

algorithms offer formal, proven methods to determine a geometric path through a the problem's configuration or state space to its goal. On the other hand, constrained simulation provides a powerful set of tools for physically-plausible motion of a range of bodies in complex situation.

By themselves neither method is completely capable of providing practical solutions for finding paths for complex agents in complex environments. Motion planning algorithms are intractable even for relatively simple situations and often the motion they generate is neither smooth nor realistic. Simulation methods are usually posed solely to allow the system to evolve uninterrupted or with minimal user input. As such, they lack the specific, centralized control to generally allow for generating paths even for relatively simple objects.

With regard to complex agents and workspaces, each method tends to account for what the other lacks. The goal of this work is to integrate these methods to solve a wide variety of motion planning problems which were previously not possible. The simulation core can be used to manage the complexity of the scene, while planning algorithms can be used to control the motion in the scene.

1.4 Thesis

Coupling traditional motion planning methods with constrained simulation through physics-based sampling enables the generation of more realistic motion trajectories for high degree of freedom articulated robots, deformable robots and for multiple robots in complex environments in a practical amount of time.

1.5 Main Results

In this section, we present the main results of this dissertation. With regards to motion planning and simulation, these fall into two main categories: a general motion planning

framework incorporating physics-based sampling and applications of the planning framework. We also present several optimizations to improve the performance and capabilities of the simulation core for specific, planning-related circumstances.

1.5.1 Physics-based Sampling

Many standard motion planning techniques generate motion which is neither "smooth" nor physically-plausible or realistic, nor do they scale to higher DOF problems. For example, consider the roadmap and cell decomposition solutions presented earlier (Figures 1.4 and 1.5). Neither path is smooth, and in the case of the roadmap, the path will cause the robot to move around haphazardly.

To cope with these drawbacks, we propose the use of a physics-based sampling framework. Briefly, a constrained simulation-based motion planning framework is used to generate samples and bias the search direction. Rather than randomly sampling in state or configuration space, our framework uses workspace distance information and artificial constraints in order to determine the next state-space sample and thus grows polynomially in terms of run-time complexity with respect to the number of DOFs. This approach can be seen as similar to potential fields, where we greatly augment our potential function with state space information and additional planning constraints. This has the following advantages over traditional approaches:

- **Smooth and realistic**: By using a simulation core, we ensure that the generated step adheres to the motion equations which define the simulation and subsequently to the motion of the bodies. Since typical motion equations include second-order information, it approximates a smooth curve. Since the equations also model the real world, the motion generated tends to be physically-plausible. Thus, our framework produces motion sequences which in practice are both smooth and realistic.

- **Scalability**: The complexity of this approach is dependent upon the simulation components and the planning constraints. Many of these components involved in the planning and sampling have linear time complexity, or at worst quadratic, in the number of DOFs. Therefore, the approach scales well in the number of DOFs when compared to most prior work which grows exponentially with the size of the search space.

In general, the physics-based sampling framework can be broken down into two stages. First, a general path or roadmap is determined from a simplified model of the robot or robots based on workspace information, rather than configuration or state space computation. This path or roadmap will serve as a guiding path or guidelines to bias the direction of the physics-based sampling direction. Next, in an execution phase, the robot or robots use the simulation core to proceeds along this path. The core determines the direction to final direction to move in, taking planning as well as physical, mechanical, and any dynamical constraints into account.

1.5.2 Motion planning with physics-based sampling

The physically-based sampling framework has numerous advantages of traditional planning methods and could be used to complement others. We propose several novel solutions for a wide variety of motion planning situations.

Since simulation is a core component of our approach, the performance of the simulation core as well as the quality of the motion generated plays a large factor in both the performance and quality of the resulting motion trajectory. In this work, we focus primarily on optimizations of the simulation components specifically for our planning scenarios and situations. For each extension of the planning problem, we propose novel approaches to improve the overall efficiency and reduce the total amount of computational work.

Deformable Agents

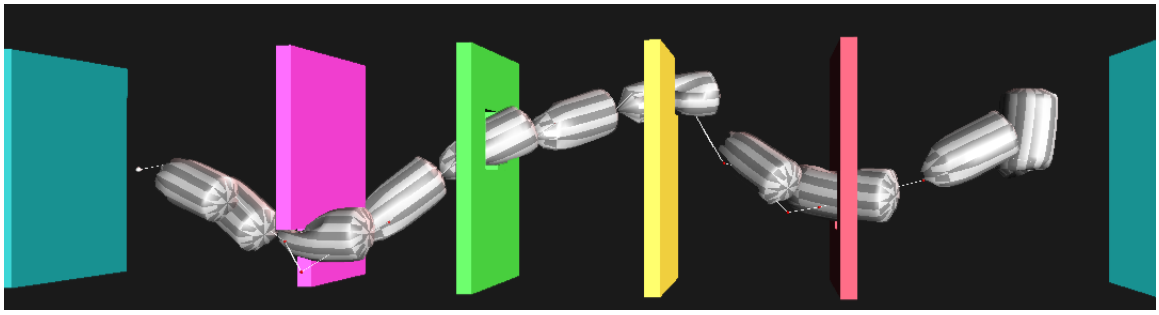

Figure 1.7: Planning of a deformable cylinder: The final path of a deformable cylinder as it travels through a series of walls with holes. The cylinder's radius is chosen so that it must deform in order to reach its goal.

We address the deformable agent extension to the motion planning problem. In one aspect, this problem simplifies collision and orientation handling since a deformable body can likely squeeze through a region rather than having to move a precise configuration. However, the high dimensionality of the configuration space poses a challenge to many approaches.

To cope with unknown or unpredictable deformations, our approach starts by generating a initial trajectory for a point robot through $\mathbf{D}$, the workspace. Akin to prior methods for planning of deformable agents, this can be done with any traditional roadmap planner, such as a probabilistic roadmap (PRM) planner or by extracting the Voronoi graph of the workspace [BLA02c]. Note that this initial trajectory may, and will likely, have collisions with a general environment once the full deformable body traverses the initial path. We use a constraint-based soft-body dynamics simulation in order to incorporate deformation into and control the motion model of the robot. The physics-based sampling framework extends this simulation in order to follow the initial trajectory and to make appropriate path corrections and deformations as needs, yielding a collision-free path (See Fig. 1.7). It should be noted that this general approach works best for agents which are fairly compressible and flexible, otherwise the initial path and following would

need to consider both position and velocity of the agent and any of its constituent parts.

We implement non-penetration, volume preservation, and soft-body constraints for the deformable robot, so that the robot will deform yet preserve its initial form as it comes in close contact with the obstacles in a physically plausible manner. In order to achieve interactive performance, we assume the robot is represented as a closed surface and relax the strict global volume preservation constraint by implementing a local method that sets a threshold on the amount of deformation (e.g. elongation or compression) based on implicit internal pressure fluctuations, simulating it as a soft-shell with internal pressure and surface tension. We demonstrate our planner on several scenarios of varying complexity. In practice, we have successfully applied our approach to environments with tens of thousands of triangles, and robots with just as many springs. Our planner can compute a collision-free path for a deformable robot in a complex environment consisting of tens of thousands of polygons in a few hours 4.2.

To improve performance, we develop a graphics hardware accelerated collision detection method that works particularly well for motion planning problems where the robot is in close proximity to the obstacles, compared to bounding volume hierarchy methods. We present a new algorithm to detect collisions between a deformable model and a complex, stationary environment. We compute a potentially colliding set of overlapping primitives using set-based computations. Our algorithm uses 2.5D overlap tests between arbitrary objects and checks for the existence of a separating surface along a view direction. We use graphics processors (GPUs) to efficiently perform 2.5D overlap tests and we compute offsets and Minkowski sums to overcome image-precision errors. In practice, our collision detection algorithm is significantly faster when compared to prior approaches based on bounding volume hierarchies.

Our planner for deformable bodies based on physics-based sampling has the following properties:

- **Physically-plausible soft body agents**: Our internal pressure model of a soft-

body can represent a wide variety of deformable objects in a physically-realistic manner. The formulation aids in computing collision response while also maintaining the general shape and volume of the body as it travels along its path.

- **Performance and efficiency**: Our graphics hardware solution for collision detection is quick and scales very well with the number of bodies. In cases where the body is usually within close proximity to the obstacles (as is the case in many interesting planning scenarios), we see a significant improvement since boundary volume methods will still require a large number of collision checks. Additionally, we use a quick and accurate solver to help minimize the amount of computational per step on the simulation core.

Finally, to test the usefulness of our approach, we use this planner to find a path for a medical procedure known as liver chemoembolization. Briefly, a physician must navigate a thin, flexible catheter through a network of arteries in order to reach a tumor, at which point a chemoembolizing agent can be administered directly to the tumor. Our physics-based sampling approach can successfully find a path for the catheter while also simulating the way it moves and bends in relation to the artery network.

Highly Articulated Agents

We present an efficient physics-based motion generation scheme for highly articulated robots, particularly in a serial linkage, or chain-like configuration. Like the deformable case, we compute an initial path in the workspace as a guiding path for the articulated body. Then, we use the guiding path to influence the search direction of our simulation core. This assumes that the robot can be guided or controlled via the "head" or end-effector of the robot. In the case of articulated agents with branches, multiple points may need to be considered.

Since the simulation of articulated bodies can be prohibitively slow for long or complex bodies, our articulated body simulation framework exploits the coherence between

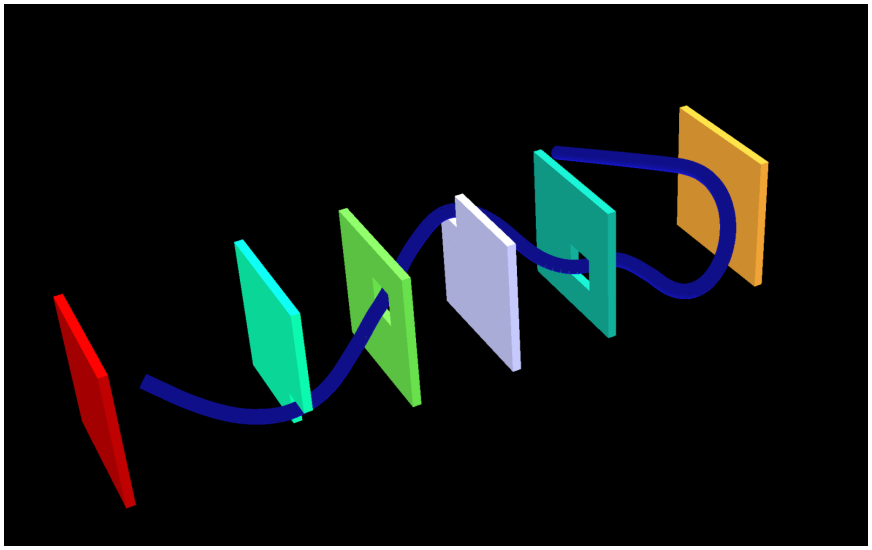

Figure 1.8: Articulated body planning: The final position of the high articulated robot as it reaches its goal. The serpentine robot needed to navigate through several walls with holes in order to reach its goal.

neighboring joints to attempt to reduce the dimensionality of the problem through the use of "adaptive forward dynamics" [RGL05]. Based on motion metrics defined over joint velocities and accelerations, we prioritize the joints to determine which ones capture the majority of the motion of the chain. Adaptive forward dynamics takes advantage of this formulation by only simulating these joints which effectively produce motion in a reduced dimension and results in improved efficiency. Therefore, the complexity of simulation portion is *sub-linear* in the number of joints. Moreover, we discuss limitations on the error associated with the approach and usability of the resulting solution when compared to earlier planners for the same robots (See Fig. 1.8).

We also present a novel and fast contact handling algorithm for *adaptive dynamics* computations of highly articulated robots. We exploit the structure of the hybrid tree representation introduced by the adaptive dynamics algorithm [RGL05] and show that we can also efficiently compute collision response for all contacts in a similar manner. We use impulse-based dynamics computations along with analytical constraint solving techniques. To improve the runtime performance, we derive a new formulation for

the *hybrid-body Jacobian*, which exploits the structure of the hybrid tree to reduce the overall computational complexity. Our algorithm has a sublinear runtime complexity in the number of articulated body DOFs and can be used to efficiently simulate the dynamics of snake-like or deformable robots with a very high number of DOFs.

We demonstrate the application of our planner on several highly articulated robots consisting of 300 to 2500 single degree of freedom joints in a serial linkage. Each local sampling step using adaptive dynamics for these robots takes few milliseconds on average. We have observed up to one order of magnitude performance improvement using our approach as compared to sampling with full dynamics.

These components have been integrated into a motion planning approach for highly articulated serial linkages. Some properties of the resulting planner include:

- **Physically-based**: We take into account forward dynamics of articulated joints during motion planning, in addition to the geometric constraints including 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.

- **High DOF robots**: Our algorithm can simulate the forward dynamics and plan the path for a robot with very high number of DOFs (in the thousands) in nearly real time, using a progressive refinement framework.

To highlight the behavior and performance of the articulated body planner, we apply it to a variety of situations using snake-like robots for search and rescue amidst debris, industrial pipe inspection for leakages, and finally as another way to solve the liver chemoembolization problem.

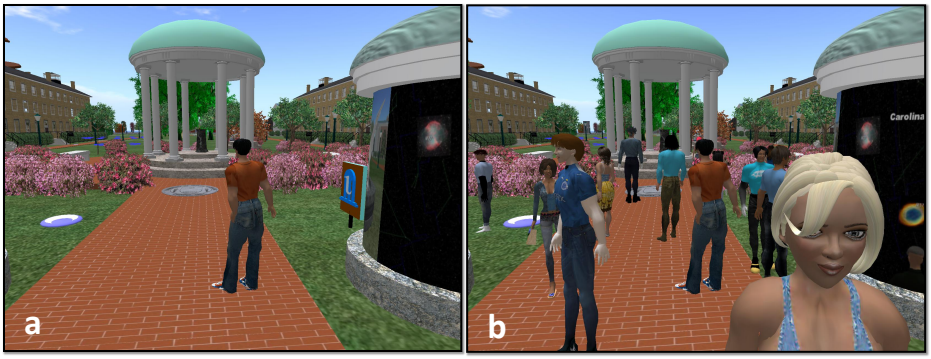

Figure 1.9: Impact of multiple agents: The presence of multiple agents can greatly impact the realism of a scene or scenario. On the left, we see a scene with only the user's avatar in an online virtual world. On the right, the avatar is joined by intelligent agents which can autonomously navigate the virtual world.

Numerous agents in dynamic environments

In the previous results, our work focused on single robots in static environments. We generalize the approach to work for multiple robots moving simultaneously through an environment. Briefly, we decouple the approach such that each agent considers all the other agents to be dynamics obstacles. Therefore, the overall approach works in environments where the motion of obstacles is not known a priori. With this decoupling, there are three ways in which physics-based sampling could be applied: via local navigation or local avoidance, via global navigation, or an integration of local and global ideas.

As in the case of a single deformable or articulated body, we could establish a static path for each agent, and have the agent follow it. Then, while the agent is following that path, various constraints would allow it to avoid other agents or other obstacles. Since this this only accounts for other bodies near the agent, these constraints can be considered part of local navigation.

Our local planning model considers coordination as a soft constraint, including general repulsive or attractive behaviors, grouping, formations, local navigation, collision avoidance, and distance constraints. While much of the related work for multiple robots assumes point, point-like, circular or cylindrical robot primitives, this approach can

handle arbitrary robot and obstacle shapes, including non-convex polyhedra. Instead of a separate coordination phase, our approach interleaves coordination with motion generation through the use of coordination constraints.

Local navigation uses social potential fields (SPF) as an additional constraint on the simulation. Social potential fields are a variation of the potential field methods whose resulting motion appears to exhibit various social behaviors. We generalize and reformulate social potential fields to be integrated with the physics-based sampling framework and also to be applicable to a more general robot primitive. The social potentials will act as a constraint on the system, allowing the robot to change its current trajectory and move toward or away from other robots or obstacles. This framework supports additional constraints, such as creating and maintaining group formations and other behaviors. To reduce the effects of local minima, an excitation factor has also been included. When a robots velocity is low and it is not at its goal, excitation will increase the strength of the forces leading toward the goal.

The local navigation approach works well in several situations, particularly when the connectivity of the environment changes little and when all the other agents are following similar motion models. However, other obstacles in the scene could also move, which could eventually invalidate our static path. In this case, we need to be able to find a new path though the environment. We have developed a novel roadmap scheme based on physics-based sampling which works well in a wide variety of situations. A second approach could then be to have the agents follow this roadmap in a prescribed manner. Since all the motion is controlled through this roadmap, we refer to it as part of global navigation.

We present a novel, adaptive motion planning data structure called Reactive Deforming Roadmaps (RDR), for motion planning of multiple robots in dynamic environments. Each link on a RDR can adapt, or deform, based on the motion of robots and obstacles the environment. As such, it falls into the class of path modification algo-

rithms. Essentially, we are computing and updating an adaptive roadmap of deforming links. Our overall representation provides the capability to perform path maintenance at local and global levels, since moving obstacles can alter or invalidate portions of the roadmap. Each link reacts to obstacle motion to remain in $\mathcal{F}$ via link modification, and the roadmap structure maintains connectivity or removes invalid links through roadmap maintenance steps. By using a RDR, less connectivity information is lost since it becomes more difficult to invalidate a link as compared to links which cannot react to obstacle motion. To improve performance, an adaptive sampling scheme ensures that the areas of the roadmap closest to obstacles receive the most computational resources.

By integrating our local and global navigation schemes, we show that our physics-based sampling approach for multiple agents can handle thousands of agents in several situations. We apply it to several realistic situations, as well as to online virtual worlds.

We have specialized integrated social potential fields and RDRs for the task of simulating numerous heterogeneous humanoid agents or crowds in both specified environment and popular online virtual worlds. On top of RDR, we define link bands to represent the portion of freespace nearby each link in the roadmap. These bands are used to couple RDR with the local agent navigation. They help with both agent clustering and resolving collisions among multiple agents. Due to lazy and incremental updates to the roadmap and efficient computation of guiding-path forces using link bands, our approach can scale to hundreds or thousands of individual agents. In practice, we can perform realtime global navigation of many independent agents in complex, changing environments, with generic obstacles and no restrictions on agent motion. Within a link band, a social potential field based on a pedestrian dynamics model is used to guide agent interaction and general motion.

To summarize, our planner for multiple robots has the following properties:

- **Adapting roadmaps**: We propose novel approaches to maintain and update a roadmap in a dynamic environment. Briefly, we can attach a physical state to

each configuration on a roadmap. Then, using the simulation core, we can update it much in the same way that we update the robots. By placing bounds on the roadmap, we can allow it to bend and deform for smaller motions, but also to break and repair itself for larger motions. This blend allows for both local and global updates in an efficient manner that fits naturally within our existing framework. Furthermore, the configurations adapt in a physically realistic way which helps ensure that the final path is smooth and realistic.

- **Local avoidance and coordination**: We generalize and integrate the pieces of our physics-based sampling approach using Generalized Social Potential Fields (GSPF). These have the advantage that arbitrary robots can be used and their underlying motion model helps to ensure that the resulting motion is smooth and realistic. The model is based on ideas from pedestrian dynamics (the study of how humans move with and among each other), and thus the agents inherit behaviors which are both repulsive and attractive.

- **Efficiency**: We develop an adaptive sampling scheme defined over the links of the roadmap to help minimize unnecessary computation and link modification. This helps to lower the overhead of maintaining the RDR. Also, we exploit grouping and clustering of agents to help accelerate distance queries. These optimizations combined with an efficient simulation at the core of the physics-based sampling allows for the planning of thousands of agents in a variety of situations.

We demonstrate our approach in two types of scenarios. For crowd and social simulation modeling, we apply the physics-based sampling approach to complex indoor and outdoor scenarios, including a city scene consisting of 2,000 pedestrians with 50 moving cars and an exhibition hall with 511 stationary booths and 1,000 individual agents on foot and avoiding each other. In order to demonstrate the benefit of a global navigation approach, we also place up to 1,000 agents in a maze environment. Our implementation

is able to perform motion simulation of independent agents for these highly challenging scenes in a fraction of a second per frame.

For additional real-world applications, we present an approach for adding numerous autonomous virtual agents into online virtual worlds with a centralized server network topology. We combine techniques from physics-based sampling to control the motion of each agent based on local and global navigation with a general networking model. Briefly, a variation on physics-based sampling determines how agents coordinate with nearby objects while a separate global navigation module allows agents to select goals in order to complete their current task or reach the goal position. To reduce the impact of network latency, we make assumptions about the linearity of motion in a short time interval and augment the local navigation model with a velocity-bias social force. Since global planning is agent-specific, our algorithm that runs on the server can be used to balance the computational load between client and server. We have implemented our approach into the Second Life virtual world using a motion controller built with LibOpenMultiverse (LibOpenMV, formerly LibSecondlife). We highlight our results by simulating up to 18 agents over two different client computers of varying computational power, in different geographic locations which are connected via different speed connections (See Fig. 1.9).

1.6 Organization

The remainder of this dissertation is organized as follows. In **Chapter 2**, we survey related work in the areas of motion planning with an emphasis on high-DOF planning, dynamic environments, and for multiple robots. **Chapter 3** presents our framework for physics-based sampling and the remaining chapters show the framework applied to those situations. In particular, **Chapter 4** introduces a novel framework for physics-based motion planning with deformable robots. **Chapter 5** extends these ideas onto

more rigid and highly-articulated agents. **Chapter 6** presents adaptations to handling numerous agents while also considering dynamic environments. Finally, **Chapter 7** discusses future directions and concludes this work.

Chapter 2

Related Work

There is a substantial amount of work in the area of motion planning. In this section, we describe work that is related to the problems mentioned here. This includes planning for different types of complex robots, planning for multiple robots, and efficient replanning methods. For a broader treatment of the field, we refer our readers to [Lat99, CLH+05, LaV06b].

2.1 Randomized Motion Planning

Exact, or complete, solutions to the general, single-robot, motion planning problem have a computational complexity that is exponential in the problem dimensionality [Can88b]. As previously mentioned, to deal with this complexity, many algorithms use randomization to exchange completeness for practical performance. Randomized methods, such as the Probabilistic Roadmap Method (PRM) by Kavraki et. al and its variants, create a graph in the configuration space [KSLO96]. This graph can then be rapidly queried at run-time to provide answers to planning problems. Rapidly-exploring Random Trees (RRT), proposed by Kuffner and Lavalle, work by iteratively extending a tree toward goals and unexplored regions in the $\mathcal{C}$ [KL00]. These, and most other randomized approaches, offer a weaker form of completeness called probabilistic completeness. Briefly, this means that given enough time to adequately sample the

$\mathcal{C}$, they will find a path if one exists. Furthermore, many of these methods and their variants are relatively simple to implement.

There are many considerations with these approach. First of all, solutions are susceptible to the narrow passage problem and their effectiveness is dependent upon the sampling method used to generate the milestones. Many algorithms have been proposed to improve the performance of PRM planners in terms of developing better sampling strategies and handling narrow passages [HKL+98, ABD+98, WAS99b, FGLM01].

Performance of randomized planners is dependent on determining whether or not milestones and links are collision-free, via collision detection or distance queries.

2.1.1 Kinodynamic Planning

While many deterministic solutions have been proposed for kinodynamic motion planning, these approaches typically do not scale well to high DOF robots [DXCR93] since they must plan over the $\mathcal{S}$-space rather than the $\mathcal{C}$. Many authors have also proposed randomized kinodynamic planners based on PRMs [HKLR02] and RRTs [LK99]. Like their $\mathcal{C}$-space analogues, the performance depends upon the dimensionality of the state space. Ladd and Kavraki pose a tree-based kinodynamic solution where the path segments are the samples rather than states [LK05]. Storing state can be generalized to include the state of the entire simulation, allowing for planning with arbitrary changes in both the environment and obstacles, such as deformations [RLA06b]

2.1.2 Sampling Strategies for Motion Planning

The problem of sample generation for various planning strategies has received a considerable amount of attention. Several extensions have been proposed for PRMs. For instance, visibility information can be used to generate and store fewer samples [SLN00]: medial-axis based sampling [PHLM00, GHK99, Wil99] and sampling near the obstacle boundaries [ABD+98] can help in dealing with narrow passages. Gaussian weighting can

aid to place samples where they would reveal most information about an environment [BB05].

With respect to RRTs, the standard algorithm already performs extra biasing [KL00] to try to build more optimal trees [US03]. Better directions of growth based on Voronoi diagrams of the C-space can be generated [LL05], if the Voronoi diagram of the configuration space with the samples is available. Dynamic domains are used to improve expansions in RRT [YJSL05]. Flood fill strategies can provide a deterministic sampling strategy for tree-based planners . Forests (multiple trees) have been used as a good trade-off between RRTs and PRMs, and also as a way to maintain configuration spaces with dynamic obstacles [PBC+05, GKX07, ZKB07].

2.2 Deformable Robots

2.2.1 Motion Planning for Deformable Robots

Most of the literature in robot motion planning has focused on robots with one or more rigid segment or link. Some of the earlier work on deformable robots included specialized algorithms for bending pipes [SSL96], cables [NK97] and metal sheets [NM96]. Holleman et al. [HKW98] and Lamiraux et al. [LK01] presented a probabilistic planner capable of finding paths for a flexible surface patch by modeling the patch as a low degree Bèzier patch and used an approximate energy function to model deformation of the part. Guibas et al. [GHK99] described a probabilistic algorithm for a surface patch by sampling the medial axis of the workspace. Anshelevich et al. [AOLK00] presented a path planning algorithm for simple volumes such as pipes and cables by using a mass-spring representation. Bayazit et al. [BLA02a] described a two-stage approach that initially computes an approximate path and then refines the path by applying geometric-based free-form deformation to the robot. More recently, Rodriguez et al. [RLA06b] presented an RRT-based algorithm for a deformable robot in a deformable

environment.

2.2.2 Collision Detection between Deformable Models

While collision and overlap detection has been actively researched for many years, much of that has focused on rigid bodies. In this section, we give a brief overview of related work in collision detection between deformable models. For general coverage of the problem of collision detection, we refer the interested reader to some recent surveys [Eri04, LM03, TKH+05].

Many of the most commonly used algorithms for collision detection are based on bounding volume hierarchies (BVHs). These hierarchies cull away portions of a model that are not in close proximity. Examples of such hierarchies include sphere-trees, AABB-trees, OBB-trees, k-DOP trees, etc. [Eri04] and they are typically computed during preprocessing. Recently, algorithms have been proposed to lower the overhead of updating the hierarchy during every step of deformable simulation [JP04, LAM01, vdB97]. However, the cost of updating the hierarchy for complex models can be relatively high. Moreover, in many close proximity configurations, these hierarchies may be unable to perform significant culling which results in a high number of exact collision tests between the primitives.

Many collision and proximity computation algorithms exploit the computational capabilities of graphics processors (GPUs) [BW02, GRLM03, HTG03, KP03]. Recent work also includes methods for self-collisions of deformable models [GKJ+05]. Most of these algorithms involve no preprocessing, therefore applying to both rigid and deformable models. The GPU-based algorithms perform image-space computations and use the computation power of rasterization hardware to check for overlaps. However, a major issue with current GPU-based algorithms is limited accuracy due to image-space resolution, possibly resulting in missed collisions between small triangles due to sampling errors.

2.3 Articulated Robots

Articulated body or multi-body motion planning has seen considerable attention from the research community for many years. In many cases, much of the related work reduces to standard randomized approaches (PRMs, RRTs) but with sampling schemes or local planning methods which are optimized for various articulated structures.

2.3.1 Articulated Body Motion Planning

In general, much of the existing work in motion planning can be applied to articulated chains. In fact, this is a special case of the more general planning problem applied to branched or closed loop articulated bodies.

In the majority of these cases, various sampling schemes or improved local planning are used to improve the efficiency and effectiveness of the planning approach. This includes optimizations for objects that are similar to articulated chains, such as pipes, cables, ropes, and flexible wires [SSL96, NK97, AOLK00, LK04, MK04, MK05, GSLM05a]. Another approach is to determine the principal components using statistical methods to reduce the dimensionality of such a high DoF configuration space [TPK03]. Also, Barraquand et. al used potential fields with random walks for high-DOF articulated robots [Bar91b]. However, many of these algorithms do not consider the kinematic or dynamics constraints related to these types of structures.

Several algorithms take into account kinematics information to aid in generating samples for PRM-based planners [LYK99, HA00, CSL02]. However, these approaches are targeted toward closed loop linkages and do not take contact responses such as friction into account.

2.3.2 Articulated Body Dynamics

Multibody dynamics has been extensively studied in the literature. We refer the readers to a recent survey [FO00]. Theoretically optimal, linear-time forward dynamics algorithms [Fea87, Hol80, Bar96, RK97] that depend on a recursive formulation of motion equations have been proposed. Reformulations of the motion equations have also been developed using new notations and formulations, including the spatial notation [Fea99a, Fea99b], the spatial operator algebra [RJKD91], and Lie-Group formulations [MM03]. More recently, parallel algorithms have also been introduced to compute the forward dynamics of articulated bodies using multiple processors [Fea99a, Fea99b, YN02].

Our work is based on the "Adaptive Dynamics" (AD) algorithm proposed by Redon et al. [RGL05]. This approach enables automatic simplification of articulated body dynamics. Using well-defined motion metrics, the algorithm can determine which joints should be simulated in order to minimize the computation errors while approximating the overall motion of an articulated robot. However, AD algorithm cannot handle collisions and is limited to freely moving robot arms with no contacts or collisions. In contrast, our approach introduces an adaptive contact handling technique that is tightly coupled with the model representation of AD and enables collision response computation for highly articulated bodies in sub-linear time.

2.3.3 Collision Detection and Response

Similar to the deformable bodies, many algorithms based on bounding volume hierarchies have been proposed for collision detection between articulated models and the rest of the environment [LM03]. These hierarchies are updated at each discrete time step. Moreover, efficient algorithms have been proposed to perform continuous collision detection between two discrete time instances [Xav97, Xav02, RKLM04]. The continuous algorithms model the trajectory of each body as a swept volume and check each swept volume for overlap with the environment.

Several algorithms have been presented to simulate colliding rigid bodies. These include impulse-based dynamics [Mir96], penalty-based methods, and constraint-based dynamics: e.g. Gauss' principle of least constraints [RKC02] or the linear complementarity problem (LCP) formulation [Bar91a, TTP01]. Post-stabilization techniques for rigid body simulation with contact and constraints have also been proposed [CP03].

Mirtich [Mir96] and Kokkevis [Kok04] have described linear-time methods to handle collisions based on Featherstone's Articulated Body Method (ABM). Weinstein et al. [WTF05] present a linear time algorithm to simulate articulated rigid bodies that undergo frequent and unpredictable contacts and collisions.

2.4 Multiple Robots

The multiple robot extension is unique from other planning problems in that typically the DoFs between different robots can be decoupled. This leads to two different classifications of planners for multiple robots: *centralized* and *decoupled*. There is extensive literature on path planning for multiple agents in robot motion planning and virtual environments [LaV06a].

Several works have realized the various benefits of each planning model, and combined them into hybrid approaches. For instance, [CRL03] combines a centralized planner for intra-group coordination while using a decoupled planner for inter-group planning. Both types of planners have been applied to a number of situations such as flocking and shepherding [BLA02d, LRMA05].

2.4.1 Centralized Planners

The centralized approach to planning for multiple robots considers the robots as a single system, rather than independent entities. In this case, much of the work previously mentioned can be directly applied [BLA02b, FTT99, KO04, LD04, PLT05, SKG05,

SAC+07].

However, direct application of these ideas may also lead to complications. If robots are in close proximity to each other, this may result in narrow passages in the $\mathcal{C}$-space. For instance, it may be difficult to generate a random sample such that a robot in the center of a cluster moves in unison with the surrounding robots. Li et al. identify and accommodate for groups in planning is one method aid in this problem and improve performance [LC03b]. Aronov et al gave centralized solutions for specific cases, such as for pairs or triples of roots in a low density workspace [AdBvds+99]. Several other algorithms have been proposed to extend the roadmap-based methods to dynamic environments and multiple agents [GO07, LG07, PLT05, ZKB07]. Again, in many of these cases, they have only been applied to relatively simple environments composed of a few robots and restricted obstacles. These approaches may not scale well to environments with a large number of independent agents.

The primary advantage of centralized methods is in their theoretical completeness. Additional tasks such as coordination are not necessary since it is automatically solved by the planner. However, again this comes at the price of complexity.

2.4.2 Decoupled Planners

In the decoupled approach, motion planning for multiple robots usually takes two stages. In the first stage, a path is found for each robot without consideration of the paths of the other robots. In the next step, paths are adjusted to try to allow for robot-robot interactions, usually through some sort of coordination, such as *velocity tuning*. This is often referred to as fixed-path coordination and is essentially adjusting velocities along the paths such that no collisions occur. It is often done by exploring a *coordination space* which represents the parameters along each robot path. In this approach, completeness is lost since it is possible that no solution can be found in coordination space of the predetermined paths even though a path may exist in $\mathcal{C}$-space.

Prioritized planning for multiple robots is a variant of the decoupled approach. The idea is that each robot is assigned a priority ahead of time. In the order determined by priority, each robot plans a path while treating the robots whose paths have already been determined as dynamic obstacles [ELP86]. Choice of the priorities can have a large impact on the performance of the algorithm [vdBO05]. Some planners also search through a space of prioritizations [BBT02].

Alternatively, other decoupled schemes such as coordination graphs [LGP05], incremental planning [SI06], integration with replanning [BTK07], and velocity obstacles [vdBLM08] can help to ensure that no collisions occur along the paths even in cases with complex dynamics.

Another class of decouple planners allows agents to move independently and in the reactive style of potential fields. [Kha85, War90, GC02, BBGN07]. They can handle large dynamic environments, but suffer from 'local-minima' problems and may not be able to find a collision-free path when one exists [LaV06a]. Often these methods do not give any kind of guarantees on their behavior. Other route planning algorithms are based on path or roadmap modification, which allow a specified path for an agent to move or deform based upon obstacle motion. These methods include Elastic Bands [QK93] and Elastic Roadmaps [YB06]. Social potential fields alter the navigation function such that the resulting motion reflects social behaviors [RW99]. Our work is most closely related to this approach. Our approach bears some close resemblance to these techniques, but RDR is lazily updated to deal with dynamic obstacles and we present many new techniques to plan collision-free paths for multiple agents simultaneously using RDR.

2.4.3 Crowd Dynamics and Human Agents Simulation

Many different approaches have been proposed for modeling movement and simulation of multiple human agents or crowds or individual pedestrians. [ALA+01, SS01, ST05, TOCD06, Rey06, vdBPS+08]. They can be classified based on specificity, problem

decomposition (discrete vs continuous), stochastic vs deterministic, etc.

Discrete methods

Discrete methods rely on a sampling of the environment or of the agents. Some common approaches include:

Agent-based methods: These are based on seminal work of Reynolds [Rey87] and can generate fast, simple local rules that can create visually plausible flocking behavior. Numerous extensions have been proposed to account for social forces [CBS+05], psychological models [POSB05], directional preferences [SGC04], sociological factors [MT97], tactical behaviors [Ree03], etc. Velocity obstacles have been utilized for improving local avoidance in dense crowds [vdBPS+08].

Different techniques for collision avoidance have been developed based on grid-based rules [LMM03] and behavior models [TT94]. Most agent-based techniques use local collision avoidance techniques and cannot give any guarantees on the correctness of global behaviors. Although, in many situations these resulting motion matches how people navigate (e.g. people packing dense in a narrow corridor).

Cellular Automata methods: These methods model the motion of multiple agents by solving a cellular automaton. The evolution of the cellular automata at next time step is governed by static and dynamic fields [HLB+00]. While these algorithms can capture emergent phenomena, they are not physically based.

Particle Dynamics: Computing physical forces on each agent is similar to N-body particle system [SS01, HBW03]. Sugiyama et al. [SNH01] presented a 2D optimal velocity (OV) model that generalizes the 1D OV model used for traffic flow. Our formulation is built on some of these ideas and we elaborate them in Section 6.5.2.

Continuous Methods

The flow of crowds or multiple agents can be formulated as fluid flows. At low densities crowd flow is like gases, at moderate densities it resembles fluid flow, and at high densities crowd has been compared to granular flow [HBJW05]. Most recently, a novel approach for crowd simulation based on continuum dynamics has been proposed by Treuille et al. [TCP06]. We compare our approach with these methods in Section 7.

2.5 Dynamic Obstacles

In many real-world situations a robot must operate in an environment with moving or otherwise changing obstacles. In some ways, the dynamic obstacle problem is related to the multiple robot problem in the case where each agent views the other agents as dynamic obstacles. Dynamic elements greatly increase the difficulty of motion planning. In fact, motion planning for a single disc with bounded velocity among rotating obstacles is PSPACE-hard [RS85]. However, there have been many attempts to provide practical methods to cope with changing environments. For example, Stentz et al proposed the D* deterministic planning algorithm to repair previous solutions instead of re-planning from scratch [Ste95, KL02].

There have been two types of approaches for adapting randomized planners to dynamic environments. The first type includes both PRMs and RRTs that reuse previously computed information to aid in finding a new path [LH00, LS02, KM04, JS04, FKS06].

The Dynamic Rapidly-exploring Random Tree (DRRT) proposed by Ferguson et al. [FKS06] trims away subtrees of nodes found to be invalidated by an obstacle and causes regrowth using the trimmed tree if the solution path is invalidated.

The second type of approach integrates obstacle motion directly into the planning process. Some variations plan directly in a $\mathcal{C}$-space augmented with a *time* parameter. For example, the method of [Fra93] generates a roadmap in this space. Other approaches

additionally model differential constraints placed on the robot [HKLR02]. From the dynamics information, motion bounds can be computed from trajectory information to remove the chance of being in a state where is a collision is definitely going to occur [PF05].

Rather than changing the roadmap, other work for dynamic environments has focused on adjusting of modifying the path. Potential field planners use gradient descent to move toward a goal, at a potential sink [Kha85]. As obstacles move, the potential field changes which will be immediately seen by a robot. Since these global operations are more expensive, other approaches instead adjust the path itself rather than entirely recompute it. This allows a robot to automatically adjust to changes without having to do a great deal of work.

Building on these ideas, Quinlan and Khatib first proposed elastic bands, which represented the path as a sequence of overlapping *bubbles* in the free-configuration space. Internal forces kept bubbles from separating, while external forces from obstacles encouraged them to move towards the free-space [QK93]. By maintaining a set of homotopic paths in similar free-space bubbles in the workspace a variety of paths can be selected quickly at runtime [BK00]. Elastic strips simplified the problem by mapping portions of articulated bodies to the workspace [BK99]. Elastic roadmaps exploited restrictions on task-specific goals and workspace visibility to quickly update and maintain a roadmap [YB06]. Additionally, Lamiraux et. al extend the ideas to include nonholonomic paths [LBL04].

Chapter 3

Physics-based Sampling

3.1 Introduction

Motion planning research has made great strides in providing solutions in a wide range of application areas for wide ranges of environments, for a large variety of classes of robots, and with numerous types of constraints and tasks. However, the majority of this work still fails to scale well as the complexity of the scenes and robots increase or as the search space becomes overly tightened. As mentioned, the goal of this work is to integrate motion planning methodologies and concepts with motion guidelines from physics simulations in order to create a planner which not only scales well in the number of agents and types of environments but is applicable to a variety of situations and types of robots.

The main observation in creating such a method is to note that constrained simulation in the workspace is related to a directed search in the configuration (or state) space. Rather than completely mapping out the configuration space, the constrained simulation is biased or sometimes even forced to move toward or through what are essentially narrow passages in the configuration space or state space. Furthermore, the bias can often be formulated as a set of forces defined in the workspace. Thus, unlike the majority of planning methods mentioned thus far constrained dynamical simulation

is intended to work well in environments with extremely large numbers of degrees of freedom and in constrained passages.

It is these properties which make it a good candidate for providing practical solutions to otherwise challenging planning situations. However, the difficultly in using simulation for planning is that of control. Simulation is intended to be predictive and undirected whereas planning requires control and spatial reasoning (in high dimensional spaces).

We develop a novel framework called physics-based sampling which couples spatial reasoning and control in a simulation framework. The main idea is to use the constrained space to our advantage. Rather than allowing the robot or agent to move arbitrarily, it is necessary to define a set of constraints which will guide it toward its goal.

In the remainder of this chapter, we describe and develop the fundamental ideas of physics-based sampling. First, we introduce the underlying constrained simulation. Then, physics-based sampling is built as a series of constraints and a method for generating state or configuration space space samples to serve as a path. Finally, we discuss some of its general properties when compared to different types of planning approaches.

3.2 Constrained simulation

Constrained simulation is a core computational component in physics-based sampling. It is responsible for maintaining any constraints required for motion planning and finally determining the next sample given a set inputs. This section describes the core ideas behind physical simulation and constraints defined on top of it.

3.2.1 Physically-based Simulation and Modeling

At the heart of a simulation is a set of motion equations which define how an object moves based on its physical properties and external forces acting upon those bodies. In basic *unconstrained* simulation, the bodies solely follow these laws and are not concerned

with external interactions such as joints, collisions, or other geometric constraints. In this section, we first give a brief overview of physically-based simulation for particles and rigid bodies, and then provide and describe constrained dynamics as they are related to this work. For more information, we refer the interested reader to [WB97].

Particle simulation

In its most basic form, the Newton-Euler laws of motion describe movement and changes in movement of particles. Consider a particle $\mathbf{p}$ to be a point in $\mathbb{R}^2$ or $\mathbb{R}^3$ with a mass m to which external forces can be applied. We define the time-varying position of $\mathbf{p}$ as $\mathbf{x}(t)$ at some time t. From the Newton-Euler laws of motion, we get that the motion of this particle is described by $\mathbf{f} = m\mathbf{a} = m\ddot{\mathbf{x}}$, where $\mathbf{f}$ is the net external force. Using the fact that the velocity of $\mathbf{p}$ is $\mathbf{v} = \dot{x}$, we get a pair of coupled first order differential equations,

$$\begin{pmatrix} \dot{x} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \frac{\mathbf{f}}{m} \end{pmatrix}$$

Using position and velocity, we describe the *state* of the particle at time t as $\mathbf{q} = \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{pmatrix}$. Finally, to solve for $\mathbf{q}(t)$ at time t, we determine the total force $\mathbf{f}$ acting on $\mathbf{p}$ and solve our system of differential equations given its current state. Additional forces such as gravity, wind, or air resistance can be easily applied to the particle under this formulation. For a system of many particles, we can store the state of each particle and solve for them individually. Conceptually, one can consider the task of solving this system as a state transition function.

Rigid body simulation

While particles are useful conceptually, it is an insufficient representation for a realistic robot, particularly since a robot's geometry is necessary part of the planning problem.

Robots are often described by either a single rigid body or a collection of rigid bodies. As in the particle case, we can consider the state $\mathbf{q}$ of a single rigid body to include both a position and a velocity. However, unlike the particle case, both the position and velocity each now contain an angular component. Let $\mathbf{R}(t)$ and $\omega(t)$ represent the orientation and angular velocity of our body, respectively. Furthermore, since the body is not a point in space, the *inertial tensor* $\mathbf{I}$ describes its reaction or resistance to torques τ.

A rotational analog to our motion equations can be formulated by the following system of differential equations:

$$\begin{pmatrix} \dot{x} \\ \dot{\mathbf{R}} \\ \dot{\mathbf{v}} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \omega\mathbf{R} \\ \frac{\mathbf{f}}{m} \\ \tau\mathbf{I}^{-1} \end{pmatrix}$$

For a more complete derivation of this equation, we refer readers to [WB97].

Similar to particle simulation, to solve for the position and velocity for a rigid body at time t, we first determine the total force and torque acting upon the body and then integrate to solve the differential equations.

Articulated and deformable body simulation

Articulated and deformable bodies in physical simulations add flexibility in terms of what types of bodies (or robots) can be represented. As in the previous examples, both articulated bodies and deformable bodies can be described by a state $\mathbf{q}$ which is updated based on a set of motion equations. Since deformable bodies and articulated bodies (particularly with many links or a high branching factor) are a focus of this work, their physical descriptions and their motion equations can be found in Chapters 4 and 5, respectively.

Numerical integration

The prior sections define the motion of a body but omits a key step for simulation, how to computationally solve for the body's motion over time. In general, this requires solving the motion equation ordinary differential equations (ODEs). In some cases, a closed solution can be found given some initial values. A closed form solution could be sampled at any resolution to get positions and velocities along the motion path. For physical systems, closed form solutions are often to determine, especially when certain unpredictable external factors such as collisions are involved. Instead, we can use *numerical integration* to approximate the solution over some small change in time. By iteratively applying numerical integration for a discrete time step, we determine a sequence of states over time for our bodies.

One of the simplest integration techniques available is Euler's Method (see Appendix A.1). Let the current state of a body be $\mathbf{q}(t)$ and its derivative be $\dot{\mathbf{q}}(t)$. Using Euler's Method, we can advance time by a step h using the following formula:

$$\mathbf{q}(t+h) = \mathbf{q}(t) + h\dot{\mathbf{q}}(t)$$

Conceptually, Euler's computes a piecewise approximation to the true state solution by sampling the derivative at the current state as a guideline for which direction to move in. Then, the state is updated by proceeding in that direction for a small portion of time.

While simple and quick to compute, Euler's is fairly inaccurate. A smaller time-step can be used to improve accuracy, but this requires more computation to evolve the system by some amount of time. There are numerous other integration methods available, each often varying degree of accuracy, performance, simplicity and stability. One of the keys to quick and realistic simulation is appropriate choice of integration method. In our implementations, we use a variation of "Verlet" integration (See Appendix A.2).

3.2.2 Constraint-driven simulation

In the previously unrestricted simulation environments, particles or rigid bodies were allowed to move freely through the environment using simple rules for motion equations. In general, the idea of constraint-driven or constrained simulation is that we want to be able to do more than apply external forces, we want to be able to actually restrict the way particles or other bodies are allowed to move in a scene. This is an important concept in general, as it paves the way for more compelling and efficient simulations. Constraints allow us to connect two bodies via hinge, require that certain bodies stay a fixed distance apart, ensure that no pair of bodies are overlapping or interesting, and many more. In dynamic simulation, there are effectively two ways to constraint the motion of body. First, the motion equations themselves can be derived to include these constraints. This is often the case hinges and joints on articulated bodies or wheeled vehicles. Otherwise, the total force acting on a body can be constrained such that the force component that causes a constraint to become un-satisfied is removed. This is often implemented a constraint force, or a force which eliminates the portion of the force that causes a constraint to be invalidated. Even more generically, the constraint force could also be viewed as an influence on the motion. While more firm constraints eliminate portions of the total force, other constraints could bias the motion by contributing to the total force. The idea of constraint forces is heavily exploited throughout this work since it enables a way to both impose restrictions and influence the final motion.

Constraint forces

Constraints are firm requirements that a particle or body must obey during a simulation and can take many forms. A large subset of constraints on the motion itself take the form of constraint forces and are solved as such. Constraints could include fixing the body to a path or certain types of motions and ensuring that collisions, contacts, and their related response forces are adequately considered. Joints could fall into this

category although common practice is to incorporate articulated motion directly into the motion equations. Certain constraints, such as those defined over the geometry (e.g. non-penetration constraints), could also be enforced by geometric transformations. Alternatively, a net force could be found to effectively cause the same transformation. In practice, both methods work well in different situations.

For example of constraints and constraint forces, consider a "point-distance" constraint between a pair of points $\mathbf{x}$ and $\mathbf{y}$. Mathematically, this constraint is defined as $\mathcal{C}_{dist}(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}|| - d$ for some specified distance d. During simulation, if $\mathcal{C}_{dist} \neq 0$, then let δ_d be the difference between d and the actual distance. We can move each body a distance of $\frac{\delta_d}{2}$ toward each other, or otherwise alter the force component of each body in the direction of the other body to have the same effect, to enforce the constraint. Many geometric constraints can be formulated in this manner, several of which will receive more attention in the subsequent chapters.

The complete derivation of constraint forces is more involved and much of it is beyond the scope of this work. For simplicity we will derive the underlying ideas for constraints on a single particle [WB97]. This can be generalized for a system of constraints as well as for general rigid bodies. For a more complete treatment, we refer readers to [WB97, Kok04].

Let us define a constraint function for a particle $\mathcal{C}(\mathbf{x}(t))$ as a scalar implicit function, i.e. it describes a property and is satisfied when $\mathcal{C}(\mathbf{x}(t)) = 0$. To satisfy a constraint we want to apply a constraint force in the direction of the gradient of the constraint. Thus, the resulting force will be of $\mathbf{f_c} = \lambda\dot{\mathcal{C}}$, where λ is an unknown scaling factor which must be determined.

If we assume that constraint forces are passive, that energy is neither added or lost in the system ($\ddot{\mathcal{C}} = 0$), then:

$$\dot{\mathcal{C}} = \mathbf{N}\dot{\mathbf{x}}$$

$$\Rightarrow \ddot{\mathcal{C}} = \frac{\partial}{\partial t}[\mathbf{N}\dot{\mathbf{x}}]$$

$$\Rightarrow \ddot{\mathcal{C}} = \dot{\mathbf{N}}\dot{\mathbf{x}} + \mathbf{N}\ddot{\mathbf{x}}$$

$$\Rightarrow \ddot{\mathcal{C}} = \dot{\mathbf{N}}\dot{\mathbf{x}} + \mathbf{N}\frac{\dot{\mathbf{f}} + \mathbf{f_c}}{m}$$

$$\Rightarrow \ddot{\mathcal{C}} = \dot{\mathbf{N}}\dot{\mathbf{x}} + \mathbf{N}\frac{\dot{\mathbf{f}} + \lambda\mathbf{N}}{m}$$

where $\mathbf{N} = \frac{\partial C}{\partial \mathbf{x}}$ and $\dot{\mathbf{N}} = \frac{\partial^2 C}{\partial \mathbf{x}\partial t}$. Setting $\ddot{\mathcal{C}} = 0$ and solving for λ we get:

$$\lambda = -(\frac{m\dot{\mathbf{N}}\dot{\mathbf{x}}}{\mathbf{N}\mathbf{N}} + \frac{\mathbf{N}\mathbf{f}}{\mathbf{N}\mathbf{N}})$$

which gives us our constraint force given the partial derivatives of our constraints. Thus, for a motion constraint, the first task is to compute its derivatives. Then, they can be put into this "template" to be applied to the body. Conceptually, this can be thought of as eliminating the portion of the external force that would allow a constraint to be broken. This works well in general, but has a few drawbacks. It is possible that the constraint may not be satisfied in the initial state or numerical drift can cause the constraint to no longer be held. In both cases, a common approach is to include a feedback term which counteracts these situations.

3.3 Physics-based sampling

As stated in the thesis statement, one of the primary goals of this work is to develop a planning approach that couples traditional motion planning and constrained simulation. We propose physics-based sampling as one solution on how to accomplish this task.

Conceptually, motion planning based on physics-based sampling reformulates the motion planning problem as a constrained dynamical system coupled with a set of constraints. This idea, and similarly the main idea in "constraint-based motion planning" [GL02], exploit the relationship and resemblance between motion planning and the boundary value problem, where the boundary conditions are the initial and goal configurations of the robot. Physics-based sampling is used to compute the intermediate states that link the start and goal configurations by traversing a path while satisfying constraints imposed on the dynamical system.

The key differences between this approach and existing geometric techniques is that motion planning is no longer treated as a purely geometric problem. With this framework, we can automatically incorporate the mechanical and physical properties of the robots and obstacles, in addition to their geometric description. This core computational task of this framework is based on constrained dynamics [WB97], where both standard simulation and planning constraints are enforced by virtual forces imposed on the system. By combining constraint forces chosen for planning into a simulation framework, the constrained dynamical system can guide the robot to follow these constraints at each time step, which can lead to a path toward the final configuration. An overview of the framework and the computation of constraint forces is shown in Fig. 3.1. We refer the readers to [GL02] for more detail.

As a result, solving a problem with physics-based planning first requires reformulating the scenario as simulation inputs and planning constraints. The simulation inputs consider the workspace or environment, obstacles and their geometry, and the robots and their geometry. Since physical bodies also include a mass (such that they have an inertia), some mass must be assigned to the robots as well. This value can be altered depending on the desired results, but a uniform mass across all bodies tends to work well in most of our applications. The challenge in physics-based planning is determining a suitable set of constraints for solving a planning problem. In many cases, the choice

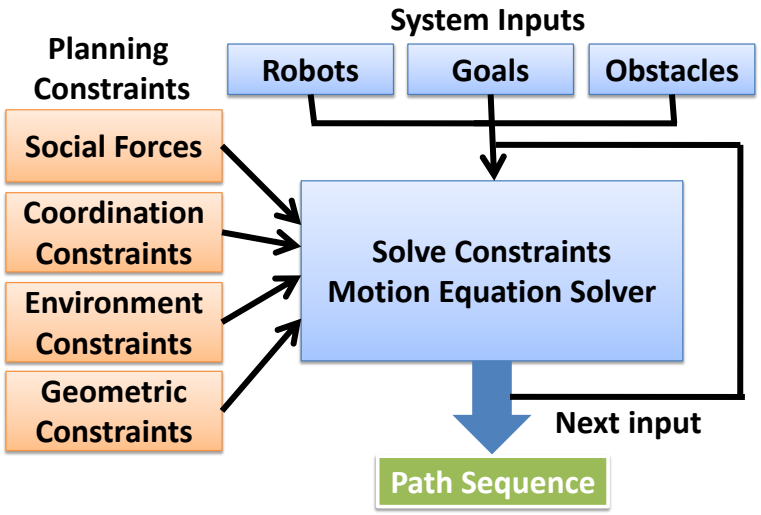

Figure 3.1: Planner Architecture: The physics-based motion planning cores uses the robots, goals, obstacles, and a set of planning constraints as its input. By solving the constraints (several different types are mentioned in the figure) and applying the resulting forces, motion is generated and the system is updated. The resulting value is a waypoint along the path and serves as the next input.

of constraints directly impacts the ability of the planner to control the agent and find a path as well as the performance of the planner.

For the remainder of this work, we will define two categories of constraints used in planning. *Hard* constraints are firm restrictions on the motion or state and must be enforced at all times. This is the type of constraint defined in Sec. 3.2.2. *Soft* constraints, on the other hand, are more akin to a relaxation of the constraint functions. These "constraints" are used to influence the motion or behavior at a given state, but are not required to be maintained at all times. It is an important to note that soft constraints, while mathematically defined in a manner similar to hard constraints, are not truly constraints. Instead, they are more related to biases on the motion. However, for the sake of uniformity, we will refer to them as constraints throughout the remainder of this work.

In summary, hard constraints directly remove or changes properties of the body or

its forces whereas soft constraints act alongside (or in opposition to) external forces. When combined, these constraints greatly enhance the capabilities of a simulation. The remainder of this section describes and defines both hard and soft constraints and also details the process of generating samples based on the simulation and constraints.

3.3.1 Hard constraints

Hard constraints are firm requirements that a particle or body must obey during a simulation and can take many forms. Constraints on the motion itself are usually applied via constraint forces and are solved as such. Since hard constraints are the type as defined in Sec. 3.2.2, details on how to define the constraint force for a particular constraint follow from the derivation.

Examples of hard constraints in robotics and planning include distance constraints, direct motion constraints, and non-penetration constraints. While joint or hinge constraints could fall into this category, common practice is to incorporate articulated motion directly into the motion equations. Direct motion constraints fix the body to a path or certain types of motions, such as bead traveling along a (solid) wire. Non-penetration and other constraints defined on geometric properties can be alternatively be enforced by geometric transformations or also by appropriately set constraint forces. In practice, both methods work well. We refer readers to Garber and Lin for additional common hard constraints used in planning problem [GL02].

In most cases, there will be a number of hard constraints acting on a simulation. It is entirely possible that solving one hard constraint will cause another hard constraint to become un-satisfied. To handle this situation, we iteratively relax hard constraints until they are all satisfied. Briefly, the value of each constraint function is evaluated. The constraint with the highest magnitude constraint value is solved and the result is applied to the simulation. This evaluation and relaxation continues until all constraints are satisfied or some fixed number of iterations. In practice, it is very uncommon to

reach a state which cannot be resolved.

Hard constraints enhance simulations by giving them a strong and stable means by which to restrict motion. For instance, it would be much easier to simulate a bead on a wire or a slot car racing along a fixed track or path. By augmenting simulation with well formulated hard constraints, users gain firm control while also maintaining a physically-plausible and smooth simulation over a variety of situations.

3.3.2 Soft constraints

Unlike hard constraints, soft constraints are relaxed requirements and are applied to a body without regard of the other forces acting on it. Again, it is this aspect that makes them not really constraints but more like biases or biasing forces. As such, these constraints are often implemented as "penalty" forces, i.e. when the constraint is not being satisfied, a force is applied to it that will move in a direction that will satisfy the constraint. Other factors which add external forces, such as energy fields and springs, can often be considered or adapted to be soft constraints as well.

For example, consider a simple "goal attraction" soft constraint. Clearly, this constraint will only be satisfied (its constraint function will evaluate to zero) once the robot, or whatever part of the robot on which the constraint is defined, reaches the goal. A penalty force for this constraint would simply point directly toward the goal and could be applied to the entire robot or to whatever part of the robot needs to reach the goal. Since soft constraints act irrespective of other constraints, these can be solved and directly applied without regard for the other soft constraints.

Soft constraints have several advantages, particularly in that they can influence the motion of a system or otherwise attach some behavior to it. However, at the same time, there are many challenges in choosing a good set of soft constraints such that they are strong enough to have the desired effect while not causing the system to become stiff. It is easy to define a large number of soft constraints, but this can be detrimental

to performance. Finally, it is important to evaluate soft constraints prior to the hard constraints since it is possible that a force from a soft constraint could cause a hard constraint to become un-satisfied.

As with hard constraints, many more soft constraints will be described in the subsequent chapters. These constraints complement hard constraints and provide a flexible to influence the underlying motion of a body in the simulation. In some cases, soft constraints can be thought more of as directly a simulation toward a specified direction or behavior as much as they also alter the underlying motion.

3.3.3 Sample Generation

Given an appropriate set of constraints, the final task is to use those constraints to generate samples. In many aspects, this methodology effectively generates joint samples in a search direction biased by the constraints. Samples are built off one another (akin to state-space RRTs [LK99]) by solving the motion equation ordinary differential equations. sampling in state or configuration space and moving in that direction, we use workspace distance information in order to generate forces which direct the growth.

The ideas behind biasing forces are similar to potential field method for a high-DOF robot, and similarly is susceptible to local minima in the force field defined by the biasing forces. To help alleviate the impact of local minima, it is common practice with physics-based sampling to generate an initial path or roadmap prior to generating samples. After extracting a path to the goal, a soft constraint defined using this initial path can aid in avoiding or escaping these minima.

Given the set of constraints which are treated as biasing forces in the system, each simulation step is then solved using numerical methods to advance toward the goal configuration. The general procedure is given in Algorithm 1. This fragment is repeated until we reach the final configuration. This simulation framework generates samples that are physically plausible at a potentially fast rate.

Data: Robot r
Data: Obstacles $\mathbf{O}$
Data: Soft constraints $\mathcal{F}_{soft}$
Data: Hard constraints $\mathcal{F}_{hard}$
$\mathbf{f_{net}}$ = ComputeExternalForces(r, $\mathbf{O}$, ...);
$\mathbf{f_{net}}$ = $\mathbf{f_{net}}$ + SolveSoftConstraints($\mathcal{F}_{soft}$,r, $\mathbf{O}$, ...);
Robot r_{next} = UpdateState(r,$\mathbf{f_{net}}$);
while *NotSatisfied*(r_{next}, $\mathcal{F}_{hard}$) **do**
    $\mathbf{f_{net}}$ = RelaxConstraints(r_{next}, $\mathbf{f_{net}}$, $\mathcal{F}_{hard}$);
    r_{next} = UpdateState(r_{next},$\mathbf{f_{net}}$);
end
return r_{next};

Algorithm 1: **Generating physics-based samples:** This algorithm generates generically performs a simulation step while maintaining the various hard and soft constraints. Given the simulation inputs and constraints, the algorithm first computes any forces due to external simulation factors and soft constraints (ComputeExternalForces(r, $\mathbf{O}$, ...) and SolveSoftConstraints($\mathcal{F}_{soft}$,r, $\mathbf{O}$, ...), respectively. Then, it solves the motion equations (UpdateState(r_{next},$\mathbf{f_{net}}$)) in an iterative manner and relaxes the hard constraints while simultaneously adjusting the net force (RelaxConstraints(r_{next}, $\mathbf{f_{net}}$, $\mathcal{F}_{hard}$)). Finally, a new sample is returned.

3.4 Discussion and analysis

Physics-based sampling offers a novel way to approach the general motion planning problem. As with most other planning algorithms, there are several trade-offs and limitations on the effectiveness of the approach. Since many of the goals and driving applications motivating this work require finding paths for very high-DOF robots in geometrically complex environments, there are many considerations which must be taken into account. In both concept and practice, motion planning based on physics-based sampling has able to accomplish many of these goals. In this section, we discuss several key aspects of physics-based sampling and compare it to related planning methods.

3.4.1 Performance and applicability

The main advantage of physics-based sampling is that the majority of the work is done in the workspace, as opposed to the configuration space. Therefore, the dimensionality

of the configuration or state space plays less of a role despite the fact that it is inherently navigating in these spaces. As a result, performance is instead dominated by that of the simulation core, and particularly the expected performance of generating a single sample and each task in a simulation step can potentially be a large bottleneck. Fortunately, a wide body of research exists on fast and effective simulation.

External force computation usually includes physical elements which are outside the scope of soft constraints (although they can sometimes be treated progamatically like soft constraints), such as gravity. Ordinarily, this does not pose a large computational task.

The performance of constraint solving varies greatly based on robot and environment. For higher DOF robots, there tends to be many more soft constraints whereas each hard constraint usually takes substantially longer to relax or resolve. For example, each surface elements on deformable robots is a single soft constraint. On the other hand, the problem of non-penetration on a deformable robot is considerably more difficult especially if self-collisions are possible. The computational burden of both soft and hard constraints must be considered when defining the constraints. Many examples of the performance of both soft and hard constraints are in the subsequent chapters.

Finally, state updates can be particularly intensive and the magnitude of the time step plays a large role in the total planning. Since the size of time step bounds the total distance the robot can travel, if a time step is too small then the robot may not make sufficient progress toward the goal. To complicate the issue further, there is a clear trade-off between performance and accuracy. Larger time-steps tend to lead to less accurate (and sometimes non-converging) simulations. More accurate ODE solvers (which allow larger time-steps) perform several force evaluations (and hence constraint solving) per step. In any planning situation, the balance between accuracy and performance must be considered.

By optimizing across all of the items, it is possible to use physics-based simulation

for a wide variety of higher DOF problems. As is described in the subsequent chapters, planning using physics-based sampling can solve problems with hundreds and thousands of DOFs and in environments with tens of thousands of triangles in a relatively short amount of time.

3.4.2 Compared to potential field planners

Physics-based sampling shares several ideas with potential field planners. In general, soft constraints (and particularly the associated biasing forces) act much like potential fields. As a result, physics-based sampling shares a problem of local minima. In practice, planners based on physics-based sampling use a guiding path to help steer the robots out of local minima.

For low DOF (purely translational robots) robots, the resulting motion of potential fields and physics-based sampling can be fairly similar depending on how the soft constraints are defined. However, general potential fields are less practical higher DOF robots (since the potential field is often created in the configuration space rather than the workspace). However, by planning in configuration space, certain variations of potential fields have been shown to be probabilistically complete whereas there are currently no clear convergence guarantees for physics-based sampling. This is an area of future work for the approach.

3.4.3 Compared to randomized planners

Randomized methods such as PRM and RRT explore the configuration space by random sampling and tree expansions, respectively. Both are known to be effective for relatively high DOF situations. Both of these randomized methods have been shown to handle 10s of DOFs in a practical amount of time. For certain classes of robots and tasks, these methods have even been shown to successfully plan for hundreds of DOFs (e.g. [LC03a]). In comparison, the class of planning methods developed in this work scales

to thousands of DOFs.

The type of paths generated by each approach will vary significantly. Most randomized solutions incorporate straight-line local paths between milestones. As a result, the final path will only be smooth if additional smoothing is used or if planning is done in the state space. Both of these operations increase the complexity of the randomized approaches, but are features that are built into physics-based sampling due to way it samples.

Finally, randomized approaches are probabilistically complete which again is a quality that needs to be considered with respect to motion planning using physics-based sampling.

Chapter 4

Deformable Agents

4.1 Introduction

A vast majority of the work in motion planning has focused on robots that are either rigid bodies or articulated models. To this point, there has been comparatively little research on motion planning of a deformable or otherwise flexible robot. Flexible robots are becoming increasingly important in industrial and medical applications. For instance, motion planning can be used for cable placement in large factory plants and buildings, wire routing for nano-, micro-, to mega-scale electronic and mechanical structures, surgical procedure planning and simulation, etc. Endoscopic manipulators for minimally invasive surgery, power assist suits for human-movement support, and flexible agents for entertainment are some examples of a growing number of "deformable robots" populating through many different applications. One of the major challenges in this area is controlling and planning the motion and behavior of these robots in simulated environments.

A driving application of this work was insertion of flexible catheters in human vessels for planning and guiding surgical procedures [GHSC96, vWV98]. For example, consider the situation in Figure 4.2. In a procedure called liver chemoembolization, a thin flexible catheter is inserted into an opening in the femoral artery and must be navigated to

reach a tumor in the liver. There are multiple concerns regarding the choice of catheter. Manipulation of catheters in small vessels frequently causes spasms if the cross-sectional area of the catheter is close to that of the vessel being navigated, then the size similarity will reduce fluid flow. On the other hand, if the cross sectional area is too small then it may be difficult to use it to administer a treatment. Accurate path planning studies can help overcome these obstacles by becoming an integral part of preoperative surgical planning, i.e. choosing the size and properties of the catheter. However, accurate geometric models of the arteries and a deformable catheter consist of tens of thousands of primitives (e.g. polygons). Many algorithms for motion planning and collision detection between deformable models are unable to handle models of relatively high geometric complexity.

Motion planning for deformable robots introduces several challenges. In contrast with rigid or articulated robots, the space of all states of a deformable body is intractably large. Even a simple discretized deformable object can have hundreds of control points or elements, leading to thousands of degrees of freedom. Furthermore, the task of generating physically-plausible motion for a deformable body is considered a difficult problem. In order to create any planning algorithm for flexible robots, we need to either enumerate its configurations [ABG06] or be able to model the physical properties and mechanical constraints of the robots. The computational requirements of generating an accurate deformation using a continuum model can be rather high.

Another challenge is fast and accurate collision detection between a deformable robot and surrounding obstacles. Current algorithms for collision detection between deformable models have a high overhead. Moreover, in many deformable planning scenarios, the free space of a robot becomes very constrained. The robot is often close to the obstacle boundary, leading to an extremely high number of proximity queries or complex interactions with the environment. As a result, collision detection is a major bottleneck in terms of developing efficient planners.

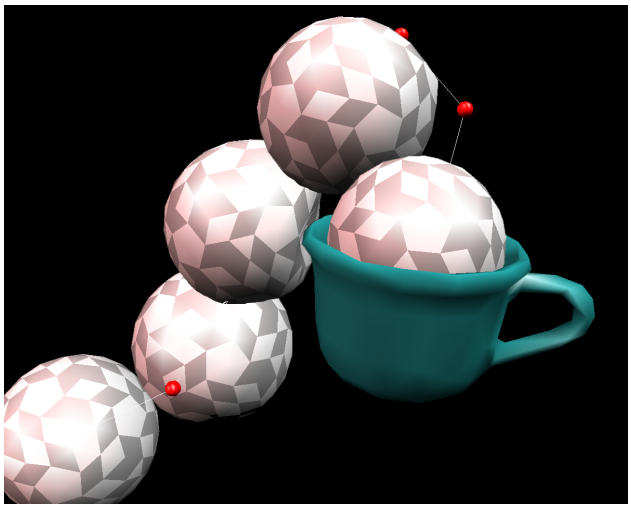

Figure 4.1: The Ball in Cup Scene: The goal is to plan a path to move the ball into the cup. Since the cup's rim has about the same diameter as the ball, a deformation on the ball simplifies the path planning. The image, generated by our planner, shows the path of the deforming ball as it moves toward the center of the cup. Note that the final path causes the ball to deform around the rim of the cup.

The standard algorithms for rigid robots are not directly applicable to deformable robots. Most of the related work on motion planning for deformable agents has focused on smaller environments with relatively simple agents. At the same time, the case of deformable bodies can be seen as being more forgiving or lenient than rigid or articulated bodies. In cases of narrow passages, in many cases a deformable robot can be "pushed" or "pulled" through rather than having to move into a particular configuration. However, there are also limits to how much a body can bend or stretch, just need to be maintained.

One of our major goals is to investigate and develop physically-based sampling methods for general deformable robots. More specifically, we extend the following modified version of the general motion planning problem: Given a robot which deforms in a physically realistic manner and an environment with many static obstacles, find a path from an initial configuration to a final configuration. In addition, we would like to design a planner to be as efficient as possible.

In this chapter, we apply our physics-based sampling framework to the problem of

planning for deformable bodies. In our solution, the motion of any point on the body and the body itself can be represented and controlled via a simulation core. For deformable agents, it is important that our physics-based sampling framework incorporates the available motion of the body, any constraints on the body and its motion, and also a means to generate and follow some initial path or trajectory.

The choice of an initial trajectory heavily influences the resulting motion. For planning of this trajectory, the medial axis of the workspace is approximated and used as a roadmap. This is essentially a roadmap for a point-robot in the workspace. From this roadmap, an initial trajectory is determined by connecting the start and goal positions to the roadmap and performing a path search. In most cases, this trajectory is likely to have collisions for our deformable robot. As our the deformable agent navigates the environment, physics-based sampling makes appropriate path corrections in order to ensure that the path is penetration-free while simultaneously maintaining deformation and soft-body constraints. Propulsion along the path is performed by randomly selecting or specifying a "control" point (e.g. the "head" of a thin tube or the center of mass of flexible ball) on the agent and then "pushing" or "pulling" the point along the path.

We have implemented and applied our planner to a variety of simple and complex environments, including a case study of path planning of catheters for liver chemoembolization. Our planner can compute a collision-free path for a deformable robot in a complex environment consisting of tens of thousands of polygons in under two hours.

There are a few remarks about the approach developed here. The choice of a medial axis is to facilitate the motion since it will be optimally far from obstacles, although in general any initial trajectory or general roadmap in the workspace will suffice. For realistic and smooth deformation, it is important that both non-penetration and volume preservation constraints are accurately and reliably integrated with the framework. This allows the robot to deform in a plausible manner as it approaches or comes in contact with a surface. Since these constraints can be computationally expensive, we relax the

strict global volume preservation constraint by implementing a local method that sets a threshold on the amount of deformation (e.g. elongation or compression) and on internal pressure fluctuation. Since collision detection is a bottleneck, we develop and utilize graphics hardware to accelerate the collision queries.

4.2 Overview

In this section, we introduce additional notation related to deformable body simulation and our planning algorithm and give a brief overview of the approach.

4.2.1 Notation and Definitions

In relation to simulation, there are a wide variety of methods available for representing and simulation a deformable agent. Among the simplest methods is a spring-mass system, where the body is described by a collection of point masses, or particles, connected by springs and organized into a lattice structure. In practice, mass-spring systems are easy to construct and can be simulated at interactive rates on current commodity hardware. More accurate physical models treat deformable objects as a continuum. One of the most commonly used continuous models is the finite element methods (FEM). The object is decomposed into elements joined at discrete node points and a function that solves the equilibrium equation is computed for each element. The computational requirements of FEM can be high (as a function of model complexity) and it is difficult to use them for complex models in real-time applications. In fact, a spring-mass system can be seen as a discrete version of FEM.

The deformable robot, r, is represented as a time dependent set of n particles, $\mathcal{P}(t) = \{\mathbf{p}_1(t), \mathbf{p}_2(t), \ldots, \mathbf{p}_n(t)\}$, connected by a set of springs $\mathcal{E}$. Each spring $s_{i,j} \in \mathcal{E}$ attaches $\mathbf{p}_i$ to $\mathbf{p}_j$ and additionally has a rest length l, a spring constant k_s a damping constant k_d, a stress *stress*, and a threshold value δ_s. Briefly, rest length and the two constants define

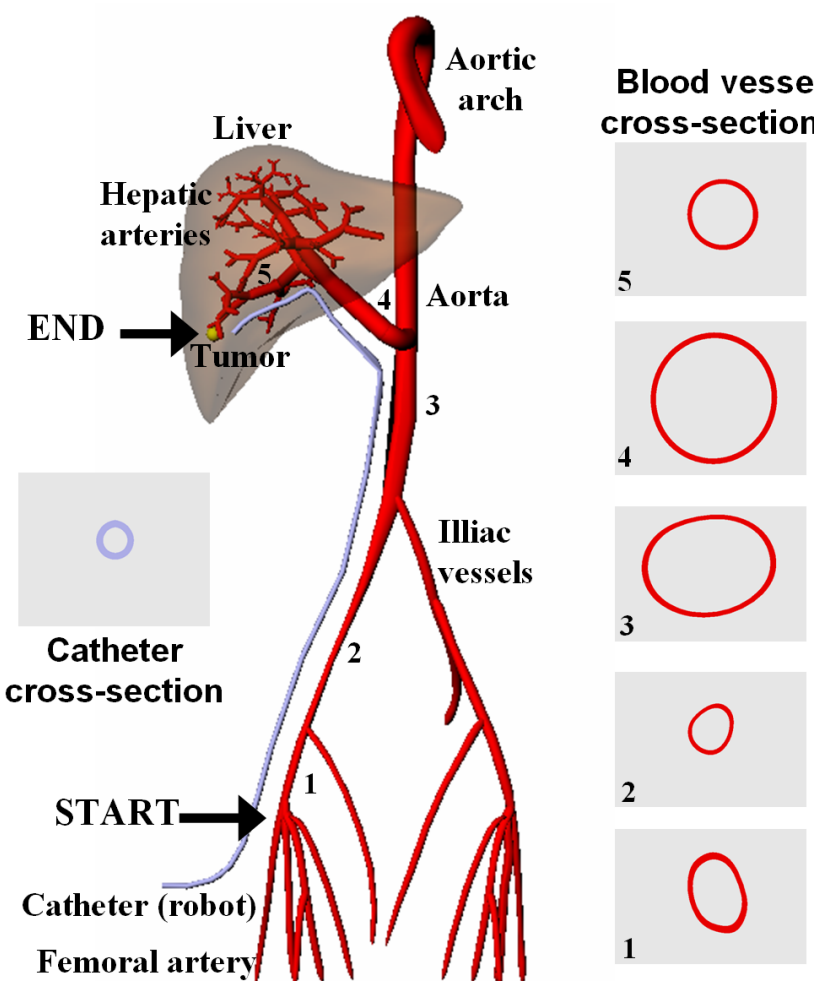

Figure 4.2: Path Planning of Catheters in Liver Chemoembolization: The deformable catheter (robot), represented by 10K triangles, is 1.35mm in diameter and approximately $1,000$mm in length. The obstacles including the arteries and liver consist of more than 83K triangles. The diameter of the arteries varies in the range 2.5-6mm. Our goal is to compute a collision free path from the start to end configuration for the deformable catheter. The free space of the robot is constrained. The path computed by our motion planner is shown in Fig. 4.9.

the behavior of the spring and elasticity of the body, while the stress and threshold are used to define material constraints. Finally, the set of positions of the particles $\mathcal{X}(t) = \{\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots, \mathbf{x}_n(t)\}$ represent the agent's configuration, $\mathbf{q}$, at time t and the set of states of particles (positions and velocities) $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{v}_1)^T, (\mathbf{x}_2, \mathbf{v}_2)^T, \ldots, (\mathbf{x}_n, \mathbf{v}_n)^T\}$ is the agent's state, $\mathbf{s}$.

Since the shape of the body can change over time, we define other terms over the body which are used to generate physically-plausible motion. We define a deformation energy function $\mathcal{E}(r)$ of the robot. $\mathcal{E}(r)$ simulates the potential energy of elastic solids, and is a measure of the amount of deformation from its rest state. The robot can be interactively deformed as contacts occur with the obstacles in the environment. This deformation may change the volume $V(r)$, as well as the energy $\mathcal{E}(r)$. To simulate physically plausible deformation, we need to find a new configuration of r, that preserves the total volume of the deformed robot, while minimizing the the total energy of the system.

Our approach uses a *roadmap* extracted from the medial axis to answer simple planning queries and provide a initial trajectory for the robot to follow. Again, the following is done by selecting or specifying a control point or a series of control points on or in the robot. The roadmap, $\mathcal{G}$, consists of a set of milestones, $\mathcal{M}$, and a set of links, $\mathcal{L}$. Our initial trajectory using this roadmap is the sequence of connected milestones and their links.

Problem Formulation: Given these definitions, we restate our problem as: Find a sequential set of robot configurations $\mathbf{q}(t_0), .., \mathbf{q}(t_1)$ such that no $\mathbf{q}(t_i)$ intersects any obstacle in $\mathbf{O}$ and $\mathbf{q}(t_i)$ satisfies the non-penetration and volume preservation constraints, where $\mathbf{q}(t_0)$ is the initial configuration of the robot, and $\mathbf{q}(t_1)$ is the final configuration.

4.2.2 Extension to Deformable Agents

We need to determine a set of constraints which will be sufficient to solve planning problems for a deformable body. We introduce constraints designed to guide the *deformable*

robot through the environment to the desired goal configuration. Using global analysis from any roadmap algorithm (e.g. PRM) to compute a possible path for a point robot, we define both geometric and physical constraints that move the deformable robot to avoid both static and moving obstacles, and also follow an estimated path to the goal. Finally, we identify some computational bottlenecks of the process and propose methods to optimize the performance of these portions. The main components of our physics-based motion planning approach include:

1. **Guiding Path**: Our framework allows the use of any estimated path computed by either quick global analysis of the environment, such random sampling [KSLO96, FGLM01, LK00] or user guided input [BSA00]. Since we can use a path for a point-sized robot, we can extract the medial axis of the workspace as our roadmap.

2. **Constraints**: To achieve the desired results without robustness issues, we determine a set of hard and soft constraints to guide our sampling process. Along with the standard hard constraints (obstacle non-penetration and staying within the environment boundaries), we additionally define soft-body and volume preservation constraints. Global volume preservation is relaxed for performance, and instead we define an internal pressure on the body and use this to help locally preserve the body's volume. Necessary soft constraints include a path following constraint and collision response to guide the robot along its initial trajectory and to keep the robot from intersecting bodies. Thus, our approach will compute the new path by taking into consideration the body's deformable properties and the interaction of the flexible robot with the obstacles in the workspace.

3. **Discretization of Continuum**: As for modeling the deformation of the robot or the environment, there exists many possible approaches. For example, these may include *physically-based* free-form deformation (FFD), mass-spring systems, boundary element methods (BEM), and finite element methods (FEM). The choice

of discretization impacts performance and also how the soft-body constraints are enforced. For coding simplicity and runtime performance, we chose an implementation based on the spring-mass system to validate our basic approach. However, we can easily incorporate any discretization technique within this algorithmic framework.

4. **Proximity Queries**: We can take advantage of methods proposed in [HCK+99, GRLM03] that uses graphics hardware to quickly perform proximity queries on the workspace or to provide dynamically updated discretized distance fields for obstacle avoidance involving deformable models.

4.3 Planning algorithm for deformable agents

Following our general physics-based planning framework, our algorithm for deformable agents consists of two phases: (a) off-line roadmap generation and (b) a runtime path query phase based using physics-based sampling. A detailed algorithm is given in Algorithm 2. The basic steps are:

1. **Roadmap Generation:** Create a roadmap, $\mathcal{G} = \{\mathcal{M}, \mathcal{L}\}$ of the environment with a point robot. Recall that links and milestones will be defined in the workspace.

2. **Guiding path:** Query the roadmap to find an initial path τ in $\mathcal{G}$ from the initial to goal configuration.

3. **Physics-based sampling:** Use physics-based sampling to follow the path (via the "control" points) while maintaining constraints. As the robot reaches each milestone in $\mathcal{M}$, we store the state, $\mathbf{s}$, of the robot in case we need to back-trace. If at any point the hard constraints cannot be satisfied, we return the agent to its last saved state and remove the link that we being traversed. Then, we replan the

initial trajectory. If no such path exists, we back-trace once again. If we return to the initial state, return with no path found.

4.3.1 Roadmap generation and guiding path

As described earlier, the goal of this phase is to create the set $\mathcal{M}$ of milestones and set $\mathcal{L}$ of links. For the sake of simplicity and efficiency, we treat the robot as a point robot. This strategy for simplifying the problem makes the generation of each milestone simple and quick. We use *estimated paths* to generate an initial approximation to the path. This idea has been used in motion planning [CB96, FGLM01, WAS99a].

A wide variety of planning methods are available to generate a primary roadmap. In particular, we use a medial-axis based approach that computes an approximate medial-axis of the work space using voxelized methods and performs path smoothing. This estimated path tends to result in smoother final paths that maintain the farthest distance from nearby obstacles.

It is likely that the initial roadmap generated could cause the resting shape of the robot to be intersecting with the environment. If the robot must reach one of these milestones during the simulation, it is possible that the robot will have to deform to reach the goal configuration subject to the physical constraint. In cases when it cannot, the edges linking to this milestone will be pruned away and the planner can look for another path. This is done by storing the state of the robot as it passes through each milestone. When a path forces the robot to go break a constraint, we remove the link and replan.

One potential drawback to this method is that the resulting path may not be a minimal energy path. This could be accomplished by computing the minimal distance from an obstacle to the link and weighting the links accordingly. Since the largest deformations are most likely to occur near an obstacle, by staying farther away from obstacles, we can reduce the energy required to perform all deformations along a path.

The main benefit of computing the roadmap in this manner is speed and simplicity.

4.3.2 Physics-based sampling phase

This phase follows the general framework described in Section 3.3. In particular, we augment this process to include the constraints and optimizations specific to planning for deformable robots. As a result, it creates a path for the robot which is smooth and physically-plausible for a deformable body. The remainder of this chapter details these additional constraints and optimizations.

4.4 Deformable agent simulation and constraints

In this section, we detail the components simulation framework used in the physics-based sampling core for deformable robots and the constraints used to direct and control the simulation.

4.4.1 Soft-body motion

Following our approach, we need to describe a deformable robot as a dynamical system. As mentioned earlier, we have chosen to discretize the continuum via a mass-spring system. Let $\mathbf{q}_t$ be the configuration of the robot at some time t and $\mathcal{X}(t) = \{\mathbf{x}_1(t), \ldots, \mathbf{x}_n(t)\}$ be the set of particle positions.

As before, a goal is to determine how this body moves. The motion equations of a mass-spring system can be described by a second-order ordinary differential equation (ODE):

$$M\ddot{\mathcal{X}}(t) + C(\dot{\mathcal{X}}(t)) + K\mathcal{X}(t) = \mathbf{f_c}(S^t) + \mathbf{f_g}(\mathbf{q}(t))$$

where M is a diagonal mass matrix (the mass of each particle $\mathbf{p}_i$ is on the i^{th} diagonal), C is a diagonal matrix of spring damping constants as specified similarly to M, and K is a

```
Data: Robot r
Data: Obstacles O
Data: Environment D
Data: Initial configuration q_initial
Data: Final configuration q_final
Roadmap G = BuildRoadmap(O, D);
Path τ = QueryRoadmap(G, q_initial, q_final);
r.q = q_initial;
τ.SetLastState(r.s);
while r.q != q_final do
    ConstraintForces) = ComputeConstraintForces(r, τ, O, D);
    SolveAndUpdate(r, ConstraintForces);
    if Not ConstraintsMet(r) then
        r.s = τ.GetLastState();
        G.RemoveLink(τ.CurrentLink);
        τ = QueryRoadmap(G, q_initial, q_final);
        if τ is empty then
            return "No Path Found";
        end
    end
    if τ.CrossedMilestone(r) then
        τ.SetLastState(r.s);
    end
end
```

Algorithm 2: **Planning for deformable robots:** This algorithm outlines the procedure used for path planning for deformable robots. Briefly, given a robot's description and an environment, first determine a roadmap in that environment (BuildRoadmap(**O**,**D**)). Then, extract a path for this particular planning query (QueryRoadmap($\mathcal{G}$, $\mathbf{q}_{initial}$, $\mathbf{q}_{final}$). After initializing planner state, the algorithm begins computing the constraint forces (both hard and soft constraints) (ComputeConstraintForces(r, τ, **O**, **D**). Next, it applies these forces and updates the state of r. If constraints cannot be satisfied from this position, then we perform our back-trace and path update steps. Finally, for maintenance, we store the state of the robot at milestones as we cross them (τ.SetLastState(r.**s**)), for use in the back-trace stages if needed. This process continues until the robot reaches its goal.

banded matrix of spring constants. Note that K is banded since it must represent spring forces which are functions of the distance between two masses. $\mathbf{f_c}(\mathbf{q}(t))$ and $\mathbf{f_g}(\mathbf{q}(t))$ are $3N$-dimensional vectors representing the constraint and external forces acting upon each of the N masses. To help reduce numerical instability from stiff systems, we solve the ODE with a semi-implicit Verlet integration scheme (see Appendix A.2). This solver requires only one additional force computation step, keeping the overall computational cost low. Thus, our simulation uses this scheme to generate the next physically-based sample for the robot's path.

After advancing the system, we must verify if the geometric and physical constraints are satisfied by iterating through the list of constraints. We verify the constraints subject to minimization of the total energy in the system. If any constraint is not satisfied, we perform a pruning step as previously mentioned by backtracking through the set of valid configurations stored at the milestones. This process is detailed in Algorithm 2.

4.4.2 Constraints

We impose a number of known geometric, physical, and mechanical constraints suitable for the problem and to handle deformations. In our implementation, the hard constraints are solved for and enforced through the collision detection and response, as described in 4.5. For deformable bodies, the soft constraints are often computationally expensive and additionally challenging. Here, we provide details on the soft constraints as they relate to our deformable agent.

Deformation Energy Function

The elastic deformation energy measures the amount of deformation. In this case, deformation is essentially local stretches of the body in various directions. If the motion is simply a rigid transformation, meaning that it preserves the distances between all particles (no stretches), the energy must be zero. The local deformation is governed by

the deformation gradient. The right Cauchy-Green tensor, $C = F^T F$, is physically the square of local change in distances due to deformation where $F = \mathbf{f_c}(S^t) + \mathbf{f_g}(\mathbf{q}(t))$ is the total force acting on the system. It measures the length of an elementary vector after deformation, and is insensitive to rigid body transformations.

Let $\mathcal{E}(\mathcal{X})$ be the energy density function of an elastic solid undergoing deformation. The total energy is obtained by integrating $\mathcal{E}(\mathcal{X})$ over the entire volume of the solid. $\mathcal{E}$ can be expressed as a function of the right Cauchy-Green tensor C for elastic materials unless zero $\mathcal{E}$ is allowed for a non-rigid transformation (spurious zero-energy mode).

In fact, the simplest law uses a quadratic function of the right Cauchy-Green tensor C [Tal94]. Since F and C are a linear and quadratic functions of $\mathcal{X}$ respectively, $\mathcal{E}$ is at least a quartic function of $\mathcal{X}$. We have chosen the energy function of a spring network that connects the neighboring nodes. The energy function can be written as:

$$\mathcal{E}(\mathcal{X}) = \sum_j \frac{k}{2}(||d_j|| - l_j)^2$$

where j is the index of a spring and l_j is the natural length of the spring and $||d_j||$ is the distance between two particles $\mathbf{p}_i$ and $\mathbf{p}_k$ connected by the spring.

Basically we would like to find $\mathcal{X}$ by solving

$$min \;\; \mathcal{E}(\mathcal{X}) \text{ subject to } \nabla V(\mathcal{X}) \leq \epsilon.$$

We relax the hard volume preservation constraint by allowing the change in volume to be less than a given tolerance ϵ. This problem can be solved by using a global constrained minimization technique. Our current implementation uses a local method that checks whether the internal pressure fluctuation is bounded and the deformation at each edge/spring s_i does not exceed certain pre-defined tolerance (i.e. $stress_i < \delta_s$) to achieve the same effects.

Volume Preservation

One limitation of the mass-spring system is the difficulty of representing objects with sharp edges. A possible solution would be to add extra angular springs. In order to achieve a similar effect and satisfy the volume preservation constraint, we rely on the ideal gas law. By definition, the force due to pressure on a surface, $\mathbf{f_p}$, has the magnitude:

$$\mathbf{f_p} = PA$$

where P is the total pressure inside the object, A is the surface area and $\mathbf{f_p}$ has the same vector direction as the face normal $\hat{\mathbf{n}}$ pointing away from the surface. To quickly approximate the pressure inside the object, we use the *ideal gas law*:

$$PV = nR_gT$$

where V is the volume of the robot, n is the number of moles of gas, R_g is the universal gas constant and T is the temperature of the gas. For any given situation, we can set nR_gT to be a user-defined constant since it should not vary within the simulation (otherwise we implicitly allow the volume to change). The remaining unknown in this equation is the volume of the robot. Given this formulation, we need to apply it as a constraint force on the particles in the body.

Once we solve for P, we can compute the pressure force $\mathbf{f_p}$ for a triangle. By computing this quantity for each mass, we obtain a simple simulation of pressure in a soft

object. For details, see Algorithm 3.

Data: Robot r

Data: Environment $\mathbf{D}$

V = ComputeVolume(r);

$P = \frac{nR_gT}{V}$;

forall *Face* f *on* r **do**

  $\mathbf{f_p} = PA_f$;

  forall *Particle* $\mathbf{p}_i$ *on a vertex of* f **do**

    nIncidentFaces = numFacesIncidentToParticle($\mathbf{p}_i$);

    $\mathbf{F}^{int}_{\mathbf{p}_i}$ += $\frac{\mathbf{f_p}}{nIncidentFaces}$;

  end

end

Algorithm 3: **Volume preservation forces:** This algorithm is used in the simulation core to determine the local volume preserving, pressure-based force on each particle. Briefly, ComputeVolume returns the total volume of the body, A_f is the surface area of face f, and numFacesIncidentToParticle($\mathbf{p}_i$) returns the total number of faces which include $\mathbf{p}_i$ as a vertex.

Volume Computation

To compute the volume, we break the space into triangular prisms. Each prism is formed as the swept volume from the triangle to its projection onto the XY plane. To find the volume of the prism, we multiply the average height by the area of the projected triangle. Or:

$$V = \sum A_{base} h,$$

where

$$A_{base} = \frac{1}{2}((p_2 - p_1) \times (p_3 - p_1))_z$$

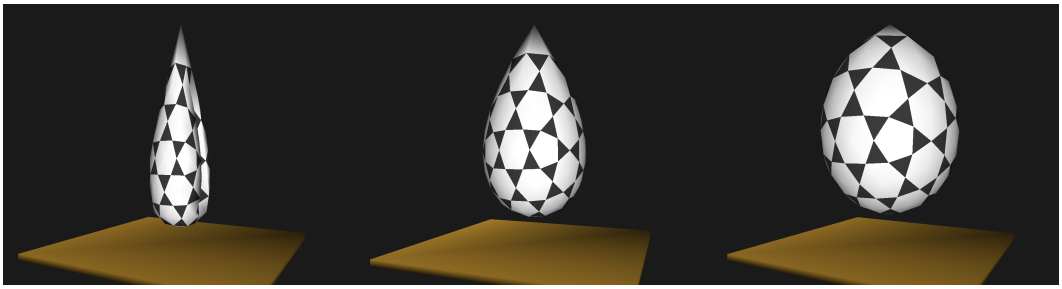

Figure 4.3: Soft-body robots with varying internal pressures: The leftmost robot has a relatively low pressure, while the rightmost has a fairly high pressure. The middle image has a pressure value in between the other two.

and

$$h = \frac{(p_1 + p_2 + p_3)}{3}.$$

where p_1, p_2, and p_3 are the vertices of the triangle. It is important to note that PV is a constant. Thus, if P does not greatly vary, then V also will not change much.

4.4.3 Deformation Step

Taking discrete time steps can occasionally lead to situations where the object penetrates or intersects one of the obstacles, if it is moving fast enough. Since a collision can also result in penetration, it is important to check for contacts during the simulation. For robustness of the implementation, we consider that the robot is colliding with an obstacle if the robot is within tolerance to an obstacle, or when the robot has actually intersected the obstacle. We can handle this situation by performing backtracking in time and then applying the collision resolution step. To accelerate the collision detection, we utilize the method described in Section 4.5.

4.5 Hardware Accelerated Collision Detection

The running time of most practical motion planning algorithms is dominated by collision detection [Lat99]. Collision detection is considered a major bottleneck in computing a

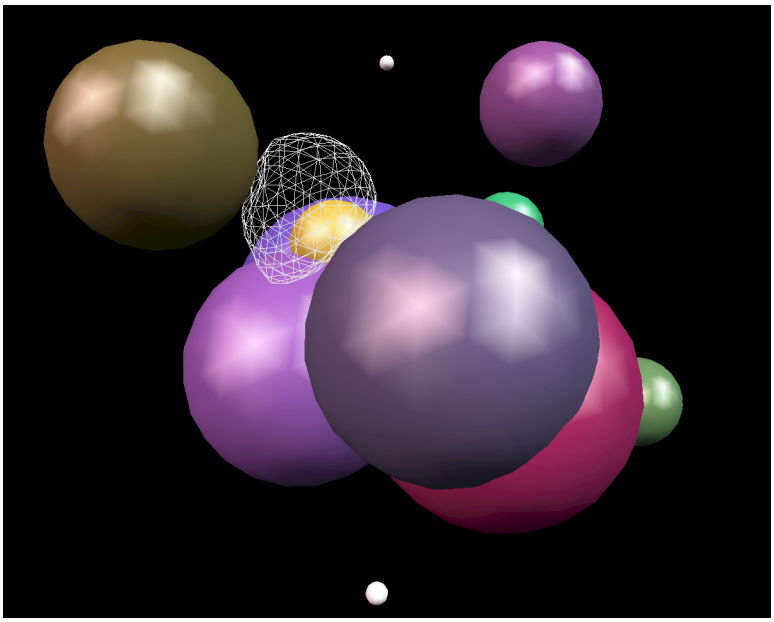

Figure 4.4: SphereWorld benchmark: The robot is shown in wireframe, and is deforming between some of the larger spheres. Each of the larger spheres are stationary, rigid obstacles.

collision free path for a robot from an initial configuration to a final configuration. Most of the prior algorithms are based on bounding volume hierarchies and work well for rigid robots [LM03]. Even in case of motion planning for rigid objects, collision detection can take more than 90% of the running time for a path planner [Lat99]. In the case of path planning for flexible or deformable robots, collision detection becomes a greater bottleneck for the following reasons:

- The free space of a deformable robot is constrained, and in several configurations the boundary of the robot comes into close proximity of the obstacle boundary. This close proximity leads to a higher number of potential contacts with the obstacles.

- Most prior collision detection algorithms are based on bounding volume hierarchies. As the robot deforms, the precomputed hierarchy needs to be updated to account for non-rigid motion. The cost of re-computing a hierarchy can be signif-

Scenario	Obstacle (tris)	Robot (tris)	Path Est. time (sec)	Total Sim. time (sec)	Average Step time (sec)
Ball In Cup	500	320	1	41.5	0.015
Sphere World	3200	320	1	333.16	0.077
Holes In Walls	216	720	48	605.958	0.037
Tunnel	72	720	575	833.24	0.068

Table 4.1: Deformable robot planning performance: This table highlights the performance of our planner running on a laptop with a 1.5GHz Pentium-4 processor. We highlight geometric complexity of the environment in terms of the number of triangles. We report the high-level path generation time and simulation time. The last column reports the average time taken per simulation step.

icantly higher for complex deformable models. Furthermore, the hierarchies may not be able to provide sufficient culling when the robot is in close proximity to the obstacles.

We present a collision detection algorithm for a deformable robot undergoing motion among rigid obstacles. Our goal is to compute a small subset of potentially colliding primitives (e.g. triangles) and only perform exact interference tests among these primi-

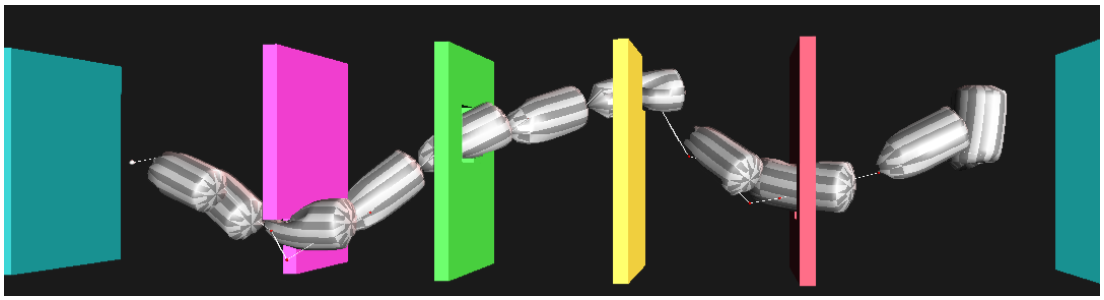

Figure 4.5: Walls with holes benchmark: This scenario shows a sequence of four walls with holes in them. To reach the goal, the robot travels through each hole. This figure highlights the various states of the deformable robot while traveling along the displayed path.

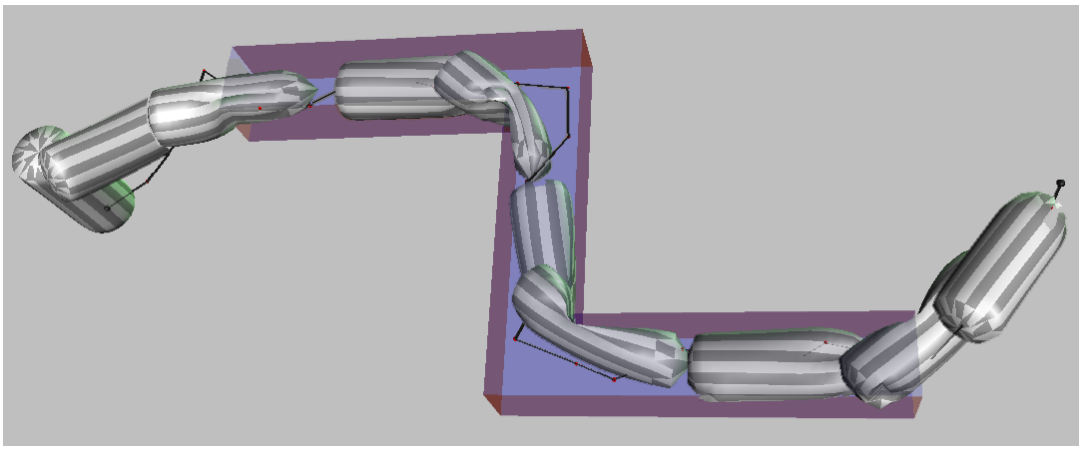

Figure 4.6: Tunnel benchmark: This environment is a simple tunnel which the robot must travel through in order to reach the goal. In this image, the striped cylinders represent the deformable robot at various states in its path through the tunnel. Note that a collision-free would not be possible if this robot was not deformable

tives. Our algorithm is based on two main components:

1. **Reliable 2.5D overlap tests using GPUs:** Since the robot is close to the obstacle boundary, we perform a tighter overlap test by checking whether there exists any separating surface between the robot and the obstacles. We perform this test using the rasterization capabilities of the GPU. We also compute *Minkowski sums* of the robot and the environment with bounded spheres in order to overcome image-precision errors.

2. **Set-based computations:** In order to deal with a high number of colliding primitives, we compute sets of potentially colliding primitives as opposed to computing each pair of overlapping primitives explicitly. The size of each set is at most $O(n)$, the number of triangles in the model, whereas the number of pairs in close proximity can be super-linear (or even $O(n^2)$ in the worst case) for a robot in a constrained free space.

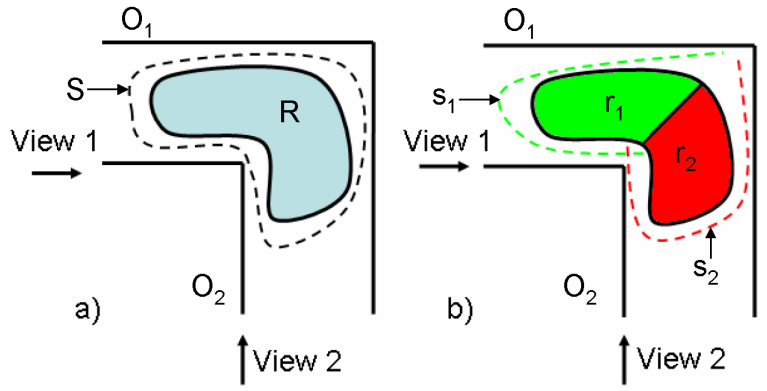

Figure 4.7: 2.5D overlap tests used for collision detection: The query checks whether there exists a separating surface along a view direction of depth complexity one. S has depth complexity more than one from View 1 as well as View 2. In the right image, s_1 has depth complexity one from View 1 and s_2 has depth complexity one from View 2. As a result, we use two 2.5D overlap tests to decide that R_1 and R_2 are not colliding with the obstacles (O_i).

4.5.1 Reliable 2.5D overlap tests using GPUs

The robot undergoes non-rigid deformation between successive steps of path planning. Instead of using BVHs, we check for overlaps between the robot and the obstacles using the rasterization capabilities of graphics processing units (GPUs). The GPUs are widely available on all commodity PCs and their computational capabilities are increasing at a rate exceeding Moore's law.

We perform visibility computations [GRLM03] between the objects on the GPUs to check whether $\mathcal{R}$ (robot) and $\mathcal{O}$ (obstacles) overlap. In particular, we choose a view direction, usually along an axis, and check whether $\mathcal{R}$ is fully visible with respect to $\mathcal{O}$ along that direction. If $\mathcal{R}$ is fully visible, then there exists a separating surface between $\mathcal{R}$ and $\mathcal{O}$ (see Fig. 4.7). Moreover, the separating surface needs to have a one-to-one mapping with a plane orthogonal to the viewing direction or depth complexity one along the view direction. We call this the *2.5D overlap* query; this provides a sufficient condition that the two primitives do not overlap. The 2.5D overlap test is significantly less conservative and more powerful as compared to earlier collision detection algorithms

that check whether two bounding volumes (e.g. spheres, OBBs, etc.) overlap in 3D. For example, in Fig. 4.7(b) there exists a single separating surface between R_1 and the obstacles as well as R_2 and the obstacles. In this case, we can verify with two 2.5D queries that the robot does not overlap with the obstacles.

A main problem with a GPU-based overlap test is the underlying image precision used to perform visibility computations. In particular, the rasterization of $\mathcal{R}$ or $\mathcal{O}$ introduces many sampling errors, including projective errors and depth-buffer precision errors. In order to overcome these errors, we compute and render a bounding offset for each object. Let the dimension of square pixel used for orthographic projection be p. Moreover, let S_p represent a sphere of radius $\sqrt{3}p/2$ and $\mathcal{R}^{S_p}$ and $\mathcal{O}^{S_p}$ represent the Minkowski sum of $\mathcal{R}$ and $\mathcal{O}$ with S_p, respectively. In this case, we use the following lemma:

Lemma: *If $\mathcal{R}^{S_p}$ is fully visible with respect to $\mathcal{O}^{S_p}$ from any view direction under orthographic projection on a 2D discrete grid with pixel size p, than $\mathcal{R}$ and $\mathcal{S}$ do not overlap.*

This lemma provides us with a sufficient condition that the robot and the obstacle do not overlap. The exact computation of the Minkowski sum of a primitive with a sphere corresponds to the offset of that primitive. The exact offset representation consists of non-linear spherical boundaries. Instead, we compute a bounding approximation of the offset. In case of obstacles, we decompose the boundary into triangles, edges and vertices. The offsets of each of these primitives are represented as swept sphere volumes: as rectangular swept-sphere (RSS), line swept-sphere (LSS) and point swept-sphere (PSS), respectively [LGLM99]. We precompute the swept spheres to enclose the obstacle primitives. Since the robot undergoes deformation, we dynamically compute a bounding OBB (oriented bounded box) for each triangle on the boundary. The cross-section of the OBB has the same plane as the triangle and the height of the OBB is equal to $\sqrt{3}p$. Moreover, we perform the 2.5D overlap test from a number of fixed directions

(e.g. X, Y and Z axes) to check for the existence of a separating surface.

4.5.2 Set-based Computations

Our algorithm uses the concept of a potentially colliding set (PCS) of objects or primitives [GRLM03]. In this section, we present a specialized algorithm for a deforming robot among fixed obstacles. Given a collection of primitives, $\mathbf{P} = \{p_1, \ldots, p_n\}$, we initially insert all the primitives into a PCS. Next, we check whether p_i overlaps with the remaining objects: $\mathbf{P} - \{p_i\}$. If they do not overlap, we remove p_i from the PCS. Based on this property, we reduce the number of object pairs that need to be checked for exact collision. There are two main issues in using set-based computations for collision detection:

- *Set-based overlap tests:* We need the capability to perform overlap tests between two different sets of objects. In particular, we need a simple test to check that the objects in $\mathbf{S_1}$ do not overlap with objects in $\mathbf{S_2}$. We use the reliable 2.5D overlap test described above.

- *Set partitions:* A set of n objects has 2^n subsets and we cannot check every possible pair of subsets for overlap. Rather we want to perform almost linear number of set-based overlap tests.

We compute two sets for collision detection. These are the $\mathcal{R}$-set and the $\mathcal{O}$-set. The $\mathcal{R}$-set $= \{r_1, r_2, \ldots, r_m\}$ consists of all the polygonal or triangular primitives used to represent the robot. If the number of triangles in the robot is high, we group them into small clusters and each r_i represents a cluster of triangles. In the same manner, $\mathcal{O}$-set $= \{o_1, o_2, \ldots, o_n\}$ is a set of obstacles in the environment and we ensure that each obstacle o_i does not have a high polygon count.

We update the vertices of the robot based on the deformation and compute a new bounding OBB for each triangle on its boundary. The set-based collision proceeds in

two passes.

In the first pass, we compute $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. In particular, $r_i \in \mathcal{R}$-PCS, if r_i does not overlap with all the obstacles in $\mathcal{O}$-set. Similarly, $o_i \in \mathcal{O}$-PCS, if o_i does not overlap with all the r_i's in $\mathcal{R}$-set. The $\mathcal{R}$-PCS is computed by performing 2.5D overlap test between each $\{r_i\}$ and $\mathcal{O}$-set. Similarly, $\mathcal{O}$-PCS is computed by performing 2.5D overlap tests between each $\{o_i\}$ and $\mathcal{R}$-set.

In the second pass, we perform set-based 2.5D overlap tests in a recursive manner. We represent $\mathcal{R}$-PCS $= \{\mathcal{R}_1, \mathcal{R}_2\}$, where $\mathcal{R}_1$ and $\mathcal{R}_2$ have approximately the same number of elements. Similarly we decompose $\mathcal{O}$-PCS $= \{\mathcal{O}_1, \mathcal{O}_2\}$. We perform 2.5D overlap tests between the following set combinations: ($\mathcal{R}_1$, $\mathcal{O}_1$), ($\mathcal{R}_1$, $\mathcal{O}_2$), ($\mathcal{R}_2$, $\mathcal{O}_1$) and ($\mathcal{R}_2$,$\mathcal{O}_2$). If none or only one of the 2.5D overlap test results in a separating surface, we terminate the recursion and perform exact collision checking between $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. Otherwise, we remove either $\mathcal{R}_1$ or $\mathcal{R}_2$ from $\mathcal{R}$-PCS or remove $\mathcal{O}_1$ or $\mathcal{O}_2$ from $\mathcal{O}$-PCS. The set-based culling algorithm is applied recursively to the new PCS's.

Analysis: The running time of the set-based culling algorithm for potentially colliding pairs is bounded by $O(m \log n + n \log m)$. We assume that the cost of performing each 2.5D overlap test is constant. The first pass of the algorithm takes linear time. In the second pass, the algorithm takes linear time during the first iteration. During each successive iteration we reduce the number of objects in one of the PCS's by half and therefore, performing $O(m \log n + n \log m)$ 2.5D overlap tests in the worst case.

4.5.3 Exact Collision Detection

Given the potentially colliding sets, $\mathcal{R}$-PCS and $\mathcal{O}$-PCS, we perform exact tests between the primitives to check for collisions. If the number of primitives is small, we check all pairwise combinations. Otherwise, we compute a bounding box for each primitive of $\mathcal{R}$-PCS and $\mathcal{O}$-PCS. We perform pairwise overlap tests between the bounding boxes by projecting the bounding box along the X, Y and Z-axes and compute the overlapping

intervals using insertion sort. If the projections of any bounding box pair overlaps along any axis, we explicitly check whether the corresponding 3D bounding boxes overlap and perform exact intersection tests between the primitives.

4.6 Implementation and Results

We have implemented and tested this algorithm on on two platforms. For a proof of concept design, we used a laptop with a 1.5 GHz Pentium-M processor, 512 MB of main memory, and a 64 MB ATI Radeon 9000 Mobility graphics card. Finally, to include the GPU-accelerated collision detection algorithm, we tested them on a PC with a 2.8 GHz Pentium IV processor, 1 GB of main memory, and a NVidia GeForce FX 6800 card. We used of NVidia's occlusion query extension along with offsets to perform visibility queries for reliable 2.5D overlap tests.

To test the effectiveness of our algorithm, we applied our planner to various scenarios that evaluate different aspects of the planner. They are as follows:

- **Ball In Cup** - The simplest of the scenarios, this environment has a spherical, deformable robot which must find its way into a cup. The cup's rim has been adjusted so that the ball will barely be able to fit in, and will thus have to deform to do so. (See Fig. 4.1)

- **Many Spheres** - This is primarily a test on how varying degrees of geometric complexity affects the performance of the planner. The environment is composed of a collection of spheres of varying positions and radii. A spherical, deformable sphere must move through these spheres to reach a goal in the opposite corner of the space. (See Fig. 4.4)

- **Walls with Holes** - Based on one of the Parasol Motion Planning benchmarks [Gro03], this scene has a sequence of four walls, each of which has a small hole.

In order to reach its goal, the robot must make its way through each hole. We modified the model to have a wide, flexible cylindrical robot that either cannot, or barely can, fit through the holes. It is wide enough that finding a collision free path would be very difficult, if not impossible, if the robot was rigid. The goal of this scenario was to fit a deformable robot through a small space. (See Fig. 4.5)

- **Tunnel** - This is also based on a Parasol Motion Planning benchmark [Gro03]. The scene contains a small tunnel with two bends in it. Like in the Walls benchmark, we augment the model by having a robot that is both long and wide. These constraints would make it impossible for a rigid robot to find a collision-free path. This scenario tested the planner's ability to force the robot to bend and deform around sharp corners. (See Fig. 4.6)

Results from each of these tests can be seen in their associated figures and are also summarized in Table I. The first two columns of the table describe the geometric complexity of the scene, for both the robot and the obstacles. The next columns give performance results of the planner by highlighting the time spent in various stages. One column gives the time the planner spent in high-level path generation, and the next gives the total time the planner spent in simulation steps. Lastly, we give an average step time for each simulation step.

Taking differences of systems into account, the average step time reported here is comparable to the time to process purely geometric deformations (based on FFD) used in other planners for deformable objects [BLA02a]. However, our technique also takes into account the physical constraints.

As can be seen from the table, the bulk of the time is in the simulation steps. These steps include the force calculation, collision resolution, establishing constraints, and solving the resulting system. Our current implementation is not optimized and the performance of each of these stages can be further improved.

4.6.1 GPU acceleration analysis

In order to test the effectiveness of our collision detection algorithm, we have done additional testing for two scenarios:

- **Serial Walls.** This scenario is based on a Parasol Benchmark [Gro03] in which a stick-like robot must navigate through a series of walls with holes (shown in Fig. 3). We have extended the benchmark by changing the robot into a soft-body sphere with 1280 polygons and the walls are represented using 18K polygons. As before, the sphere's diameter is set to be larger than the holes, but small enough so that the robot's material constraints allow it to fit through, forcing it to deform to reach its goal configuration. It takes about 16 minutes to compute a collision-free path.

- **Liver Chemoembolization.** This scenario demonstrates the ability of our planner to work in a realistic complex environment and acts as our target driving application for deformable path planning. We attempt to plan the path of a tube-like cylinder, called a *catheter*, through a set of arteries in order to mimic the catheterization process in liver chemoembolization. More details are given below.

We highlight the performance of our algorithm in these environments in the table and graph. Fig. 4.2 shows a detailed breakdown of the average time spent in various stages of a simulation step. The collision detection and response phase consists of both the 2.5D overlap tests along with the exact triangle-triangle intersection tests. Finally, we highlight the time spent in solving the motion equations and the total time spent on a time step. One curious result is that more time was spent on exact test in the Walls scenario even though it is a simpler environment. This result is due to greater culling effectiveness in the catheterization case.

Fig. 4.8 compares the effectiveness of the 2.5D overlap tests as a function of the scene complexity. To measure the performance of our new collision detection algorithm,

Scenario	Robot Complexity (tris)	Obstacle Complexity (tris)	Constraint update (s)	2.5D Overlap Test (s)	Exact Triangle Intersection Test (s)	Solve System (s)	Total Time (s)
Walls	1280	18432	0.0071	0.0232	0.0252	0.0323	0.0878
Catheter	10080	80086	0.0159	0.1596	0.0062	0.0227	0.204

Table 4.2: Planning performance with 2.5D overlap test: This table gives a breakdown of the average time step for each scenario. Constraint update refers to the time spent in computing each constraint for the given configuration. The 2.5D overlap test along with the exact triangle intersection test are the two stages of the collision detection algorithm. The Solve-System time is that spent in solving the motion equations during each step.

we varied the scene complexity of the Walls environment and ran our algorithm with and without the GPU-based 2.5D overlap tests. The case of no overlap test reduces to solely using the bounding volume hierarchies. In the graph, we see a greater speedup in collision detection as the obstacle complexity increases.

4.6.2 Path Planning of Catheters in Liver Chemoembolization

We use our path planning algorithm as a guidance tool for a catheterization procedure, specifically chemoembolization of liver tumors. Liver chemoembolization involves the injection of chemotherapy drugs directly into the hepatic artery that supplies a tumor. The procedure takes advantage of the fact that liver tumors obtain their blood supply exclusively from the branches of the hepatic artery. Under X-ray guidance, a small tube or *catheter*, is inserted into the femoral artery and is then advanced into the selected liver artery supplying the tumor. Chemotherapy drugs, followed by embolizing agents, are then injected through the catheter into the liver tumor.

During this procedure, careful manipulation of the catheters is essential [GHSC96, vWV98]. Manipulation of catheters in small vessels frequently causes spasms, which prevent adequate flow to carry the chemoembolization material into the tumor. Another

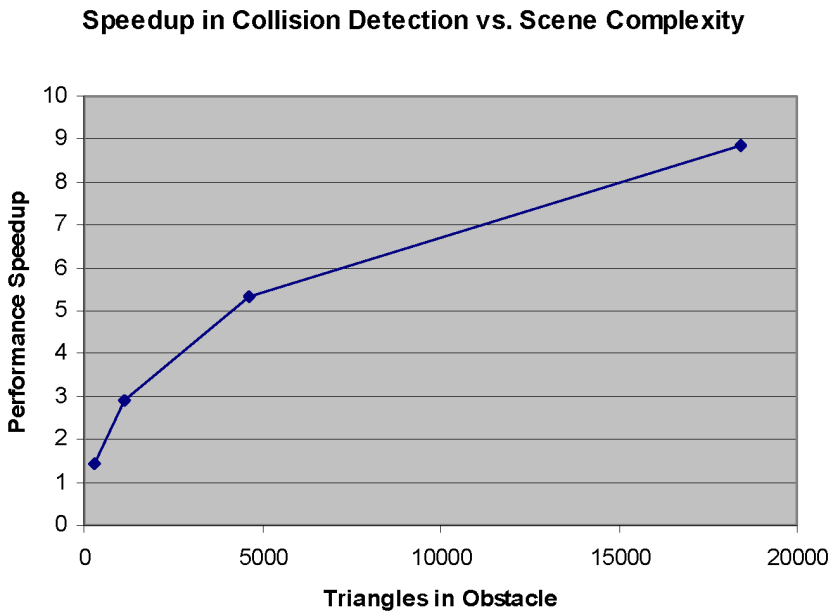

Figure 4.8: 2.5D speed-up vs. triangle count: This graph highlights the speedups obtained by utilizing the 2.5D overlap tests in our collision detection algorithm, as we increase the polygonal complexity of our scene. We observe nearly an order of magnitude improvement in complex scenes over prior algorithms based on bounding volume hierarchies.

problem may arise if the catheter has a cross-sectional area close to that of the vessel being injected. In this case, the size similarity will also reduce fluid flow and increase the risk of reflux of chemoembolization material into other arteries. Accurate planning studies can help to overcome these difficulties. Preoperatively, path planning can be used as part of surgical planning techniques to help choose the size and properties of the catheter used. We used a geometric model of the catheter and arteries shown in Fig. 4.2 (along with their relative dimensions). The catheter is modeled using $10,080$ triangles. The model of the arteries consist of $70,006$ triangles and the liver is represented using $12,459$ triangles. The start and end configuration of the catheter are shown in the same figure.

We used our motion planning algorithm to compute a path for a flexible catheter to a specific hepatic artery that is supplying a tumor inside the liver. The 3D models of

the liver and the blood vessels, that make up the environment, were obtained from the 4D NCAT phantom [Seg01]. The flexible, snake-like, catheter was modeled as a cylinder with a length of 100 cm and a diameter of 1.35 mm. Figure 4.9 shows a 3D rendering of the models used in this study with the starting (insertion of the catheter) and ending (tumor supplying vessel) locations marked.

Despite the scenario's complexity, our planner was able to successfully plan a path for this problem. A breakdown of the step time averages is given in Fig. 4.2. As the table shows, a large portion of the computation time is spent done in the collision detection phase (more than 80%). Further optimizations in GPU-based 2.5D overlap tests would improve the performance of the overall planner.

We use a large number of material constraints and are able to generate fairly smooth deformations throughout the simulation (as shown in Fig. 4.9). An additional path smoothing step further helps to improve the quality of the deformation.

4.6.3 Conclusion

In this chapter, we have developed constraints and methods for path planning for deformable bodies. Since we use a well-established motion model and realistic physically-driven interactions with the environment, the resulting path is smooth and reasonably "natural" in terms of the generated motion. Furthermore, with physics-based sampling and the accelerated simulation, the planner typically returns a path in a reasonable amount of time.

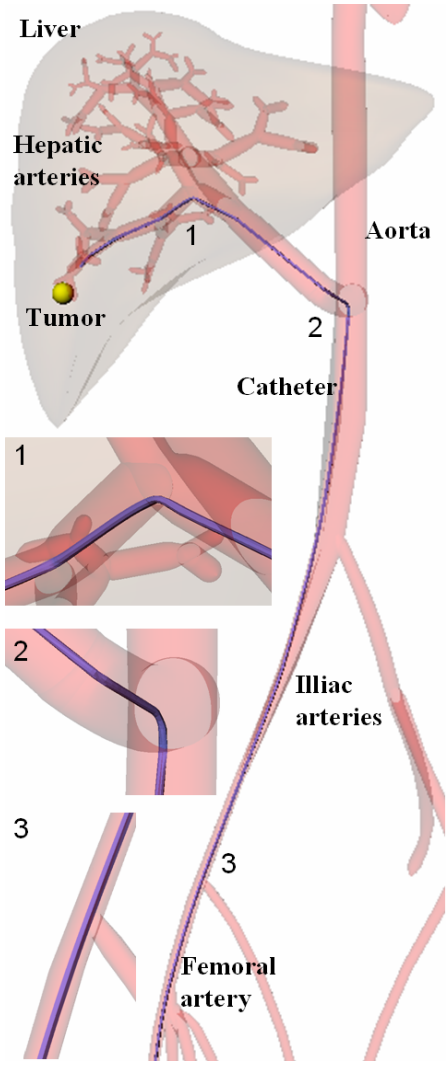

Figure 4.9: Path Planning of Catheters in Liver Chemoembolization: We highlight the collision-free computed by our algorithm for the catheter shown in Fig. 4.2. We show the overall path from the start to the end configuration in the rightmost image. The left images highlight the zoomed portions of the path, showing bends and deformations.

Chapter 5

Articulated Agents

5.1 Introduction

Modeling and simulation of multi-body dynamical systems has been well-studied due to its wide applications including robotics and automation, molecular modeling, computer animation, medical simulations, and engineering analysis. Examples of highly articulated robots include snake or serpentine robots, reconfigurable robots [CB94b, GRC05], and long mechanical chains. In molecular modeling, long series of atoms typically represented by hundreds or thousands of links are commonly used [Sur92]. Catheters [GSLM05b], cables [LK04], and ropes can also be modeled as articulated robots with a very high number of joints.

Highly articulated robots, such as snake or serpentine robots, with many degrees of freedom (DoFs) have recently received considerable attention from the robotics community [CB94c, HHC98, WJC+03]. Chirikjian and Burdick first introduced the term hyper-redundant robots to describe such robots with a very high number of 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.

As previously mentioned, randomized motion planning research has made great strides toward providing planning solutions in high dimensional configuration spaces. Probabilistic roadmaps (PRMs), rapidly-exploring random trees (RRTs), and several variants can solve complex problems with dozens of DoFs [KSLO96, KL00]. Since they rely upon randomization to map or explore the configuration space, their performance degrades considerably for robots with much higher dimensionality (e.g. hundreds or thousands) for a couple of reasons[PK06]. As the number of degrees of freedom increases, it becomes more expensive to compute representative samples of the configuration space for these planners. Moreover, dynamics or kinematics constraints on the problem further constrain the search space and make sample generation even more difficult since many planners only consider geometric constraints (such as non-penetration). For instance, consider the motion generated by a either a PRM or RRT. In most variations of these methods, the links are typically generated as *straight-line* paths in the configuration space using a local planner and only take into account geometric constraints (e.g. collision avoidance). As a result, motion is non-smooth when transitioning between links. A local planner will also have to perform additional checks along the link to ensure that kinematics constraints are satisfied. On the other hand, if the motion generated was completely physically-based, the path would be more realistic.

One of the key components of multi-body dynamics simulations is forward dynamics computation, which determines the acceleration and resulting motion of each link, given a set of active joint forces and external forces. The optimal algorithms have a linear runtime complexity in the number of degrees of freedom (DOFs). However, these algorithms may still not be sufficiently fast for complex systems that have a very high

number of degrees of freedom. Furthermore, if the environment consists of many obstacles or multiple articulated robots, dynamic simulation with robust contact or collision handling can become a major bottleneck for real-time applications.

5.2 Overview

In this section, we define the notation used throughout the chapter and introduce the main ideas needed for physics-based sampling for highly articulated bodies.

5.2.1 Notation and Definitions

Our articulated robot is represented using Featherstone's recursive definition of an articulated body: an articulated body is composed of two other articulated bodies that are connected through a *principal joint* (See Fig 5.1) at a "handle", the location on the link where the joint is connected [Fea99a].

At the base level, we have a set of n rigid bodies. For simplicity, we define our joint primitives to be 1-degree of freedom (DoF) revolute joints. More complex 2-DoF revolute joints can be added in a standard way as follows. A rigid body, b_{thin}, with zero (or sufficiently small) width is placed between each pair of bodies, b_i and b_{i+1}. A standard 1-DoF joint will connect b_i and b_{thin}, and another 1-DoF joint rotated 90 degrees about the center of the chain will connect b_{i+1} and b_{thin}.

We represent the articulated body's *state* at time t as $\mathcal{A}(t) = \{\mathcal{B}, \mathcal{J}, \mathbf{q}(t), \dot{\mathbf{q}}(\mathbf{t})\}$, where $\mathcal{B} = \{b_1, b_2, \ldots, b_{2n-1}\}$ is the set of rigid bodies, $\mathcal{J} = \{j_1, j_2, \ldots, j_{2n-2}\}$ is the set of joints, $\mathbf{q}(t)$ is a vector of length $2n-2$ of joint positions, and $\dot{\mathbf{q}}(\mathbf{t})$ is a vector of length $2n-2$ of joint velocities at time t. To build the articulated body, pairs of bodies are connected via principal joints from the bottom up, following the recursive definition. The robot's principal joint is then the middle joint along the linkage. This process forms an *assembly tree* (See Fig. 5.1).

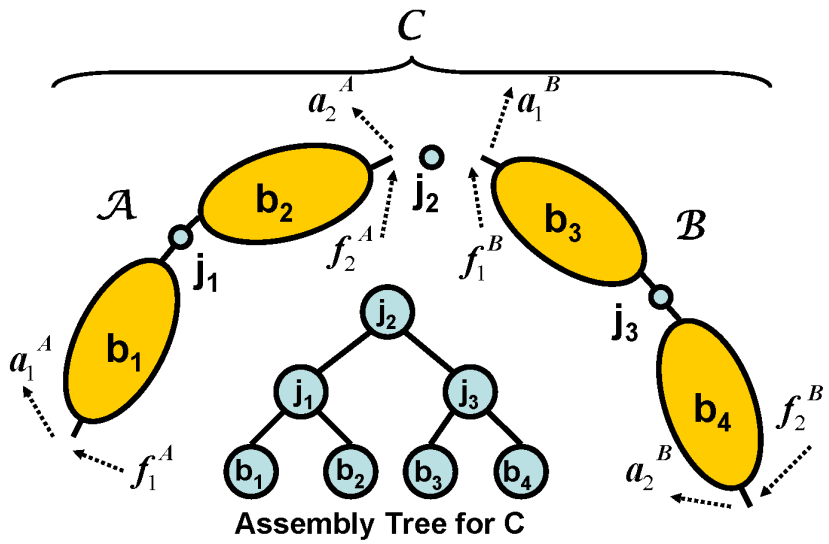

Figure 5.1: Construction of an articulated body: An articulated body $\mathcal{A}$ is connected to body $\mathcal{B}$ at the principal joint, j_2, to form body $\mathcal{C}$. The assembly tree for $\mathcal{C}$ is shown beneath the body. Forces and accelerations which govern $\mathcal{C}$'s motion are shown.

For the forward dynamics problem, our goal is to compute the joint accelerations, $\ddot{\mathbf{q}}$, given the state of the body and any external forces acting upon the body. This information is used to advance the simulation in time. The spatial acceleration of the body is given by:

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

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 (i.e. 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.

This system of equations is expressed using spatial notation. The underlying notation

is essentially an aggregate of linear and angular components of the physical quantity. For example, if a is the linear acceleration of a body, and α is the angular acceleration, then its spatial acceleration is $\hat{a} = \begin{bmatrix} \alpha \\ a \end{bmatrix}$. Spatial algebra simplifies some of the notation, as compared to the 3D vector formulation of these quantities. A detailed explanation of spatial notation and algebra is given in [Fea87].

With these motion equations, standard numerical integration techniques such as Euler's method or various Runge-Kutta schemes can be used to determine the next state of the system. For complete dynamics simulation, it is necessary to determine how the body responds to interactions with the environment. Standard techniques use analytical constraints and impulse-based dynamics [Mir96, Kok04].

We can restate the path planning problem for articulated bodies as: Find a sequential set of robot configurations $\mathbf{q}(t_0), .., \mathbf{q}(t_1)$ such that no $\mathbf{q}(t_i)$ intersects any obstacle in $\mathbf{O}$ and $\mathbf{q}(t_i)$ satisfies the non-penetration, kinematic, and articulated dynamics constraints, where $\mathbf{q}(t_0)$ is the initial configuration of the robot, and $\mathbf{q}(t_1)$ is the final configuration.

5.2.2 Extension to articulated agents

Our intended application of this work is an articulated chain, where actuators may exist to control the internal joints. However, the ideas presented in this paper are also applicable to rigid bodies and branching articulated structures.

Multi-body systems and forward dynamics are central to simulation of an articulated robot. We make use of a simplified articulated model, much like that proposed by [RGL05], in order to reduce the dimensionality of the configuration space. An impulse-based contact determination and response algorithm for adaptive forward dynamics was described by [GLM06]. Other physical constraints on the joints can be efficiently simulated through *analytical constraints* [Kok04]. These are the necessary components to ensure realistic samples for our algorithm. Due to space limitations, we refer the reader

to a recent survey on multi-body simulation [FO00] for more details. The following items are the main components in our physics-based sampling approach applied to articulated bodies.

1. **Guiding Path**: Just like the deformable body case, we use the medial axis of the workspace as our roadmap. Generally, this path will guide the end effector or some other single point on the articulated agent.

2. **Constraints**: As before, the goal of the constraints is to adjust the path to satisfy hard constraints during path execution and to guide the sampling rather than to predict trajectories. Several hard and soft constraints are used to guide the sampling. With articulated bodies, additional hard constraints include kinematic constraints and joint limitations. Similar to other cases, a path following and collision response are needed as soft constraints to help guide the robot along the path it is following. With these constraints, the agent will move toward a final configuration while considering the agent's kinematic and dynamics constraints (via the articulated body equation and its related collision handling).

3. **State-space reduction**: In general, highly articulated bodies includes hundreds or thousands of joints. This leads to a very high degree of freedom problem which is embedded in a complex state space. However, with articulated bodies we can exploit the fact that at many instances, joints tend to move very little or not at all. We can freeze or otherwise restrict the motion of various joints dynamically while allowing others to be "active" or more freely to create a reduced dimensionality problem. Dynamics simulation on such bodies has been shown to be very efficient with up to a two orders of magnitude improvement compared to a full articulated body. The run-time complexity is linear in the number of active joints, which is often much less than the total number of degrees of freedom.

   with often a sub-linear run-time complexity with respect to the number of DoFs

in the entire body.

4. **Collision handling**: For articulate bodies, collision detection and handling are complex tasks and are often computational bottlenecks in simulation. We use the graphics hardware optimized approach as detailed in 4.5 to accelerate collision queries. Finally, we develop a novel collision handling scheme to work well with our reduced dimensionality articulated bodies.

5.2.3 Articulated Chain Planning

Several ideas motivate our approach of using dynamics simulation to improve sampling and path generation.

Randomized planning algorithms such as PRMs and RRTs have recently been popular due to their success in a wide range of applications and in high-dimensional configuration spaces, or C-spaces. Probabilistic Roadmaps (PRMs) explore a configuration space by randomly placing milestones throughout it. Then, it determines which milestones connect to each other and uses this information for a roadmap. The advantage of this approach is that roadmaps can often be reused to solve planning problems with varying start and goal configurations on the same environment.

On the other hand, Rapidly-exploring Random Trees (RRTs) grow a tree in order to explore the space. Since the growth originates from a root, the search is more oriented towards regions of space near the root and thus are not as useful when a different start or goal configuration is used. On the other hand, due to the RRT construction algorithm, growth of the tree is biased toward unexplored regions of space. This makes it an appealing choice for a number of applications, but it also suffers from less optimal paths.

For highly articulated robots, the performance of these planners is dictated by the dimensionality of the configuration space. It would require a large number of samples are needed to adequately cover or explore the space. While the resulting path would

be valid, the motion would not look "natural" since randomized planning does not take the chain dynamics into account.

5.2.4 Planning in state-space

We use kinodynamic planning which generates realistic motions and paths. Specifically, we incorporate both the kinematic and dynamics constraints directly into planning; by working in a *state space*, or *phase space*. This state space maintains the physical state of the robot, often its configuration augmented by trajectory information. However, the kinodynamic planning problem is often considered much more difficult since state spaces are typically at least twice the dimensionality of configuration spaces. For each element in configuration space, at least an associated velocity is typically required. Like configuration-space planners, the complexity for the problem lies mainly in the dimensionality of this space.

Due to the dimensionality of the state-space of a highly articulated robot, these algorithms are likely to take an extremely long time to find a solution. Like the $\mathcal{C}$-space planners, these do not scale well to extremely high dimensional state-spaces. In fact, applications of these algorithms typically involves robots with very few DoFs, typically under 15-20 [LK99]. Since this model takes trajectories into account, we're like to have a path with "natural" looking motion.

In the remainder of this chapter, we describe the various components along with the final planning approach. We conclude with results and analysis by using physics-based sampling with a reduced state-space simulation core.

5.3 Articulated body simulation

In this section, we give a brief background on techniques used to simulate articulated-body dynamics as related to this work. These include Featherstone's original algorithm

and hybrid techniques.

5.3.1 Articulated Body Method (ABM)

In Featherstone's original method [Fea87], links and joints are numbered from 1 to n (for n total links and joints) such that joint h connects link h to its parent link $h-1$. The original formulation was proposed for serial linkages with single-DOF joints and a fixed base and it can be easily extended to more general situations. Branching or looping structures are constructed as long as $i < j$, where i is the parent of link j. Multiple DOF joints are simulated by placing several joints (each with a single DOF) between the rigid bodies. Similarly, a floating base can be represented by associating six DOFs with the base: three revolute and three prismatic.

Given this representation, the ABM algorithm computes all the joint accelerations, given the current state and any external forces or torques acting upon it. The overall algorithm proceeds in four steps.

1. Velocity computation: This step is performed in bottom-up manner and is used to determine the velocity of each link.

2. Initialize inertias: This step initializes the articulated body inertial tensors and the internal articulation forces in a top down manner.

3. Inertia computation: This step also proceeds in a top down manner and computes the articulated inertias and bias accelerations.

4. Acceleration computation: This last step computes both the joint and spatial accelerations in a bottom-up manner.

The first step iterates from the end of the body to the front and computes the velocity of each link. The second step proceeds in the other direction (i.e. from the tip of the link to the base), and initializes the articulated body inertia as well as internal articulation

forces associated with each link. The third step also proceeds from the tip to the base and computes the articulated inertia along with the articulated zero acceleration force at each link. Finally, the fourth pass proceeds from the base to the tip, and computes the joint accelerations and spatial accelerations for each link.

Mirtich's Hybrid Response

Contact handling has been one of the core difficulties and challenges with multi-body simulations. Impulse-based dynamics provides a local solution to the contact handling problem for articulated bodies [Mir96]. Briefly, collisions are resolved by applying impulses at the contact point. The 'hybrid' in this computation refers to combining kinematic joint constraints with impulses. One advantage of the method is that friction can be easily incorporated in this formulation.

Impulse-based dynamics computes the response based upon three assumptions, which provide a unified framework for resting and separating contact of rigid bodies. Mirtich generalized this framework to articulated models. Based upon the ABM formulation, the equations of motion for a robot in collision at an end effector is given by,

$$\ddot{q}(t) = H^{-1}(q(t))[Q(t) - C(q(t), \dot{q}(t))\dot{q(t)} - G(q(t)) + J^T(q(t))\hat{f}(t),$$

where H is the joint-space inertia matrix, C describes the Coriolis forces in matrix form, G describes external forces such as gravity, J is the Jacobian of the end effector, $\hat{f}$ is the external spatial force applied to the end effector, and Q is a vector of the magnitudes of forces and torques being applied at the joint actuators. Collisions at other points can be computed by aggregating the forces at that location, as well as Jacobian computation at the contact location.

Given this formulation, a 3×3 collision matrix K is defined that contains the dynamics information to locally process a collision. Determining K for a contact point requires applying test forces to a body and measuring the response. Since this is based

upon the ABM method, it has $O(n)$ complexity. One major drawback of this method is that the response time can be very high for a large number of links or when there are a large number of contacts.

In section 5.4, we generalize these ideas to create a response algorithm for adaptive articulated bodies.

5.3.2 Divide-And-Conquer Articulated Bodies

Our approach is based on Featherstone's Divide-And-Conquer (DCA) algorithm [Fea99a, Fea99b]. Rather than considering an articulated body to be simply a set of joints and bodies, the DCA method defines an articulated body as a recursive construction of articulated bodies connected by joints. This construction forms a tree structure which is commonly referred to as the *assembly tree*. The leaf nodes are the rigid bodies and all the remaining nodes represent joints. The root node represents the primary joint through which entire rigid body interaction is modeled (see Fig. 5.1).

Featherstone showed that the articulated body equations for a larger body C could be defined in terms of two children bodies A and B connected by a joint. In particular, if the motion equations for A and B are,

$$a_1^A = \Phi_1^A f_1^A + \Phi_{12}^A f_2^A + b_1^A; \qquad a_2^A = \Phi_{21}^A f_1^A + \Phi_{12}^A f_2^A + b_2^A,$$

$$a_1^B = \Phi_1^B f_1^B + \Phi_{12}^B f_2^B + b_1^B; \qquad a_2^B = \Phi_{21}^B f_1^B + \Phi_{12}^B f_2^B + b_2^B.$$

Then the equations for C are given as,

$$a_1^C = \Phi_1^C f_1^C + \Phi_{12}^C f_2^C + b_1^C; \qquad a_2^C = \Phi_{21}^C f_1^C + \Phi_{12}^C f_2^C + b_2^C,$$

where

$$\Phi_1^C = \Phi_1^A - \Phi_{12}^A W \Phi_{21}^A; \qquad \Phi_2^C = \Phi_2^B - \Phi_{21}^B W \Phi_{12}^B,$$

$$\Phi_{21}^C = \Phi_{21}^B W \Phi_{21}^A = (\Phi_{12}^C)^T; \quad b_1^C = b_1^A - \Phi_{12}^A \gamma; \quad b_2^C = b_2^A - \Phi_{21}^B \gamma,$$

and

$$W = V - VS(S^T VS)^{-1} S^T V; \qquad \gamma = W\beta + VS(S^T VS)^{-1} Q,$$

$$V = (\Phi_1^B + \Phi_2^A)^{-1}; \qquad \beta = b_2^A - b_1^B + \dot{S}\dot{q},$$

and S is the joint's motion subspace.

The motion subspace can be thought of as a $6 \times d_j$ matrix that maps joint velocities to a spatial vector, where d_j is a degree of freedom of the joint. For example, for a 1-DOF revolute joint that rotates along the z-axis, $S = [001000]^T$. At the leaf nodes, the coefficients are

$$\Phi_1 = \Phi_2 = \Phi_{12} = \Phi_{21} = I^{-1}; \qquad b_1 = b_2 = I^{-1}(f_k - v \times Iv),$$

where f_k is an aggregation of external forces, independent of acceleration, acting upon the link. This can include springs, forces fields, or gravity.

The DCA dynamics algorithm consists of four steps. The first step starts at the leaves and works up to the root while the second pass starts at the root and works to the leaves. These two passes simply update the position and velocity information for each joint and body.

The third pass, called the *main* pass, starts at the leaves and works up. This pass is used to compute the motion equations. It first computes the coefficients for the nodes, and progressively moves up the tree. Once the coefficients have been updated for two children nodes, the parent is also updated. Note that this independence from nodes other than its children also enables DCA to be easily and effectively run in parallel.

The final pass, called the *back-substitution* pass, starts from the root and proceeds to the leaves. This pass computes and updates the joint accelerations at each joint as well as the constraint forces to enforce articulation.

For more details of each the steps, we refer readers to [Fea99a, Fea99b].

5.3.3 Adaptive Articulated Body Dynamics

Redon et al. [RGL05] exploit the structure of the assembly tree and adaptively compute forward dynamics in the DCA formulation based on motion error metrics. Specifically, the assembly tree is replaced by a *hybrid tree*. In a hybrid tree, joints are allowed to be either active or inactive. An *active* joint is a joint that is simulated while an *inactive* joint is not simulated, and is also referred to as *rigidified*. We label nodes in a hybrid tree to be either rigid, in which case the node and its children have been rigidified, or hybrid, when the primary joint is active but a subset of children nodes are rigid. As a result, a fully articulated body is the one in which all nodes are active, whereas a body in which all nodes are inactive behaves like a rigid structure. We refer to an articulated body that contains some rigidified nodes as a *hybrid body*.

The motion equations for a hybrid body are similar to that used by the DCA method. The primary difference is that if a joint is inactive, its motion subspace is replaced by $\mathbf{0}$. This converts the joint into a rigid body, and its motion coefficients are reduced to:

$$\Phi_1^C = \Phi_1^A - \Phi_{12}^A(\Phi_1^B + \Phi_2^A)^{-1}\Phi_{21}^A,$$

$$\Phi_2^C = \Phi_2^B - \Phi_{21}^B(\Phi_1^B + \Phi_2^A)^{-1}\Phi_{12}^B,$$

$$\Phi_{21}^C = (\Phi_{12}^C)^T = \Phi_{21}^B(\Phi_1^B + \Phi_2^A)^{-1}\Phi_{12}^A,$$

and

$$b_1^C = b_1^A - \Phi_{12}^A(\Phi_1^B + \Phi_2^A)^{-1}(b_2^A - b_1^B),$$

$$b_2^C = b_2^B - \Phi_{21}^B(\Phi_1^B + \Phi_2^A)^{-1}(b_2^A - b_1^B).$$

The simulation of a hybrid body is similar to the DCA algorithm. Assuming that

the set of active nodes is known, the algorithm defines a passive front. The nodes on the passive front are those that have been rigidified, but have an active parent. These nodes serve as the base case, or leaf nodes, for our adaptive algorithm.

The bias accelerations are computed by making a pass from nodes in the passive front to the root. The motion equations vary based on the location of each node. Next, a top-down pass is performed to compute the acceleration of active joints. Then, the velocities and positions of the active joints are updated based upon the accelerations. Finally, the inverse inertias are updated. Redon et al. [RGL05] show that these do not need to be updated for rigidified nodes. Since each step in this dynamics algorithm only operates on the active region, the time complexity for a step is $O(d_n)$, where d_n is the number of active joints.

Since this algorithm only simulates a subset of the nodes, it can result in approximation errors in the motion of the articulated body (as compared to the original DCA algorithm). In order to quantify this error, we utilize the motion error metrics [RGL05], the *acceleration metric value* and the *velocity metric value*:

$$\mathcal{A}(C) = \sum_{i \in C} \ddot{q}_i^T A_i \ddot{q}_i,$$

$$\mathcal{V}(C) = \sum_{i \in C} \dot{q}_i^T V_i \dot{q}_i,$$

where A_i and V_i are symmetric, positive, definite $d_{J_i} \times d_{J_i}$ matrices, and d_{J_i} is the number of degrees of freedom of joint i in C. The acceleration metric can be computed without explicitly computing the accelerations of the joints in C.

One of the main components of the algorithm is the computation of the active region. This can be done in two ways: with an user defined error threshold, $error_{max}$ or an user defined number of active joints, d_n. Note that this does not need to be done at every time step, but may add additional error to the system.

For the error threshold, we maintain a current error as $error_c$. Initially, we set

$error_c = \mathcal{A}(C)$, where C is the primary joint of the root node. This represents the total acceleration metric value for the body. If $error_c > error_{max}$, we compute the acceleration and forces for the current joint, remove its contribution, $\ddot{q}^T A \ddot{q}$, from $error_c$, compute the acceleration metric value for its children, and add these into a priority queue. Body A has a higher priority than a body B when $\mathcal{A}(C) > \mathcal{B}(C)$. We proceed along the nodes with the highest priority, until we satisfy our error bounds.

For the specified number of active joints, we proceed in a similar fashion. Initially, we add the primary joint of the root node to the priority queue. For each joint at the front of the queue, we remove it, compute its joint acceleration, the metric value of its children, and add the children to the queue. We continue this process until we have simulated, or computed the acceleration and forces, for d_n joints.

The simulated joints from this step make up the *transient active region*. We then update the velocities and velocity metric for this region. From this information, we determine the new active region once again based upon the velocity metric in a similar manner to that above. Additionally, there may be nodes in the transient region that do not need to be simulated. Impulses are applied to the body to zero out these forces. For more information, we refer the reader to [RGL05].

5.4 Efficient Collision Handling for Adaptive Articulated Bodies

We address the problem of handling complex interactions between articulated rigid bodies with a high number of DOFs. Our solution incorporates static or dynamic friction states, resting states, as well as separating contacts. Many different methods have been proposed to detect such interactions and handle them robustly. Some of the commonly used approaches include constraint-based, penalty-based, impulsed-based methods, or via analytical constraints.

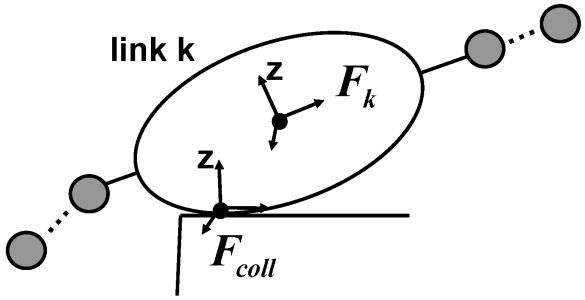

Figure 5.2: Collision Frame: The collision frame, F_{coll} is determine by the point of contact and the contact normal. By convention, the z-direction of the collision frame is in the direction of the contact normal. Forces upon link k must be transformed from the F_{coll} to F_k, the link's inertial frame.

In this section, we present our adaptive contact handling algorithm. This is based on using a hybrid hierarchy for collision detection as well as Mirtich's impulse-based dynamics algorithm for response computation. We also present a formulation of the hybrid Jacobian and show how it can be used for contact handling in sub-linear time, or linear with respect to the number of active joints (which is typically much less than the total number of joints).

5.4.1 Adaptive Impulse-Based dynamics

As shown in [Mir96], proper contact handling involves computing the *collision matrix* K for rigid as well as articulated models. Given the collision matrix, collision integration can be performed by finding a spatial impulse that is applied to the colliding link. This spatial impulse causes a change in joint velocities that must be propagated through the articulated body. In more detail, the following steps are performed to compute the response to each contact:

1. Update the articulated inertias of the bodies.

2. Apply a test impulse to the colliding link.

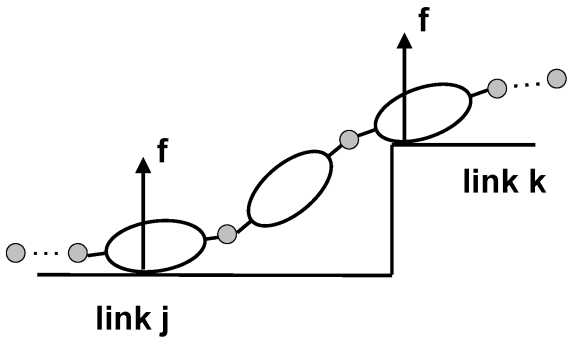

Figure 5.3: Multiple simultaneous collisions: Each collision imparts a large, external force onto the associated link. We present an efficient algorithm to compute collision response. The complexity of our algorithm is linear in the number of contacts and sub-linear in the number of DOFs.

3. Compute the impulse response of the body along the path from the colliding link to the base. This involves one pass through the links of the articulated body from the colliding link to the base, and another pass from the base to the colliding link.

4. Compute the collision matrix K by applying test impulses of known magnitude and measuring their effects.

5. Use K and collision integration to compute the collision impulse.

6. Propagate the collision impulse to the base.

7. Propagate the resulting changes in velocity throughout the body.

The primary challenge here is to adapt these steps in the framework of the adaptive dynamics (AD) algorithm. Most of the changes are algorithmic. The resulting algorithm would compute the same impulses as when Mirtich's algorithm is applied to an articulated body that has the same structure and rigid links as that of the hybrid body. The primary equations used to compute Mirtich's response, such as those necessary to propagate the impulse or the components of K, do not change. Instead, the amount of computation is reduced by utilizing the adaptive dynamics framework. Any change to

the equations, for instance the dimension of the Jacobian or of the joint acceleration and velocity vectors, vary with the number of active joints. With this in mind, we describe the algorithmic changes to Mirtich's collision response for articulated bodies.

If the inertia computation can be reused, the first step is performed during the forward dynamics calculations. Unfortunately, this is not the case for either DCA nor AD. Instead, this can be done by visiting the resulting rigidified sub-assemblies and computing their inertias. This essentially traverses the leaf nodes of the active portion of the subtree, and can be viewed as an in-order traversal of nodes on the active front. Since only active nodes are being visited, this step takes time linear to the number of active joints, rather than the total number of joints.

The next two steps are repeated for three passes for each articulated body involved in the contact, one for each of the standard basis directions. The relationship between the applied contact impulse and the change in velocity is linear (since K is constant for a given contact), which allows the response to be measured through the application of known test impulses. Each pass applies a unit test impulse in a basis direction. Application of test impulse requires evaluation of joint accelerations, which we have shown to be sub-linear by our adaptive method.

Once a test impulse is applied, the impulse response is measured. For a fully-articulated body, this operation has a worst-case run time of $O(n)$. By exploiting the adaptive front of the hybrid tree, we test the impulse response of the hybrid body, effectively treating each rigidified section as a link or a rigid body. Then, traversal to the base link is equivalent to following the adaptive front in the hybrid tree. For an adaptive serial linkage built as a balanced tree, this is a subset of the nodes along the passive front. In particular, we iterate through nodes along the passive front that are seen prior to the colliding link in an *in-order traversal* of the hybrid tree. During this step there is no need to traverse deeper than a node along the passive front. By following this front, the computation is reduced to iterating over only a subset of active joints, again

achieving a sub-linear run time for this step.

One more detail needs to be noticed here. Spatial velocities need to be tracked throughout the rigidified links. However, AD (and DCA) maintain state only through the joint positions and velocities. We rely upon the Jacobian to map joint velocities into spatial velocities. Normal analytical solutions for the Jacobian would require iterating through all the bodies and joints. Instead, we introduce an *hybrid-body Jacobian* which treats the body as having fewer active links with some joins rigidified. From this formulation, our Jacobian computation only needs to iterate through the active joints and we maintain the desired sub-linear run time for this step as well. The computation of a hybrid-body Jacobian is explained later.

Steps 4 and 5 both involve computations that are independent of the number of joints or bodies, and cannot be accelerated by the adaptive structure. Once the collision impulse, $\hat{p}_{coll}$, has been applied to the body, it must be propagated through the nodes. The first step passes $\hat{p}_{coll}$ again from the link to the base node. This is performed in the same manner as in measuring impulse response.

The final step requires propagating the change of velocities due to $\hat{p}_{coll}$ through the body. This step is accomplished through one bottom-up pass to update the joint velocity of the root, followed by a top-down pass to pass it to the remaining bodies. This step only has to be performed for nodes in the update region, achieving a sub-linear runtime. At this point, the contact has been processed and its effects have been propagated through the articulated body.

Note that this method only solves for translational contact and collision. The spatial collision impulse does not contain any torque. Thus, impacts related to joint limits or friction between joints cannot by simulated by impulse-based dynamics in this formulation. Other methods, such as analytical constraints, can effectively simulate the dynamics of these events. Next, we explain how to incorporate analytical constraints for events that require a response torque.

5.4.2 Analytical Constraints

Adaptive dynamics fits very nicely into this type of framework for response. Based upon the standard articulated body equation, Kokkevis showed that at a given time instant, there exists a linear relationship between the magnitude of the joint accelerations and the magnitude of all forces, both internal and external, acting upon the body [Kok04]. This relationship can be expressed as

$$\ddot{q}^f = kf + \ddot{q}^0,$$

where $\ddot{q}^0$ and $\ddot{q}^f$ are the joint accelerations before and after some force is applied. A proof of this relationship is given in [Kok04]. Contact, collisions, as well as other joint constraints, can all be posed in this manner. We focus on its use for joint constraints.

There are a couple of considerations when using this technique with AD. First, the motion equation is based on slightly different sets of equations than as compared to what AD is based upon. Second, this derivation leads to a computation model which is linear in the number of joints. Moreover, we can only interact with the articulated body at the handles. Thus, external forces must be transformed into equivalent forces at one or more handles, resulting in less efficient computations.

The first issue can be easily resolved. The matrix form of motion equations is computationally equivalent to Featherstone's ABM method. The second issue requires small modifications to the analytical constraints algorithm. In proving the relationship between forces and accelerations, one step requires converting all external forces acting upon the body to a generalized force and torque. This conversion is normally done through multiplication by the Jacobian at the point of contact or manipulation. Analytically computing this Jacobian is an $O(n)$ operation. Instead, we again base our response using our "hybrid-body Jacobian."

Given the hybrid tree, a "hybrid body Jacobian" can be computed or closely approx-

imated in a number of ways. The key concept is that we only need to recurse through the active joints, reducing the Jacobian computation to an $O(d_n)$ operation. We will describe this step in more detail in the next section.

It follows that for any linear function h of joint accelerations $\ddot{\mathbf{q}}$, there exists a k such that:

$$h(\ddot{\mathbf{q}}^f) - h(\ddot{\mathbf{q}}^o) = kf_\imath.$$

Therefore, by evaluating the linear constraint function h before and after the application of some known (non-zero) force $f_\imath$, we can solve for k and use that to compute the force which must be applied to obtain a given value of $h(\ddot{q})$. For the case of joint constraints or impacts, we then apply the resulting torques to the specified joint or handle.

In many cases, several constraint equations will have to be solved simultaneously in order to ensure that the solution from one constraint does not have an adverse effect on another constraint. For instance, this occurs when multiple joints are exceeding their positional limits. This method can be generalized into solving a system of p constraint equations:

$$\mathbf{h} - \mathbf{h}^o = \mathbf{K}\mathbf{f},$$

where $\mathbf{h}$ and $\mathbf{h}^o$ are p-dimensional linear functions of one or more joint accelerations, $\mathbf{K}$ is an $p \times p$ matrix with the k_{ij} values relating constraint i and j, and $\mathbf{f}$ is a $p \times 1$ vector of constraint forces.

One advantage of this approach is that it is fairly simple to use once the articulated-body implementation is in place, since it only needs to set forces and retrieve the resulting body and joint accelerations. This is similar to the application of test impulses in the previous section. Since each update is a $O(d_n)$ operation for d_n bodies, solving for p constraints takes times $O(d_n p + p^3)$, since solving the linear system is a $O(p^3)$ computation in the worst case. In our tests so far, p is typically much smaller than d_n.

One of the important applications of acceleration-based constraints in our algorithm

is to resolve joint limit violation events. A joint reaching a limit can easily be detected by comparing the current joint position to some known limit.

Kokkevis [Kok04] gives a fairly complete method for resolving these constraints, but we suggest ways to simplify the physical simulation in order to improve the overall computation. The goal of these constraints is to cause an instantaneous change in the velocity through the application of an impulse to ensure that the joint does not move further in its current direction. We also need to ensure that it is not accelerating in this direction.

The application of an impulse can be considered as applying a large force over a very short time, the compression and restitution phase of a collision. We can choose a desired resulting joint velocity (usually zero) to result from a joint impact. Given this, we can build an acceleration constraint by approximating the desired acceleration as:

$$\ddot{\mathbf{q}}^I = \frac{\dot{\mathbf{q}}^+ - \dot{\mathbf{q}}^-}{\delta t},$$

where $\ddot{\mathbf{q}}^I$ is the joint acceleration during δt, and $\dot{\mathbf{q}}^+$, and $\dot{\mathbf{q}}^-$ are the joint velocity before and after the impact. This joint acceleration could then be used with the constraint equation $h(\ddot{\mathbf{q}}) = a_i(\ddot{\mathbf{q}}) - a_i^I$ to solve for the force that will enforce the constraint.

The process for resolving joint limits is very similar. An impact constraint can be used to compute an impulse that will cause the velocity of the joint breaking its limit to be zero. A joint acceleration constraint can be used to ensure that inter-penetration will not occur in later steps. Again, since the process for joint velocities has been formulated as an acceleration constraint, we maintain the same complexity as that portion. Thus, resolving joint impacts fits within our sub-linear run-time resolution scheme.

5.5 Physics-based Sampling and Path Computation

In this section, we tie the components together to create a physics-based sampling strategy in the case of highly articulated robots. As previously mentioned, we exploit the coherence between joint angles via the adaptive forward dynamics framework. Joints are first prioritized based upon well-defined motion metrics. Then, only the most important joints are simulated which effectively reduces the dimensionality of the search space. Potential bias forces are applied to encourage movement towards a goal.

Hard constraints such as non-penetration and environment bounds are handled through the use of accelerated collision detection and impulse-based collision response for hybrid articulated bodies. It remains to define additional constraints used to direct or bias the sampling direction.

5.5.1 Additional constraints

To encourage the system toward a goal, we generate several types of constraint forces and torques as biasing forces. This has the similar effect of a potential-field planner guiding a robot towards a goal configuration. To avoid problems of local minima in the "potential" well, additional forces are introduced to ensure that the robot makes progress toward its goal.

The first force is a repulsive force that helps to keep the robot away from obstacles. If contact occurs with the environment, impulse-based responses are applied to ensure no penetration with obstacles. We can create simple limited radius repulsive force based upon the distance to the link. For each link, we apply a force

$$force_{ob_i}(b_j) = \begin{cases} -(\frac{\delta^2}{d(b_j,ob_i)^2} - 1)\hat{\mathbf{d}} & \text{if } d(b_j, ob_i) < \delta \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where δ is a distance threshold, $d(b_j, ob_i)$ is the distance between rigid link b_j and

obstacle ob_i, and $\hat{\mathbf{d}}$ is the direction from b_j to ob_i.

The second is an attraction force to the goal configuration. This force can be applied to the entire body, a subset of links or joints, or alternatively to a single node or joint such as the end effector:

$$force_{goal}(\mathbf{q}) = w_{goal}\frac{(\mathbf{q}_{goal} - \mathbf{q})}{\|\mathbf{q}_{goal} - \mathbf{q}\|}$$

where w_{goal} is a goal node weight, $\mathbf{q}_{goal}$ is the goal configuration.

To escape local minima, we use a third *path-following force* at the cost of generating a path for a point robot in the workspace. This methodology works especially well in situations where the end effector needs to reach a point, and the configuration of the remainder of the robot is less important as long as it is valid. The path following force acts much like the goal attraction force, but instead uses a configuration along the path on the way to the goal.

The fourth is a torque rather than a force. It encourages the robot toward the desired goal configuration. This control is implemented as a linear torsion spring acting on a joint:

$$torque_{j_i} = -k_s(q_i - q_{goal}) - k_d\dot{q}_i$$

where k_s is the angular spring coefficient and k_d is an angular sprint damping coefficient.

5.5.2 Motion generation

Input: Robot $\mathcal{R}$, Starting configuration q_{start}, Goal configuration q_{goal}, obstacles $o_1, \ldots, o_n$

Output: A path of robot states, $\mathcal{S}$

$\mathcal{R}^i.q \leftarrow q_{start}$;

while *d($\mathcal{R}^i.q$, q_{goal}) > GoalReachedThreshold* **do**

Add $\mathcal{R}^i$ to $\mathcal{S}$;

Apply potential bias forces to $\mathcal{R}^i$;

`/* Generate next sample */`

$\mathcal{R}^{i+1} \leftarrow$ AdaptiveDynamicsSolver($\mathcal{R}^i$);

`/* Prioritization of Joints */`

remainingMotion $\leftarrow \mathcal{A}(\mathcal{R}^{i+1}.treeRootNode)$;

PriorityQueue $\mathcal{P}$.enqueue($\mathcal{R}^{i+1}$.treeRootNode);

nodeCount = 0;

while *remainingMotion > motionThreshold* **and** *nodeCount < numActiveNodes* **do**

$n \leftarrow \mathcal{P}$.front;

Mark n to be simulated;

$\mathcal{P}$.enqueue(n.leftchild);

$\mathcal{P}$.enqueue(n.rightchild);

remainingMotion $\leftarrow \Sigma_{n_i \in \mathcal{P}}\ \mathcal{A}(n_i)$;

nodeCount $\leftarrow$ nodeCount +1;

end

$\mathcal{R}^i \leftarrow \mathcal{R}^{i+1}$;

end

Algorithm 4: Physically-based sampling and path computation

The framework described here can be used in at least two ways: either as a local planner when coupled with some high-level randomized planner, or potentially as a

way to determine the entire path. Pseudocode for the sampling algorithm is given in Algorithm 4.

The method begins by first initializing the robot, $\mathcal{R}$, with the starting configuration, q_{start}. If the robot is sufficiently close to the goal configuration, q_{goal}, then the planner has reached the goal. Otherwise, we apply the potential bias forces to the robot. Next, The adaptive forward dynamics solver is used to determine the joint accelerations and perform numerical integration in order to arrive at the next sample. Finally, prioritization of the joints determines which joints should be simulated, effectively reducing the dimensionality of the problem.

This results in a path or path segment that maintains kinematics and dynamics constraints at every intermediate configuration. With the efficient forward dynamics algorithm, we are able to compute this in a relatively small amount of time. Other probabilistic planners would likely require significantly more time or do not typically take both kinematics and dynamics constraints into account.

5.6 Results and analysis

5.6.1 Collision handling results

We have implemented our collision detection and response algorithm on top of adaptive dynamics. We use three benchmarks to test the performance of our algorithm.

1. The first benchmark shown in Fig. 5.4 allows a highly articulated pendulum to swing back and forth. Along its path are various pegs with which the pendulum collides. The pendulum is modeled using 200 DOFs and we have observed 5X speedup in this simulation using our adaptive algorithm.

2. This scenario involves taking a thread-like articulated body (Fig. 5.5), and feeding it through a sequence of holes. This benchmark involves computing the collision

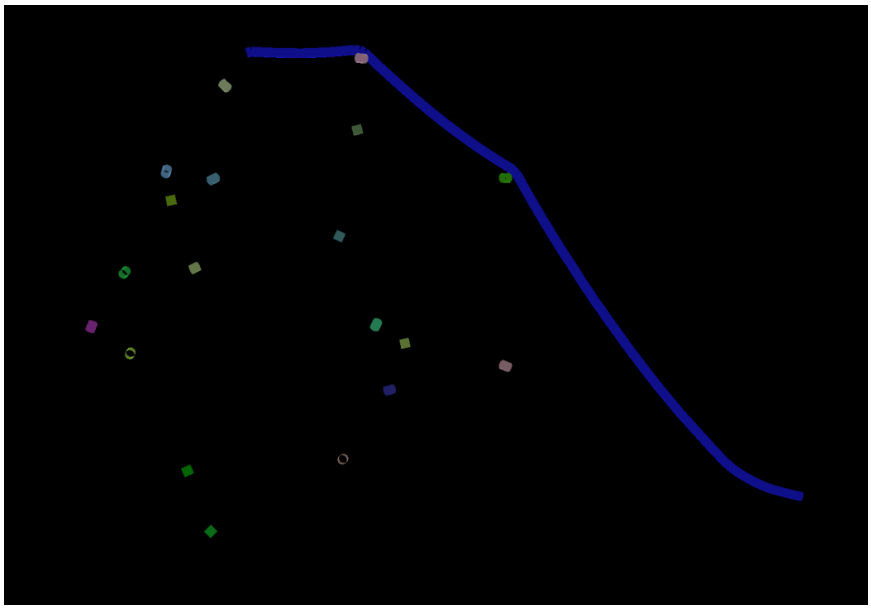

Figure 5.4: Pendulum benchmark: For testing the feasibility of our collision handling approach, in this benchmark an articulated pendulum with one end fixed to a point falls toward a set of randomly placed cylinders. The pendulum is modeled with 200 DOF. A visually accurate simulation was run with only 40 active DOFs and gives a 5X speedup in collision detection and response computation.

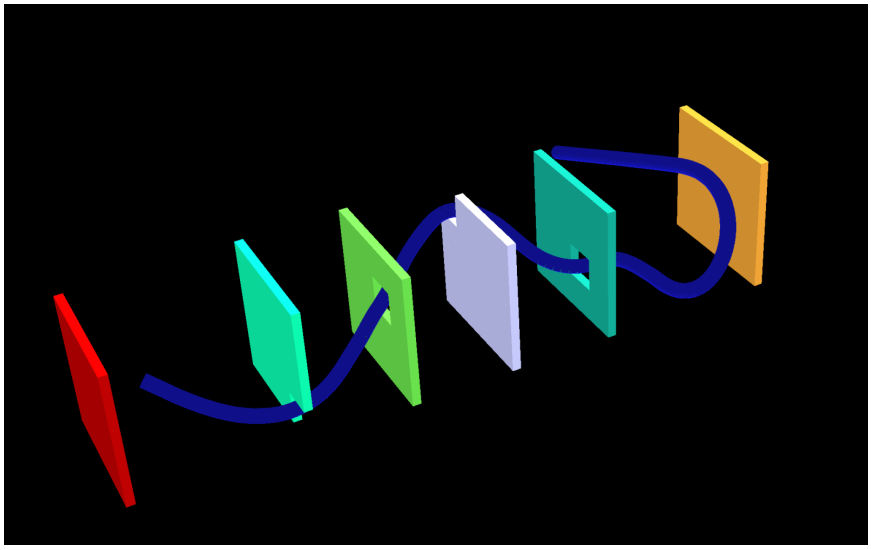

Figure 5.5: Threading benchmark: The articulated body is modeled with 300 DOFs and moves through a sequence of holes in walls. At each step of the simulation, we use only 60 DOFs for adaptive dynamics and collision response computation. Overall, our adaptive algorithm results in 5X speedup in this simulation.

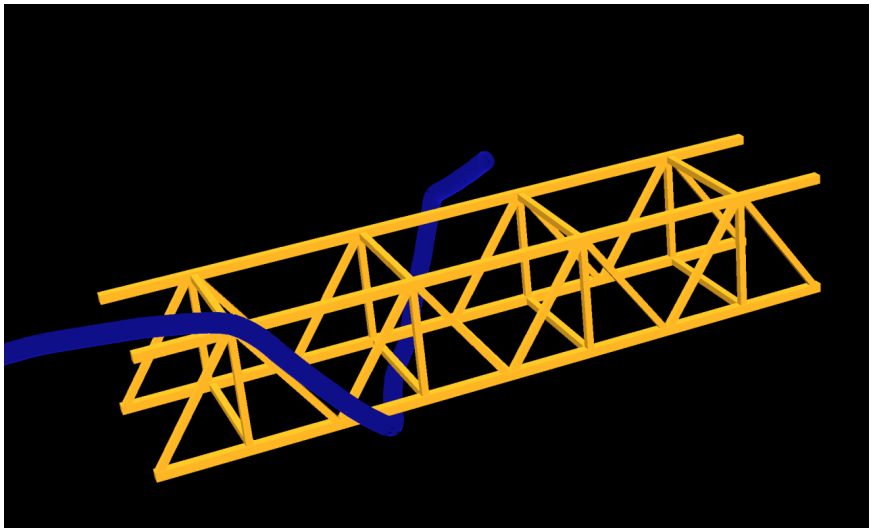

Figure 5.6: Bridge benchmark: The articulated body travels around the bridge in a snake-like motion. The body contains over 500 links and 500 DOFs (i.e. 500 DOFs). We used only 70 DOFs for adaptive dynamics and collision response, resulting in 8X improvement in the simulation.

response at the hole boundaries, but also adaptive forward dynamics computation. The articulated model contains 300 DOFs, and we observe 5X speedup in this complex simulation.

3. The bridge benchmark (Fig. 5.6) is to demonstrate the idea on snake-like robots. The articulated body wraps around the bridge as a means to travel across it. The body contains over 500 DOFs and large portions of the body are in contact with the bridge. Our adaptive simulation algorithm achieves 8X speedup in this simulation.

In our benchmarks, we observe significant improvement in the performance of collision response computation algorithm. In each benchmark we fixed the number of active joints and the performance of our algorithm is directly proportional to the number of active joints. As can be seen from the graph (Fig 7), the performance is roughly proportional to the ratio of active joints to the total number of joints. Furthermore, the absolute performance numbers are very similar for articulated bodies with the same

Figure 5.7: Collision response time vs. number of active joints: This chart shows the average time to process a collision for a fixed number of active joints. It also shows that there is a relatively linear relationship between the active joints and the running time.

number of active joints and do not vary much as a complexity of environment or the number of contacts. This result is promising since it shows bodies with any number of or DOFs can be simulated at about the same rate. However, if we use only a few DOFs, than the resulting simulation can have higher errors.

Runtime Analysis

In the modified response algorithm, each step runs in $O(d_n)$ time, where d_n is the number of active joints. From empirical results, we observe that it is sufficient to have $d_n < n$ without introducing significant error into the system for models with a high number of DOFs. For articulated bodies with a large number of DOFs, this speedup is beneficial as the hidden constants in the analysis can be fairly large. For instance, in order to compute the matrix K for impulse response, there can be as many as twelve propagations through the joints and links. This happens when two articulated bodies are colliding and each body measures the result of three test impulses and each impulse requires two passes

through the active joints. Finally, the number of these contacts depends mostly on the geometry of the environment as well as the geometry and placement of the articulated bodies.

Approximation and Collision Error

Since the original AD algorithm provides bounds on error, we wish to show that the new algorithm maintains some error bounds as well. First, note that our algorithm effectively computes the correct Mirtich-based response for the hybrid body. Each contact response computed could have a direct effect on the joint acceleration of each rigid link. This in turn also causes a change in joint velocity at each link. Since both joint acceleration and velocity are both used in the error metric, this could have an effect on the active joints.

Since the active joints are determined by a priority-based recursive evaluation of the metrics, this could mean that various resulting motions are possible with this technique. If the collision response causes a joint to have a significant acceleration or velocity, then the rigid zone update would observe it and the joint would likely be activated. On the other hand, in situations where this is not significant, perhaps when the body is at rest, or the joint velocity of a joint is much less than that elsewhere on the body, then these joints will not be activated. The resulting motion would still be valid and motion error as measured by the acceleration or velocity metric is bounded.

For best results, active region updates are performed at every timestep or at the end of every timestep that consist of a collision. However, this has a reasonably high overhead that is proportional to the number of active joints. This is especially true when the body is in constant contact with an obstacle. While asymptotically the run-time complexity is still $O(d_n)$ for these operations, the constant associated with the computation can increase considerably. In practice, it is sufficient to only apply updates about every ten to fifteen timesteps, since the long term impact of collisions do not

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

Table 5.1: Articulated body planning performance table: Planning performance breakdown across each of our benchmarks. Longer simulation times result from the robot having to travel a greater distance, whereas the average step times are relatively uniform across all environments.

disappear in a short time interval. Within this interval, changes in acceleration and velocity seem to be accurately captured without a significant hit on performance.

It should also be noted that delaying the update can also add to the amount of error. Since a reasonably good set of active joints should have been chosen prior to a collision, the motion should still be representative of the general motion. This error is essentially limited by the update interval.

We implemented this algorithm on a Dell M60 Mobile Workstation, with a 2.1GHz Pentium M processor and 1GB of main memory.

5.6.2 Planning results

We tested our algorithm on a number of benchmarks. In each case, a highly articulated robot in a serial linkage must navigate through an environment. The goal of the planning is for the end effector to reach a certain position in the workspace. The base links of the robot, including the thin rigid bodies as described in Section 5.2.2, are cylinders, each represented by 20 triangular primitives.

- **Serial Walls** - This scenario is based on a benchmark created by the Texas A&M Parasol Laboratory. A 300 joint articulated chain must travel through a sequence of walls with holes. (See Fig. 5.9(a))

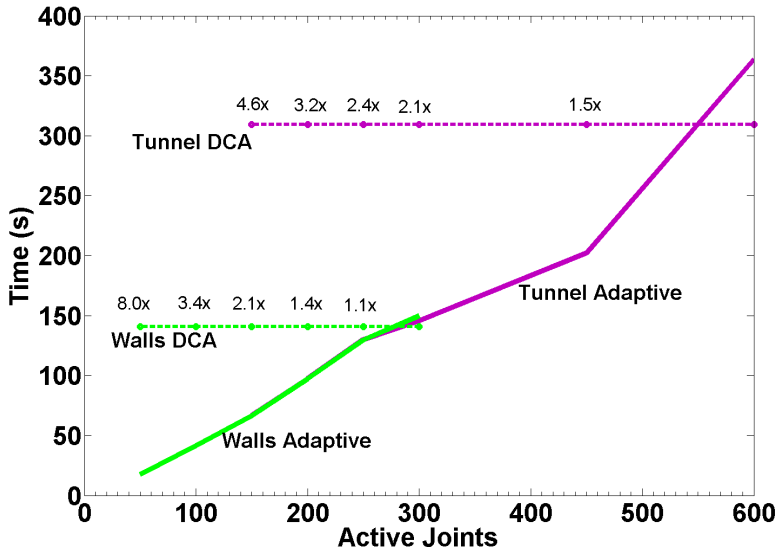

Figure 5.8: Planning time vs. Number of active joints: Relationship between active joints and planning time for the Serial Walls (300 DoF) and Tunnel benchmarks (600 DoF). The horizontal line represents the time taken by using Featherstone's DCA algorithm. The values above it are the speed-ups over the DCA algorithm when that many active joints are simulated.

- **Tunnel** - Also based on a benchmark created by the Texas A&M Parasol Laboratory, the tunnel environment requires a 600 joint articulated chain to move through a tunnel. The tunnel has two right angle bends about half way through the block. (See Fig. 5.9(b))

- **Catheterization** - This scenario is based upon a medical procedure called *liver chemoembolization*. A thin, flexible catheter, modeled as a 2500-joint articulated robot, must travel through a network of arteries. The goal is to find a tumor in the liver for cancer treatment. (See Fig. 5.9(c))

- **Pipes** - This situation represents an application of snake-like robots in a pipe inspection task. The 2000-joint robot supports itself by coiling around a pipe while searching for a leak. The coiling behavior was encoded into the goal attraction force. (See Fig. 5.9(d))

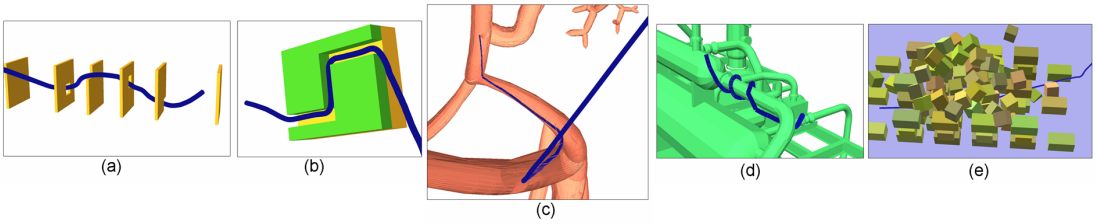

Figure 5.9: Articulated planning benchmark scenarios: (a) Serial Walls; (b) Tunnel; (c) Liver Catheterization; (d) Pipes; (e) Debris.

- **Debris** - In this case, the 2000-joint robot aids in a search-and-rescue operation. The goal is to find an opening into the debris pile and to seek a large, open pocket where a survivor or important item may be found, and then to plan a way out. (See Fig. 5.9(e))

Our sampling method was able to complete the task *without* the explicit need to perform planning in the high-dimensional configuration or state space. Planning performance and sampling time using adaptive dynamics simulation is given in Figure 5.1. We show results for a fixed number of active joints that determine the reduced dimensionality of the configuration space for the articulated body. The fourth and fifth columns give the geometric complexity of the robot and environment. Finally, the last two columns give the total planning time followed by the average time it took to generate samples en route to the goal.

As can be seen, our algorithm has very favorable performance. Aside from the complexity of the environment and robot, other factors that affect the performance include the length of the path and any velocity constraints on the robot. Longer paths or slow speeds will increase the time it takes to reach a solution since the robot only travels a finite distance for each step. We observed up to an order of magnitude speed with 10% to 15% of the joints being active. Figure 5.8 shows the speed-up over simply using Featherstone's algorithm for a fixed number of active joints.

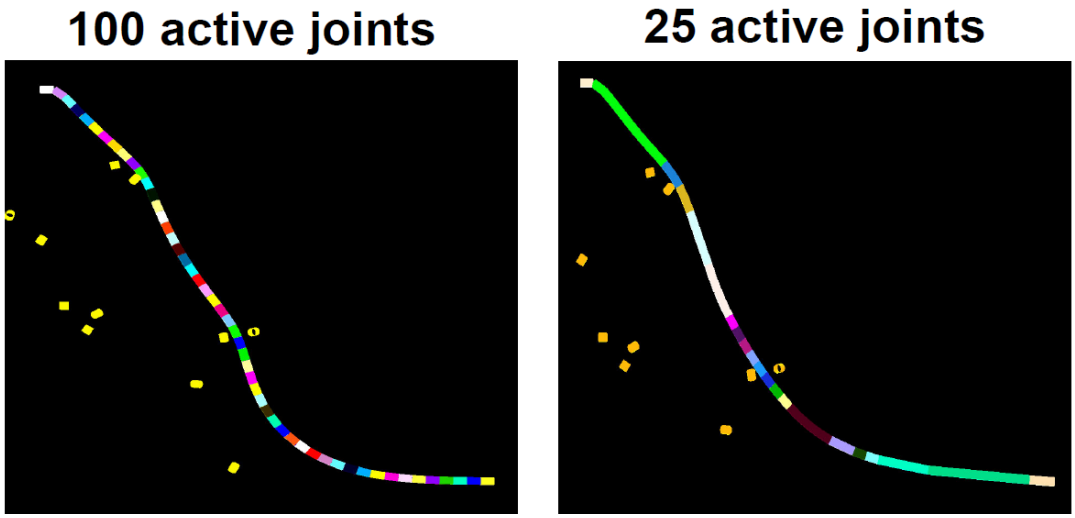

Figure 5.10: Full vs. Reduced dimensionality articulated body accuracy: Visual comparison of the simulation of an articulated with 100 joints. On the left, all 100 joints are simulated while on the left only the most important *25* of the 100 are simulated. Each segment represents a rigidified portion of the body.

Motion Deviation Analysis

The effectiveness of the state space reduction is dependent upon the allowed motion threshold. To get an idea of how the motion affects dimension reduction, we tested various threshold settings in the Serial Walls environment. We have observed that by halving the motion metric threshold, the dimensionality increases by 9% to 38%. Also, there were certain threshold values which caused a much larger change than others. This is likely due to the fact that even though we reduce motion threshold, the current motion is close enough to the actual motion that it requires a smaller threshold to capture additional motion deviations. Since the total planning time is closely related to the average dimensionality, we noted that halving the threshold increased the planning time by about 10% to 34%.

While accurate simulation is not a focus of this planning work, we like to comment briefly on the kinematic and dynamics differences that results from the reduction. First, we note that quantitatively, motion deviation metrics as defined by adaptive dynamics

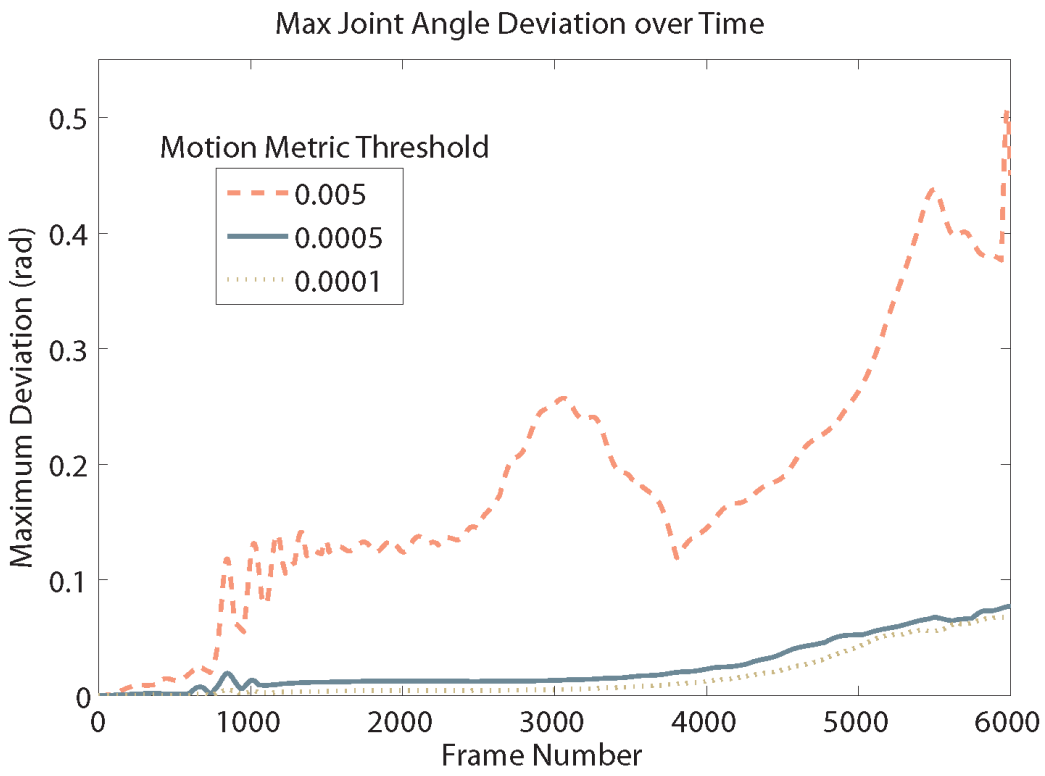

Figure 5.11: Joint angle difference over time: Each curve shows the maximum joint error during the Serial Walls benchmark for a fixed motion metric threshold.

is very difficult to interpret. Therefore, we consider the joint angle difference between adaptive dynamics and full dynamics simulation over the course of a planning situation. Fig. 5.11 shows the joint difference throughout the Serial Walls benchmark for varying amounts of motion threshold. Since the difference is bounded on a per time step basis, it has a tendency to slowly accumulate over the course of the planning. The potential forces help to reduce the total difference by ensuring that the robot follows roughly the same path regardless of the amount of simplification. This accounts for some of the sharp drops seen in the graph.

Discussion

There are several advantages and some disadvantages of this method. It is able to perform physically-based local planning in a high-DoF state space in a reasonable amount of time. Its performance makes it a favorable choice as a local planner for either PRM or RRTs to compute link queries. However, as the distance between the start and goal configuration increases, the time to find this path increases since it moves a finite distance per iteration. Additionally, it is possible that this method may fail to find a path, even if one exists. Although, in practice this has not been a problem when adequate forces are applied to the robot to escape local minima due to various constraints.

We examine the resulting trajectory generated by the various sampling criteria. Note that for a highly articulated body of 300 joints the standard PRM and RRT algorithms were not able to find a solution within a suitable period of time. Thus, the following discussion examines the trajectory if such a path was found.

- **Path Validity**

  We first mention that paths generated by this algorithm are completely valid. Contact constraints and forces ensure that no penetration occurs between the robot and obstacles. When no dimension reduction occurs, the generated solution will clearly adhere to kinematic and dynamics constraints and joint angle limitations will not be violated. When joints are rigidified, they were already in a valid state and will remain that way if become active again. Therefore, at any time instant, all the joints will be in a valid configuration and the resulting path is valid.

- **Comparison with Configuration-Space Planners**

  For randomized C-space planners, we compare our method with solutions that might be generated via PRMs or RRTs. It should be noted that in most common implementations, a robot travels between milestones along a *straight-line* path. While this will yield valid results, kinematics constraints require additional checks

for each link and dynamics constraints are not considered. So, the motion may not be smooth when transitioning between links. Thus, the quality of the resulting motion may not be as realistic, though the computed solutions may be adequate to solve the problem.

Arguably, if the space is more completely sampled by a random kinematics based method, then the milestones would be quite close. In that case, the motion generated by the resulting path would be much better. However, sufficient sampling of this space would not be practical.

In comparison, each configuration along a path generated by our method will follow a simplified kinematics and dynamics model. Thus, the resulting motion will look natural with respect to how the world is modeled and the biasing forces being applied.

- **Compared to state space planners**

  While deterministic kinodynamic solutions exist for some simple problems, many of these will not scale to the dimensionality of our benchmarks. Therefore, one of the randomized kinodynamic planners may be able to generate some solution. However, such an approach is likely to take an extraordinary amount of time and not practical. For instance, it took an RRT-based kinodynamic planner on average over 10 minutes to compute a path for just a rigid body in space (i.e. 12 DoF state space) on an 800 MHz Pentium III computer with 256 MB RAM [LK99]. While it would be much faster with current computational resources, it also was a very simple model.

  Again, in our solution the kinematics and dynamics are modeled for a simplified model. Thus, while the resulting motion may not be identical to the full kinodynamic solution, it will not be much worse, assuming that we set a sufficiently small motion threshold as can be seen in Fig. 5.11.

Chapter 6

Numerous Agents

6.1 Introduction

Multiple robots and agents are increasingly used in different applications to cooperatively solve complex tasks. These include assembly, manufacturing, swarm or distributed robotics, virtual prototyping, animation, traffic engineering, crowd simulation, etc. Given a set of rigid or articulated robots with their initial and final configurations, the goal is to compute a collision-free path for each robot as it moves among the obstacles and other obstacles.

Compared to single-robot motion or path planning, the multiple-robot problem is more challenging in terms of the theoretical complexity as well practical implementations. A variety of factors contribute to this added complexity. Most notably, in a multi-robot system each robot's degrees of freedom (DOFs) contribute to the total number of DOFs of the system. Thus, the problem complexity itself grows rapidly with the number of agents. One of the other main challenges is coordination among the different robots. In many cases, robots must work together in order carry out a task, maintain a formation, or stay within a group, all while also coordinating their motion such that they do not collide.

Multi-robot planning can also be viewed as many individual robots that must navi-

gate among many obstacles whose motion is not known a priori. In this situation, the problem is how to have this robot adjust to the motion of the surrounding obstacles. If group tasks or formations or involved, other agents must be considered even though the single robot cannot predict how the other agents are moving. These properties of multi-robot planning make it challenging to develop a complete planner even for systems with only a few robots.

Prior techniques for motion planning and coordination in multi-robot systems can be classified into *centralized* and *decentralized* methods. The centralized methods group the robots into one large system and reduce the problem to planning the motion of a single composite robot with high DOFs. Coordination requirements can further constrain the search space and result in rather narrow passages. Decentralized methods reduce this overall complexity by exploiting the independent nature of DOFs between robots. Typically, each robot initially plans a path independently of the other robots. Then, a coordination step is required to ensure that different robots do not collide with each other while traversing their paths. However, the overall approach is susceptible to reliability issues and may not work well in practice [SL02, SI06].

The physics-based sampling framework fits naturally into a decentralized planning scheme, and this can be approached in a variety of manners. The most natural extension is to arrange the robots as in the previous chapters: each robot has a guiding path, and the physics-based simulation core along with it constraints can represent the collision avoidance needed when robots are close to other obstacles and as they near their goals. However, with a fixed guiding path, this prevents the robot from adjusting to changes in the environment which invalidate the guiding path. Thus, in another approach we have the robot's path adjust as their obstacles move. In this way, we consider a physics-based roadmap which uses the sampling to drive updates. This too has some challenges. First, the cost of frequently updating a roadmap can be prohibitively expensive for large and complex problems. Furthermore, smooth path changes on the physics-based

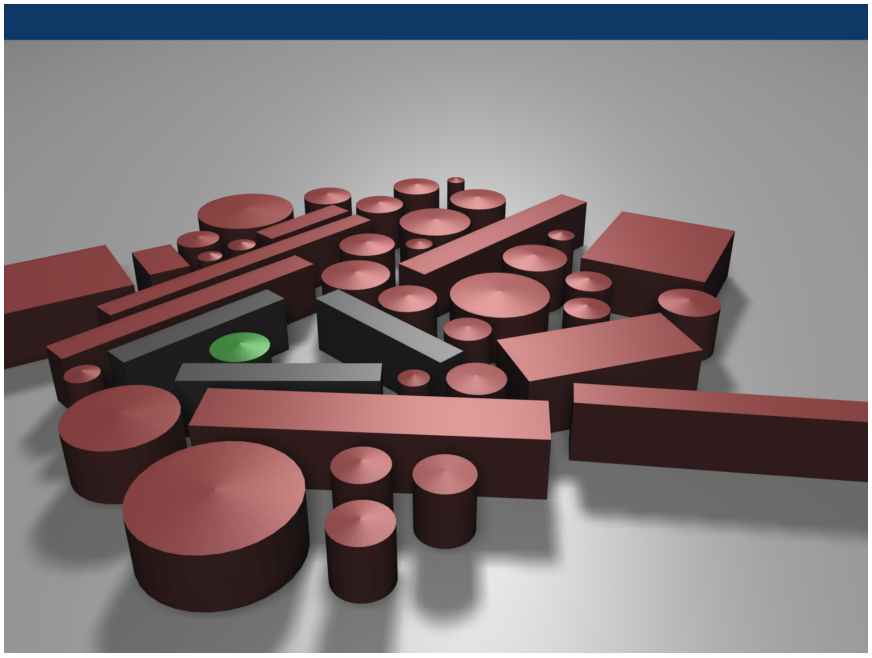

Figure 6.1: Guarding and escorting: This image is from a guarding and escorting scenario. 35 aggressive robots (in red) are trying to reach an important robot (in green). The black robots are attempting to stay in a formation while protecting the important robot. Their goal is to escort it across the environment. Our social force model allows the aggressive robot to move toward the important robot while avoiding collisions and distance coordination constraints are used to help maintain the formation. This benchmark took 132.9 s with an average step time of 42 ms.

roadmap may not always translate into motion that is physically plausible, particularly at the milestones. Finally, the motion of agents using a physics-based roadmap might ignore any require coordination between the robots unless that is also encoded into the roadmaps.

These solutions again represent both ends of a spectrum. On one end, all agent interactions are handled via physics-based sampling constraints. However, these are local in scope and cannot adjust to changes in the environment which would invalidate the guiding path. On the other hand, a more dynamic, physics-based roadmap does not scale well to large numbers of robots and is difficult to represent for a wide variety of problems. We seek a solution which uses both of these concepts. We maintain a novel

roadmap representation which incorporates some ideas from physics-based sampling to ease the work of global replanning. Individual robots, each of which follows a physics-based sampling methodology, use this roadmap as a guiding path so that they can reach their goal.

6.2 Overview

In this section, we introduce the main concepts and definitions for the remainder of this chapter. We highlight our results with a specialized model for crowd simulations in densely populated and complex environments.

6.2.1 Notation and definitions

We address the problem of collision-free motion planning for multiple robots in a dynamic environment. Consider a set of n robots, $\mathcal{R} = \{r_1 \dots r_N\}$, where each robot $r_i \in \mathcal{R}$ is a rigid or articulated body. The physical state, $\mathbf{q}_i(t)$, of robot $r_i(t)$ at time t can be described by its degrees of freedom (DOFs) and the first-order derivatives of the DOFs with respect to time, i.e. $\mathbf{q}_i(t) = [\mathbf{x}_i(t)\ \dot{\mathbf{x}}_i(t)]^T$. We make no assumptions about the motion of r relative to each other and the goal of each robot is completely independent of the other robots. A set of m constraints, $\mathcal{F} = \{C_i \dots C_m\}$ act on robots in $\mathcal{R}$ and obstacles in $\mathbf{O}$. The constraints include soft constraints such as goal seeking, the coordination constraints such as agent grouping, and other hard constraints such as non-penetration of other bodies.

A configuration $\mathbf{q}_i$ in the configuration space $\mathcal{C}_i$ of robot r_i describes the position and orientation that robot. We use the symbol $\mathcal{F}_i(t)$ to denote the free space of r_i at time t. We represent the workspace as $\mathbf{D}$ with obstacles $\mathbf{O}$ where the motion of each obstacle $\mathbf{o}_i \in \mathbf{O}$ may not be known ahead of time. We consider that state of a obstacle at time t as $\mathbf{o}_i(t)$ and refer to $\mathcal{C}_i$-obstacle as the subset $\mathcal{C} \setminus \mathcal{F}_i(t)$.

Given an initial state of the system $\mathcal{Q}(t_{initial}) = \{\mathbf{q}_i(t_{initial}) \dots \mathbf{q}_n(t_{initial})\}$, a final state $\mathcal{Q}(t_{final}) = \{\mathbf{q}_i(t_{final}) \dots \mathbf{q}_n(t_{final})\}$, and a list of constraints $\mathcal{F}$, or goal is to compute a collision-free path for each robot through the environment and among other robots or agents.

6.2.2 Extension to multiple robots and dynamics environments

Our target application is in large scale crowd and social simulations, in both simulated and online virtual worlds. Our aim to provide path planning for a large number of agents in complex environments. Thus, each agent or robot should maintain its own goal as well as any other constraints it has with respect to nearby objects or other agents.

As briefly described earlier, we decompose the problem into two separate problems. First, we consider a single agent's motion solely among other agents as would be necessary to follow a fixed, guiding path. Since this task considers constraints and obstacles only nearby a given agent, we refer to this as local navigation. Second, we consider how the guiding path itself can be adjusted using concepts from physics-based sampling to accommodate changes in which may invalidate a guiding path. Since this considers the global connectivity, we refer to this as global navigation. Finally, we highlight the integration these two approaches for a efficient and scalable planning algorithm for many agents. This will give us a list of constraints needed to solve this problem in a practical amount of time.

The following items are the main components in our multiple robot extension to physics-based sampling.

1. **Dynamic guiding path**: By utilizing the physics-based sampling core, we generalize the concept of a guiding path to be that of a "reactive deforming roadmap" (RDR). Briefly, a RDR can adapt a link or milestone within a local region of itself while motion of surrounding obstacles is relatively small. If the motion becomes too large for the link, the RDR performs a replanning step to help regain con-

nectivity if it is lost. Finally, the RDR will serve as the guiding path for each robot.

2. **Agent representation**: Following with the ideas of physics-based sampling, agents are represented by physical objects with establish motion equations.

3. **Coordination constraints**: In addition to the standard constraints defined in the physics-based simulation core, the multi-robot planning problem places restrictions on how the robots can or should move amongst each other. We develop motion constraints based on established motion models which look at how pedestrians move among each other.

4. **Integrated global and local navigation**: Rather than restrict each agent onto the path as defined by the RDR, we integrate the two methods via *link bands*. These link bands are regions of the free-space which are described by the band itself and provide an easy mechanism for defining how an agent should move through a region. These are similar to corridors, as described in some multi-robot literature [GO07].

5. **Autonomous agents in online virtual worlds**: To show the applicability of the local navigation model, we had an additional goal of planning for virtual agents in an online virtual world. Local navigation is computed for each virtual agent on its local host and commands are sent to the central server. Additional planning considerations to help account for network latencies are included.

6.2.3 Global navigation

The problem of path planning, especially in the case where obstacle motion is not known in advance, has received a great deal of attention over the past two decades. Many proposed approaches can be classified into two two broad-level techniques: *path*

modification and *replanning* algorithms. The path modification techniques allow the robot to react to dynamic obstacles, but cannot cope with changes in the connectivity of the free space. On the other hand, replanning methods update the connectivity and compute new paths during execution. However, the cost of performing these updates can be relatively high and current algorithms do not scale well to a large number of robots. To overcome these challenges, our goal is to build a roadmap data structure which incorporates both of these ideas such that the roadmap can quickly handle smaller changes to the environment, but be robust amid motion in the environment which changes the global connectivity of the roadmap itself.

Reactive Deformation Roadmaps

As previously mentioned, given a robot, r_i, we treat all the other robots as dynamic obstacles. As a result, the extended motion planning problem reduces to computing a collision-free path for each robot in a dynamic environment. By planning for each robot in this manner, each robot finds a collision free path through the environment. As the obstacles and other robots undergo motion in the environment, we update the roadmaps so that they capture the connectivity of the free space. Allowing paths themselves to react or move to remain in the free configuration space provides a simple way to respond to small changes in the $\mathcal{C}$-obstacle boundaries. To accomplish this task, we utilize *Reactive Deformation Roadmaps* (RDRs). An RDR is a dynamic data structure for motion planning of multiple robots in dynamic environments. A RDR dynamically captures the connectivity of the free configuration space $\mathcal{F}$. The RDR itself is composed of *dynamic milestones* and *reactive links*. The dynamic milestones correspond to points in $\mathcal{F}$ and are allowed to move as $\mathcal{F}$ for a robot changes. The reactive links are collision-free paths between two dynamic milestones. As the milestones associated with a link move, that link deforms based on energy minimization and thereby reacts to the motion of the dynamic obstacles and other robots. (See Fig. 6.2).

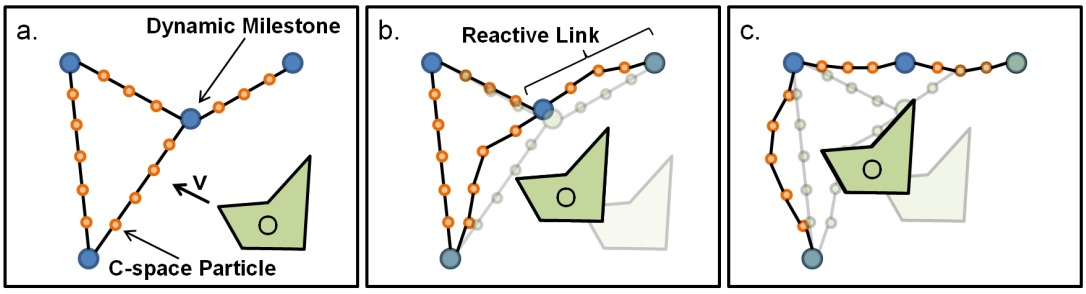

Figure 6.2: Reactive Deforming Roadmaps (RDR): The RDR contains a set of dynamic milestones and reactive links. Each of these are represented with $\mathcal{C}$-space particles: a point mass in the configuration space. As the obstacle $\mathbf{o}_i$ moves, the dynamic milestones move as well and the reactive links of the roadmap deform to avoid the obstacle boundary. (c) If a path link deforms too much or is too close to the obstacle $\mathbf{o}_i$, the link is removed.

Conceptually, the RDR itself can be thought of as a series of robot configurations along a link, each of which is using physics-based sampling to avoid obstacles and remain in $\mathcal{F}$. Instead of having a goal constraint defined by the problem, the goal constraint is defined during the creation of the roadmap instead.

We use motion equations based on Newtonian physics and Hooke's Law to determine how the milestones move and the links deform. Internally, the entire RDR (including each link) is represented as a particle system, much like in the case of our deformable soft-body agents (Chapter 4). We use internal spring forces as constraints to prevent the particles from drifting apart while external repulsive forces due to the obstacles and other robots are applied to the system. In this sense, RDRs could also be viewed as a physically-based roadmap whose motion is determined by elastic forces due to path-length minimization and an artificial potential function. We also add new milestones and links, as well as deleting some of the milestones and links to dynamically capture the connectivity of the free space.

Motion Planning with RDRs

The RDRs are used for motion planning in the following manner. In the case of a single robot, our planner proceeds in a manner similar to other planners. First, the robot's starting and final configurations are connected to the roadmap. Then, a path is computed via a graph search algorithm such as A*. As the milestones and links along the path move and "deform" the roadmap, a new path is implicitly computed and the robot automatically uses these changes so that it can follow a collision-free path accordingly. Our search algorithm also takes into account additions and deletions of links for path computation in a dynamic environment. In case of multiple robots, we may need to compute and update a separate RDR for each robot. In certain other cases, a group of robots have the same shape and configuration space, we only need to maintain a single RDR for each such group.

6.2.4 Local planning

In many randomized planners, local planning usually only considers the straight-line path between two points. If this path is determined to be collision-free, the then local behavior is to simply follow that segment (effectively separating planning from control and execution). In the case of multiple robots, local planning and behavior comes in many forms. In a centralized planner, it is simply to follow the links that lead to the goal. In a decoupled planner, many solutions are viable. If the path or roadmap is fixed, local planning usually involves an additional coordination phase where the velocities along the path are adjusted such that collisions do not occur. If a path can be rapidly replanned, then local planning again is simply following the links. Alternatively, various avoidance and goal seeking constraints can be defined which incorporate avoidance between two agents. However, all of these decoupled strategies sacrifice completeness for performance and rapid planning.

Social Potential Fields

As briefly mentioned earlier, potential field methods [Kha85] are a popular choice for real-time motion planning of multiple robots due to their efficiency and simplicity. Social potential fields (SPF) [RW99] are a variation of potential fields which characterize the potential function by inverse-force laws, rather than a navigation function. The motion of resulting systems have been shown to exhibit emergent "social" behaviors, such as clustering, guarding, or escorting.

In the initial work, given point robots $r_1, r_2, \ldots, r_n$ with positions $\mathbf{x}1, \mathbf{x}2, \ldots, \mathbf{x}n$, the force between robots i and j is defined as:

$$\mathbf{F}^{soc}(r_i, r_j) = \left(\sum_{k=1}^{L} \frac{c_{i,j}^{(k)}}{d(i,j)^{\sigma_{i,j}^{(k)}}} \right) \left(\frac{\mathbf{x}_j - \mathbf{x}_i}{d(i,j)} \right) \tag{6.1}$$

Thus, the social force acting on r_i from r_j is the summation of L force laws, where $c_{i,j}^{(k)}$ is the force coefficient for the kth law, $\sigma_{i,j}^{(k)}$ is the inverse power of the kth law, and $d(i,j) = ||\mathbf{x}_j - \mathbf{x}_i||$. Here, the sign of the force coefficient determines if the force acts in a attractive (positive) or repulsive (negative) manner. These force laws can be easily assigned based on desired behavior to an individual robot, a group of robots, or even an entire set of robots.

Other variations on the Eq. 6.1 have been proposed, such as that by Helbing et al [HBW03]. The behavior of Helbing's social force law is similar to that of Eq. 6.1, but also has been shown to exhibit emergent behavior of lane formation for point-based robot primitives. This force is defined as:

$$\mathbf{F}^{soc}(r_i, r_j) = \alpha e^{-d(i,j)/\beta} \mathbf{n}_{i,j} \tag{6.2}$$

where α is our force coefficient, β is the force fall-of distance, and $\mathbf{n}_{i,j}$ is the normalized vector between $\mathbf{x}_i$ and $\mathbf{x}_j$. Similarly, the sign of α determines whether the force is

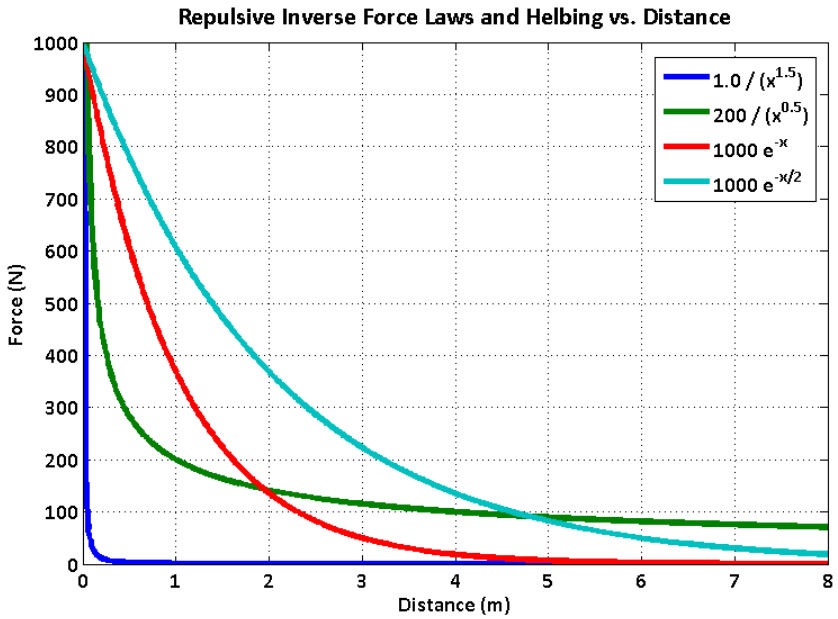

Figure 6.3: Social Potential Fields: This figure compares Inverse-force laws (dark-blue and green curves) and Helbing's social force laws (red and cyan curves) with differing parameters. The Inverse-force plots initially decline rapidly, and then the rate of decline slows greatly. On the other hand, the Helbing plots do not smoothen out quite as quickly and do not grow indefinitely as distance approaches 0.

repulsive or attractive. Additionally, like Eq. 6.1, several force terms could be used instead of just one for both repulsive and attractive behaviors.

Our Approach

As previously mentioned, our goal is to combine key ideas of physics-based sampling and social potential fields. In most cases, a potential field can be sampled and converted into an artificial force field. Thus, social potential fields can be incorporated into the physics-based sampling approach by describing the social potentials as forces. In our approach, an artificial force field applies forces directly onto the robot to influence its motion. In this manner, the artificial force field is a type of constraint in itself.

We utilize this property to establish a general local avoidance function based on social potential fields. We generalize the approach to consider both convex and nonconvex

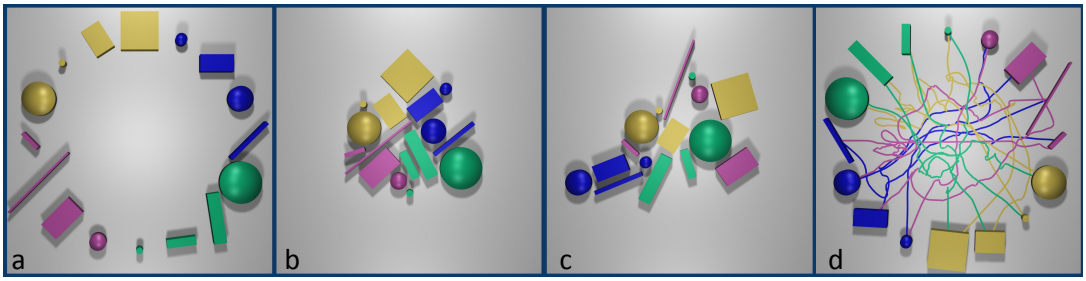

Figure 6.4: Antipodal Robots: In this benchmark, 16 polygonal robots, colored by starting quadrant, start along the edge of a circle. Each robot must travel to a position across the circle and to a different orientation. This sequence shows four steps in the planning. (a) The robots are at their initial position. (b-c) The robots are in the process of moving across the circle, where social forces help prevent collision while also leading toward the anti-podal position. (d) The agents arrive at their anti-podal positions and we see the paths they traveled.

polyhedra instead of solely point-like bodies. We consider the use of both types of social potential functions, Eq. 6.1 and 6.2. Since we achieved similar results with both, we decided to use Eq. 6.2 since its magnitude is bounded as distance approaches 0 (See Fig. 6.3). This helps greatly for simulation stability since large forces result in stiff systems. Also, the parameters of Eq. 6.2 are more intuitive to tune, since they carry physical meaning. Briefly, α is the maximum magnitude of the social force in Newtons and β is the distance at which the force is approximately $\frac{1}{3}$ of its maximum. Finally, there parameters can also allow for algorithmic improvements which can help with performance.

On top of PMP and SPF, many coordination constraints such as maintaining formations can be easily represented and quickly computed to give the robot the appearance of more advanced behaviors. In the case of crowd simulation, generalized social potential fields can be reduced to acting on a single point or particle, further reducing the computational effort required for the task.

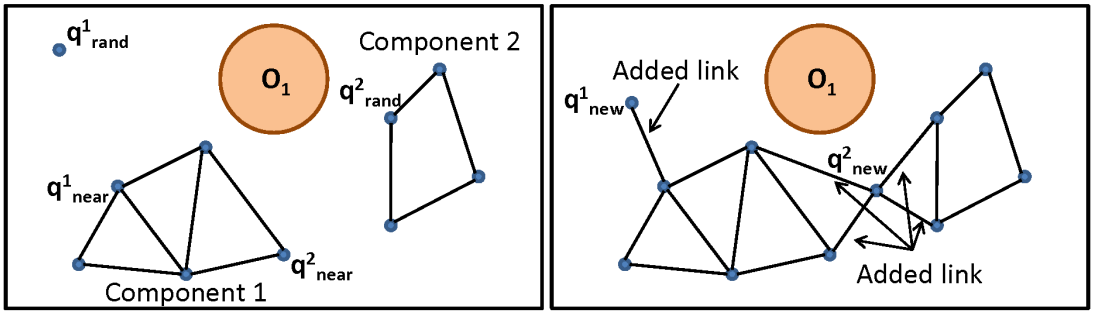

Figure 6.5: Adding links to the RDR: (Left) To explore, a random sample q^1_{rand} is generated, and its nearest neighbor to the roadmap is found, q^1_{near}. To merge different connected components, a random milestone, q^2_{rand} on Component 2 is selected and its nearest neighbor on Component 1 is found, q^2_{near}. (Right) New milestones, q^i_{new} are added by extending q^i_{near} toward q^i_{rand} and valid straight-line links are added.

6.3 Global navigation

In this section, we give details on building RDRs and modifying them based on the motion of obstacles or other robots. While RDRs are defined over $\mathcal{C}$, there are several optimizations and uses for RDRs when planning can be done in the workspace.

6.3.1 Reactive Deforming Roadmaps

An RDR, $\mathcal{R}$, is represented as a set of dynamic milestones, $\mathcal{V}$, and reactive links, $\mathcal{E}$ between the milestones, $\mathcal{R} = (\mathcal{V}, \mathcal{E})$. The milestones and links move as the free space of the robot changes over time. Each dynamic milestone is associated with a $\mathcal{C}$-particle: a point in $\mathcal{C}$ with a mass, to which forces can be applied. The state of a particle i consists of its position ($\mathbf{q}$), $\mathbf{x_i}(t)$ and its velocity ($\dot{\mathbf{q}}$), $\mathbf{v_i}(t)$, at time t can be defined as

$$\mathbf{p_i}(t) = \begin{pmatrix} \mathbf{x_i}(t) \\ \mathbf{v_i}(t) \end{pmatrix}.$$

Similarly, reactive links are represented as a sequence of these particles, placed at equal distances along an initial straight-line link. The spacing, or density, of the particles

along a link is specified in advance. This approach allows the link between connected milestones to behave as a short reactive path. These elements are shown in Fig. 6.2. In order for the RDR to be valid, the straight-line path between the connected particles must be valid, i.e. lie in the free space.

The RDR can be initialized using well-known algorithms for constructing a roadmap such as random sampling in the configuration space. In cases where it is sufficient to plan in the work space, a medial axis can be computed and sampled.

Applied Force Computation

We apply forces to move the milestones and links away from obstacles while simultaneously maintaining the connectivity of the roadmap. We define two types of forces: *internal* restoring forces and *external* reacting forces. The internal forces are spring-like or elastic such that the links attempt to stay in a state of minimum energy. This constraint helps us minimizing the jittery motion and provides smoother and shorter paths. The external forces are generated in a manner similar to that of potential field methods [Kha85] and are used to retract the roadmaps as they react to dynamic obstacles and other robots in the scene. The total force acting on mass i is given as:

$$\mathbf{F}_i = \mathbf{F}_i^{int} + \mathbf{F}_i^{ext}$$

Based on these forces, the algorithm repositions the affected particles (or milestones) thereby updating the state of the roadmap.

Internal Forces The internal restoring forces are necessary to prevent the particles on the same link from moving unnecessarily far apart during the deformation and also to prevent the roadmap from drifting too far from its initial configuration. The force on

$\mathcal{C}$-particle p_i from a connected $\mathcal{C}$-particle p_j is given as:

$$\mathbf{F}_{ij}^{int} = -k_s(\|\mathbf{x}_{ij}\| - L_{ij})\frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|},$$

where k_s is a spring constant, $\mathbf{x}_{ij}$ is the vector between particle p_i and particle p_j, and L_{ij} is the initial separation distance between the two particles. An equal and opposite force is applied to particle p_j.

External Forces As robots or dynamic obstacles move through the environment, the RDR responds to the changes in the connectivity of the free space by moving the dynamic milestones and deforming the reactive links. External forces from other moving robots and dynamic obstacles are applied to the particles. Given the particle p_i, the force from obstacle O_j is

$$\mathbf{F}_{ij}^{ext} = \frac{b}{d(p_i, O_j)^2 + \epsilon}, \tag{6.3}$$

where b is a constant. We set $\epsilon > 0$ so that the force due to the obstacle O_j does not cause numerical issues and it helps to improve stability of the system.

Roadmap Deformation Once the forces on each particles have been determined, the state of the roadmap is updated.

A complete dynamics computation is inappropriate since it can capture all the transient states, including unwanted oscillation, in the reactive links. Instead, we preform a quasi-static simulation by using a variant of forward Euler integration that considers each particles to be at rest throughout the time step. For each particle i, we update its position and velocity as follows.

$$\mathbf{x_i}(t + dt) = \mathbf{x_i}(t) + \frac{1}{2}\mathbf{F}_i^{net}(dt)^2.$$

Since this method is based on an explicit Euler integration scheme, it is susceptible to numerical issues and instabilities. Many of these issues are reduced by bounding

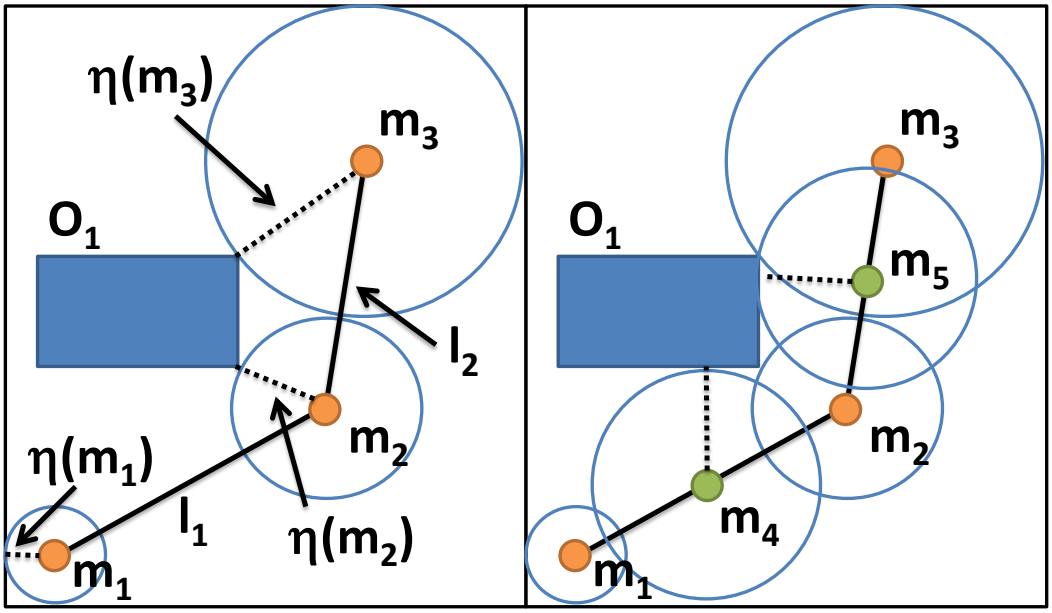

Figure 6.6: Unsafe and Safe Link Samplings: (Left) The default sampling of link l_1 is insufficient, since the safety region (circles) defined by $\eta(\mathbf{p}_1)$ and $\eta(\mathbf{p}_2)$ do not overlap. Similar for link l_2. (Right) The addition of particles $\mathbf{p}_4$ and $\mathbf{p}_5$ result in safe links, since the the entire link is covered by some safety region.

the magnitude of the potential functions. Furthermore, our potential energy based link removal method is tuned to remove links before they become unstable.

Adaptive Particle Sampling of Links

As previously mentioned, the links in a roadmap are discretized into a sequence of particles to allow them to react to the motion of dynamic obstacles. This discretization plays an important role in the performance of our algorithm. Prior work in adaptive roadmaps uses a uniform sampling based on a user-specified distance between particles [SGA+07]. While this gives control of a range of motion it also had several drawbacks, primarily on its behavior and run-time performance.

Uniform Sampling: In general, the behavior of a link is governed by the number of particles used to discretize it. Too few particles results in an inadequate range of link deformation, impeding its ability to avoid obstacles. On the other hand, too many particles can result in oscillations or unwanted bends along the link from compression.

Also, the performance of a link and the resulting roadmap is linearly proportional to the number of particles used to discretize the entire roadmap.

In the case of uniform sampling, the distance-between-particles parameter must be adjusted for each scenario in order to find a balance between desirable behavior and performance. To the best of our knowledge, no good techniques are known for automatically computing the parameter for all kinds of complex and dynamic environments.

Furthermore, uniform sampling can result is a great deal of unnecessary computation. In areas of the environment where little or no motion is happening, it is not necessary to have many particles. Similarly, we would like to localize the computation to areas in closer proximity to obstacles.

Adaptive Sampling: To overcome many of these issues related to uniform sampling, we use an adaptive procedure based on workspace distances between adjacent samples. Our formulation greatly reduces the number of particles in the scene, localizes computation for smoother links and removes the need to have an user tune a distance parameter.

Our approach, Adaptive Particle Sampling of Links (APSL), is based on the notion of distance bounds in $\mathcal{F}$. In our representation, when a particle $\mathbf{p}_i$ is farther away from an obstacle, the applied force on $\mathbf{p}_i$ will be more similar to that of its neighbors. Therefore, the resulting motion of the link will be similar to that of $\mathbf{p}_i$. As a result, we can reduce the number of particles that need to be in close proximity to $\mathbf{p}_i$. Similarly, as a particle $\mathbf{p}_j$ nears an obstacle, the motion of the link containing $\mathbf{p}_j$ will be less like that of $\mathbf{p}_j$ and more particles are needed nearby. Our approach is based on defining a notion of particle safety regions, and using them for sample generation.

In the motion planning literature, it has been shown that the straight-line between two arbitrary robot configurations is collision free [SSL02] if

$$\rho d(\mathbf{p}_i, \mathbf{p}_j) < \eta(\mathbf{p}_i) + \eta(\mathbf{p}_j) \tag{6.4}$$

where ρ is a constant defined by the agents, $d(\mathbf{p}_i, \mathbf{p}_j)$ is the distance between particles i and j, and $\eta(\mathbf{p}_i)$ is the nearest distance to a non-agent obstacle. This criterion can easily be adapted to our agent representation.

In relation to our approach, $\eta(\mathbf{p}_i)$ can be considered the safety region for $\mathbf{p}_i$. Eq. 6.4 gives a criterion for when a sampling along a link is sufficient based on its distance bounds in $\mathcal{F}$. Fig. 6.6 shows two cases where the sampling is insufficient or sufficient based on this equation.

We use a two-pass approach to generate a sample based on this distance bound. The first stage checks each segment between connected particles to ensure that Eq. 6.4 is satisfied. In the case that Eq. 6.4 is not satisfied between particles $\mathbf{p}_i$ and $\mathbf{p}_j$, an adaptive subdivision process is used generate samples until the condition is satisfied along a link. First, a new particle $\mathbf{p}_{new}$ whose position is half way between $\mathbf{p}_i$ and $\mathbf{p}_j$ is created. Then, Eq. 6.4 is checked between $\mathbf{p}_i$ and $\mathbf{p}_{new}$, and also between $\mathbf{p}_{new}$ and $\mathbf{p}_j$. The subdivision process continues until each sub-segment satisfies the criterion. Each time a pair is subdivided, the spring connecting the original pair is removed, and springs are added for each of the pairs which satisfy the criterion.

The second pass removes redundant particles. A particle $\mathbf{p}_i$ is considered redundant if it falls within the safety region of two other particles. In this case, the behavior of particle $\mathbf{p}_i$ will be represented by its neighbors and therefore $\mathbf{p}_i$ is not necessary. So, $\mathbf{p}_i$ is removed from its link. The neighboring particles are then connected by a linear spring. For simplicity, we only check pairs of particles that are at most one particle apart. Since at most half the particles along a link can be removed at any timestep, it will take at most $O(log(n))$ time-steps to remove all the particles along a link or to reach a minimal sampling based on Eq. 6.4. The results for both passes can be seen in figure 6.7.

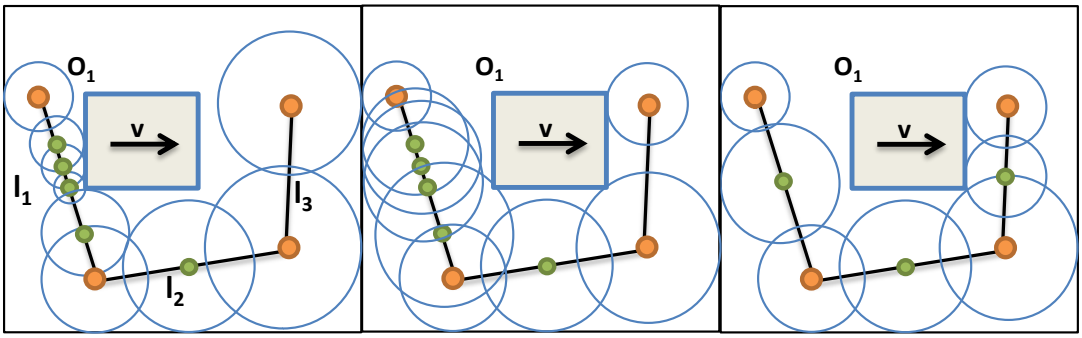

Figure 6.7: Adding and Removing Particles: (Left) The links, between orange circles as the endpoints, have been sampled based on the position of a dynamic obstacle O_1. Samples are shown as green circles. The blue open circles show the safety region for each particle (Middle) As O_1 moves, $\eta(\mathbf{p}_i)$ is updated for each particle. As a result, there are redundant particles on link l_1 and too few particles on link l_3. (Right) The add particle procedure adaptively adds particles to l_3 until it passes the safety criterion, and particles who belong in the safety regions of its neighbors have been removed from l_3.

Data: Roadmap R

Queue Q;

forall *Links* $l_i \in R$ **do**

  forall *Connected particles* $(\mathbf{p}_i, \mathbf{p}_j) \in l_i$ **do**

    if *not Safety-Criterion*($\mathbf{p}_i$, $\mathbf{p}_j$) **then**

      Remove-Pair(($\mathbf{p}_i$, $\mathbf{p}_j$), l_i);

      $\mathbf{p}_{new}.\mathbf{x}_{new} = \frac{1}{2}(\mathbf{p}_i.\mathbf{x}_i + \mathbf{p}_j.\mathbf{x}_j)$;

      Add-Particle($\mathbf{p}_{new}$, l_i);

      Add-Pair(($\mathbf{p}_i$, $\mathbf{p}_{new}$), l_i);

      Add-Pair(($\mathbf{p}_{new}$, $\mathbf{p}_j$), l_i);

    end

  end

  forall *Particles* $\mathbf{p}_i \in l_i$ **do**

    if $\mathbf{p}_i$ *is not a milestone* **then**

      ($\mathbf{p}_{left}$, $\mathbf{p}_{right}$) = Get-Neighbors($\mathbf{p}_i$);

      if *Safety-Criterion*($\mathbf{p}_{left}$, $\mathbf{p}_{right}$) **then**

        Remove-Particle($\mathbf{p}_i$, l_i);

      end

    end

  end

end

Algorithm 5: **Adaptive Particle Sampling of Links:** *Add-Pair*() and *Remove-Pair*() add and remove a connection between their arguments, respectively. *Add-Particle*() and *Remove-Particle*() add and remove a particle and remove any connections from that particle from a link, respectively. *Get-Neighbors*() returns particles which are connected to the input particle. Briefly, the first **forall** loop adds any particles to satisfy the safety criterion (Eq. 6.4) while the second loop removes redundant particles. More details are in Sec. 6.3.1.

A pseudo-code description for these passes is presented in Algorithm 5. The roadmap R is initialized so that each link only contains a single elastic connection between its two milestones. Particles are considered connected if there exists a spring connecting them in the representation. In this algorithm, the function *Safety-Criterion*($\mathbf{p}_i$, $\mathbf{p}_j$) determines whether or not the safety regions of each particle overlap. The *Remove-Pair*($\mathbf{p}_i$, l_i) and *Add-Pair*(($\mathbf{p}_i$, $\mathbf{p}_{new}$), l_i) functions remove and add connections and springs between their inputs, respectively. Similarly, *Add-Particle*($\mathbf{p}_{new}$, l_i) simply adds the particle to the list of particles. It is important to note that *Remove-Particle*($\mathbf{p}_i$, l_i) not only removes the input particle, but also removes the connections associated with that particle. Finally, *Get-Neighbors*($\mathbf{p}_i$) returns the pair of particles that are connected with the input particle, $\mathbf{p}_i$.

In this algorithm, the first of the inner **forall** loops is the first pass which subdivides the links. This routine checks each connected pair of particles for the safety criterion. If any of them fail this criterion, it places two new pairs in its place on the list of particle pairs that need to be checked. The second inner **forall** loop removes redundant particles by checking the safety criterion.

At the conclusion of the algorithm, we have guaranteed that all portions of all links are covered by the safety region of some particle. Moreover, redundant particles are removed to help minimize the computational cost of updating the roadmap.

6.3.2 Global Updates

In the previous subsection, we described a general representation for RDR and how it adjusts to the moving obstacles. However, this type of deformation can not guarantee a valid roadmap or that a path can be found at all times, e.g. when a moving obstacle moves directly into a link. These cases require additional link removal or link addition steps to update the roadmap.

Link Removal

Link removal is essential for maintaining a valid roadmap and plays a key role in its resulting behavior. If links are removed too frequently, then we may have rather small deformation in the links. On the other hand, if the links are allowed to deform considerably, this may cause problems with the numerical stability of the simulation or may unnecessarily increase the path lengths of some links. We use two properties to evaluate the links for removal in our framework: amount of deformation and proximity to the obstacles.

For the link i, the extent of deformation of a link can be measured through its potential energy,

$$E_i(t) = \sum_{s_j \in l_i} \frac{k_s^j}{2}(d_j - L_j)^2,$$

where k_s^j is the spring constant for spring j, d_j is its current length, and L_j is its rest length when unstretched or compressed. When $E_i(t) > \tau_s$ for some user specified potential threshold τ_s, the link is removed and the milestones incident to the link may no longer be connected. For example, the link in Fig. 6.2(c) is removed as it needs to be considerably deformed due to obstacle O.

In order to account for proximity to any obstacle or other robot, the minimum separation distance (in workspace) is computed from a path link to the nearby obstacles. If this distance value is less than a pre-defined proximity threshold τ_d, then the link is removed. Specifically, τ_d is chosen such that the link remains at a safe distance from the obstacle, for instance, the radius of the bounding sphere for rigid robots.

Adding Links and Milestones

The process of removing links can result in losing connectivity information about $\mathcal{F}$. Moreover, as the obstacles move, we may need to add new links or milestones to capture the connectivity. Our framework uses three methods for this computation.

In the first method, we utilize the fact that many obstacles are ephemeral and that a formerly valid path will likely become valid again. To implement this strategy, a list of removed links is maintained. Periodically, or when no path can be found for a robot, the straight-line path in the configuration space for some or all of the links in this list is tested for validity. If they are valid, a reactive link is added in its place.

The second approach tries to connect various disjoint components of the roadmap by using a construction algorithm similar to that used in the RRT algorithm [LaV06b]. Given a component C_i, for each other component C_j, a random milestone $q_j \in C_j$ is chosen. Then, the nearest neighbor $q_{near} \in C_i$ to q_j is computed. Next, a new milestone, q_{new}, whose configuration extends q_{near} toward q_j is computed such that it has a valid straight-line path to q_{near} in the configuration space. Finally, straight-line paths from q_{new} and the nearest neighbors in C_i and C_j are connected if they lie in the free space.

Our third approach extends the roadmap toward unexplored regions of the free space, also in a manner similar to that of RRT algorithm [LaV06b]. First, a random configuration q_{rand} is generated and its nearest neighbor on the roadmap q_{near}. Moreover, we bias these samples toward the obstacles, since least explored areas are likely to be near them. A new milestone q_{new} is generated that extends q_{near} toward q_{rand}. Then, straight-line paths from q_{new} to its nearest neighbors are tested and added to the roadmap if they are valid.

Numerical Stability

Since RDR relies upon numerical integration due to particle simulation, stability is a concern due to the possibility of applying large spring or repulsive forces. However, when combined with the removal rules, this has not been an issue in our benchmarks. In general, using Verlet integration greatly helps in stability by treating the particle to be at rest during integration. Also, our link removal steps help in ensuring stability. Any link which is likely to become unstable will also likely be close to an obstacle or

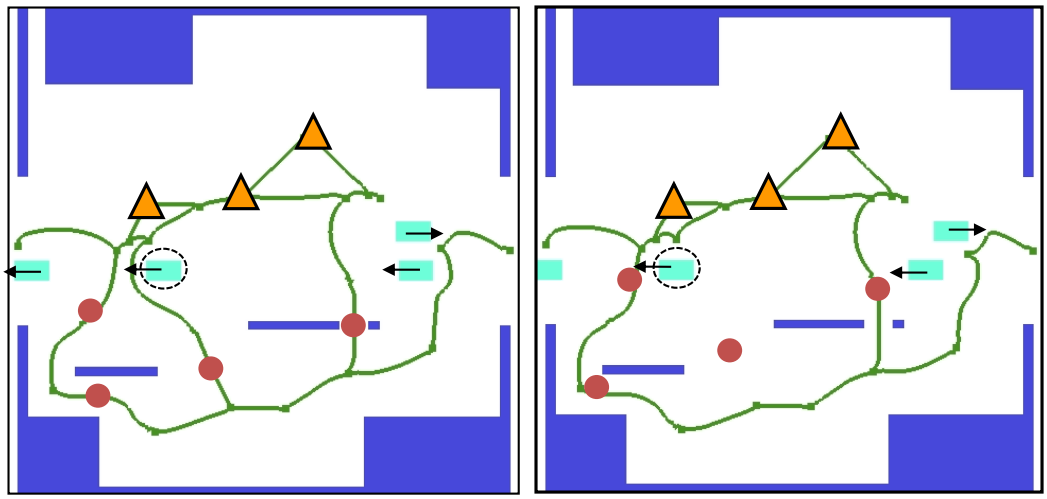

Figure 6.8: Reactive Deforming Roadmaps (RDR): An example with 4 translational agents (red circles) and goals (yellow triangles). The static obstacles shown as dark blue rectangles and dynamic obstacles are shown as cyan rectangles with arrows indicating the direction of motion. The green curves represent links of the reacting deforming roadmap, using uniform sampling. The dynamic obstacles represent cars. As the highlighted car (circled) moves, the affected link in the roadmap is removed.

otherwise highly deformed. These types of links will be removed, and thereby helping the system remain in a stable state.

6.4 Local navigation

Our approach builds upon physics-based motion planning (PMP) [GL02, GRS+07] and social potential fields (SPF) [RW99, HBW03]. PMP solves motion planning problems through the use of a constraint-directed physical simulation. Social potential fields are a variation of the potential field methods whose resulting motion appears to exhibit various social behaviors.

We generalize and reformulate social potential fields to be integrated with the PMP framework and also to be applicable to a more general robot primitive. The social potentials will act as a constraint on the system, allowing the robot to change its current trajectory and move toward or away from other robots or obstacles. The PMP framework

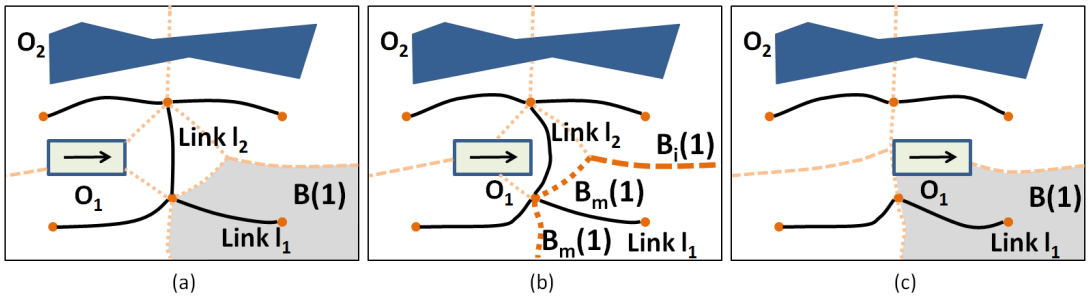

Figure 6.9: Roadmap Link Bands: Link bands are a partition of the freespace based on the links of RDR. (a) Several RDR links, in solid black lines, respond to a static obstacle, O_2 and a dynamic obstacle O_1. The link band, $B(1)$, for link l_1 is shaded, and the link boundaries are shown as dashed lines. (b) As O_1 approaches link l_2 it deforms. Link band $B(1)$'s boundary is highlighted in bold dashed lines and shows the two segments of the milestone boundary, $B_m(1)$, and the intermediate boundary $B_i(1)$. (c) Link l_2 is removed due O_1's motion while the link band $B(1)$ changes to reflect the removal.

supports additional constraints, such as creating and maintaining group formations and other behaviors. To reduce the effects of local minima, an *excitation factor* has also been included. When a robot's velocity is low and it is not at its goal, excitation will increase the strength of the forces leading toward its goal to encourage it to escape the minimum.

The method is relatively simple to implement and provides practical solutions to a wide variety of complex problems. Fig. 6.1 highlights an example of our approach applied to a group of robots (in black) guarding and escorting an important robot (in green). Additional benchmarks and more detailed results are given in Sec. 6.6.2.

6.4.1 Local Coordination with Generalized Social Forces

Social forces have been shown to work well for coordination and planning of point-based primitives. However, little work has been done on a similar formulation for a general robot. In this section, we generalize the concept of social forces and integrate it along with other coordination constraints into the physics-based motion planning framework.

Generalized Social Potential Fields

As previously mentioned, social potential fields are an useful tool for motion and coordination which resemble various complex behaviors. Here, we propose a generalized version of social potential fields which are applicable to non-point primitives. For a general body, the effect of a social potential field can be felt with varying magnitudes across the entire body. The potential field will not only apply a force, but also induce a torque on the robot.

From Eq. 6.2 we first define the value of the social potential field (in the robot's workspace) from robot r_j at an arbitrary point in space, $\mathbf{x}$ by

$$\mathbf{F}^{soc}(\mathbf{x}, r_j) = \alpha e^{-d(\mathbf{x}, r_j)/\beta} \mathbf{n}(\mathbf{x}, \mathbf{r_j}) \tag{6.5}$$

where $d(\mathbf{x}, r_j)$ is the shortest distance between $\mathbf{x}$ and r_j, and $\mathbf{n}(\mathbf{x}, \mathbf{r_j})$ is the normal vector between $\mathbf{x}$ and the nearest point on r_j. A similar social potential can be defined from Eq. 6.1.

Using Eq. 6.5, the social force between r_i and r_j, can then be defined as

$$\mathbf{F}^{soc}(r_i, r_j) = \int_{\mathbf{p} \in A(r_i)} \mathbf{F}^{soc}(\mathbf{p}, r_j)\, dA \tag{6.6}$$

where $A(r_i)$ is the region of r_i and $\mathbf{p}$ is a point on the area of r_i in 2D or a point in the volume of r_i in 3D. The associated torque on the body is

$$\tau^{soc}(r_i, r_j) = \int_{\mathbf{p} \in A(r_i)} (\mathbf{n}(\mathbf{p}, r_j) \times \mathbf{F}^{soc}(r_i, r_j))\, dA \tag{6.7}$$

In effect, the translation component of the total force on r_i will be the resulting social force over the entire region of the robot, and the total torque will similarly be the torque induced by each of these forces at each point on the body.

Like the formulations in Eq. 6.1 and Eq. 6.2, the constants of the social force can

be adjusted for varying strengths, ranges of effect, and for repulsion or attraction. For a repulsive social force, as the distance between these points decreases the resulting force and torque will increase causing those points on the bodies to separate. Then, the total social force and torque on a single robot would be the summation of the resulting values of Eq. 6.6 and 6.7.

Discretization

While the general social potential field is formulated over a continuous space, it is not simple to evaluate this integral in implementation. Instead, in our implementation we approximate the integral. Given a uniform sampling of the robot r_i's region, $\mathcal{P}^i = \{\mathbf{p}_1^i, \mathbf{p}_2^i, \ldots, \mathbf{p}_K^i\}$

$$\mathbf{F}^{soc}(r_i, r_j) = \frac{1}{K} \sum_{\mathbf{p} \in \mathcal{P}^i} \mathbf{F}^{soc}(\mathbf{p}, r_j) \tag{6.8}$$

and

$$\tau^{soc}(r_i, r_j) = \frac{1}{K} \sum_{\mathbf{p} \in \mathcal{P}^i} (\mathbf{n}(\mathbf{p}, r_j) \times \mathbf{F}^{soc}(\mathbf{p}, r_j)) \tag{6.9}$$

It should be noted that this can lead to a very large number samples, which could become computationally expensive. By using Helbing's formulation of the social force, the force parameters provide several ways of approximating the summations. First, note that the parameter β controls the rate at which the $\mathbf{F}^{soc}$ declines. Thus, when the distance between points and robots is greater than $c\beta$, for relatively large c, the value of the force is negligible and does not contribute greatly to the social force. In our implementation, we set c such that $\alpha e^{-c} < 0.001$. We can use this value as a distance bound and quickly cull away the negligible point-robot pairs. (See Fig. 6.10).

Excitation

One well known drawback of potential field based methods is the possibility of getting stuck in a local minima. This is also the case for the forces generated by our approach. To help overcome a local minima, we define an *excitation* factor, E. Briefly, while an agent has not reached its goal and its velocity is below a predefined threshold, v_0, its

excitation level increases. As excitation rises, the attractive force in the direction of its goal grows as well.

We define excitation to evolve over time as

$$\frac{dE}{dt} = -\frac{E}{T} + (1 - \frac{||\mathbf{v}||}{v_0})$$

where T is a damping factor (we use $T = 2$). Essentially, the first term smooths the excitation so that it does not grow too large too quickly, while the latter term increases or decreases the level of excitation depending on how fast a robot is moving in relation to v_0. Then, given E, we can redefine α in Eq. 6.2 by

$$\alpha(E) = \alpha * (1.0 + E)$$

In practice, this approach has performed well and we have not observed many situations where a complete deadlock has occurred due to local minima.

6.4.2 Additional coordination constraints

The social potential forces provide a strong basis for coordination between several robots. A variety of behaviors including grouping, following, guarding, moving toward a local goal, and more can be directly implemented through manipulation of the social potential fields. However, there are a number of situations which are less suitable for use with these fields.

Distance Constraints

There are several situations in which group of robots need to be able to form and maintain a formation. While strong attractive social fields could be used to help keep robots in close proximity, they lose strength rapidly when robots get further apart. Therefore, we add a distance constraint which works to maintain a distance between points on a pair of robots. Depending on the strength of the constraint, other robots

may or may not be able to easily affect a formation built from these constraints. Briefly, we can define a distance constraint between points $\mathbf{x}i$ and $\mathbf{x}j$ on different robots by the function:

$$C_{dist}(\mathbf{x}i, \mathbf{x}j) = ||\mathbf{x}j - \mathbf{x}i|| - r$$

where r is the desired distance between the points. While this constraint could be solve exactly, we relax the constraint so that the formation can also adapt to necessary changes in the environment or by other robots. Instead, we use a spring-like force:

$$\mathbf{f}_{dist}(\mathbf{x}i, \mathbf{x}j) = -\left(k_s(|\mathbf{x}| - r) + k_d\frac{\dot{\mathbf{x}} \cdot \mathbf{x}}{|\mathbf{x}|}\right)\frac{\mathbf{x}}{|\mathbf{x}|}$$

where k_s is a spring constant, k_d is a spring damping constant, and $\mathbf{x} = \mathbf{x}j - \mathbf{x}i$. As before, the related torque on robot r_i is:

$$\tau^i_{dist}(\mathbf{x}i, \mathbf{x}j) = (\mathbf{d}_{\mathbf{x}i,O}) \times \mathbf{f}_{dist}(\mathbf{x}i, \mathbf{x}j)$$

where $\mathbf{d}_{\mathbf{x}i,O}$ is the vector from the origin of the inertial frame O to $\mathbf{x}i$. The net result of this force can be seen in Fig. 6.1, since the guard robots use a distance constraint to maintain their formation.

6.4.3 Online virtual world constraints

Large, online 3D virtual worlds (VWs) have been growing rapidly in popularity due to rich environments, widespread interactivity and immersion, and large user bases. Millions of users are registered and are actively participating in worlds such as Second Life® (SL™), World of Warcraft™ (WoW), or OLIVE™ by Forterra Systems, Inc. In addition to these online worlds, Microsoft's Virtual Earth™ or Google Earth™ represent large and detailed navigable environments, although they do not currently support avatars. Many of these virtual worlds have been used for gaming applications

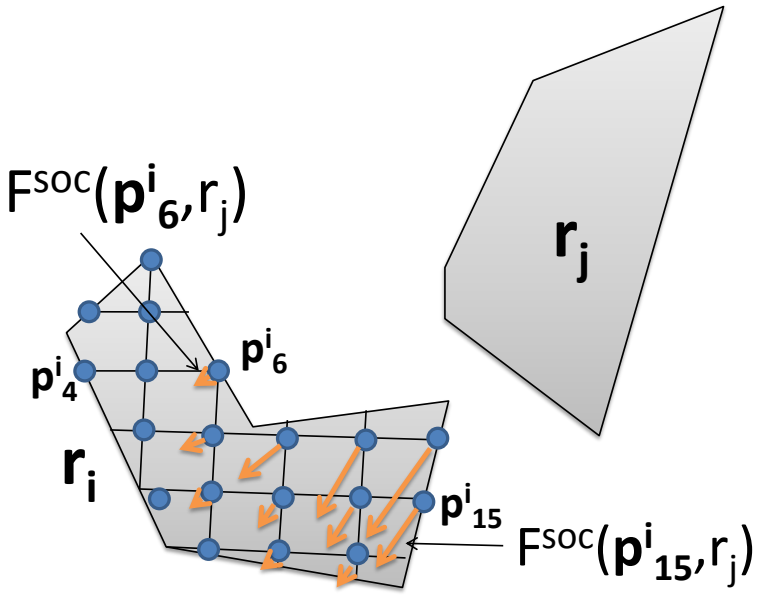

Figure 6.10: Social Force Discretization: Since the general social force integral is difficult to compute, we discretize the domain. This figure shows the discretized social force computation on r_i from r_j. Robot r_i has been uniformly sampled into a set of points, shown as small blue dots. For each sample, $\mathbf{p}_n^i$ within a fixed distance from r_j, $\mathbf{F^{soc}}(\mathbf{p}_n^i, r_j)$ will be computed. By the force law definition, points closer to r_j such as $\mathbf{p}_{15}^i$ will receive larger forces than points farther away, such as $\mathbf{p}_6^i$. For points too far away, such as $\mathbf{p}_4^i$, no force is computed.

and recently they have been shown useful for remote collaboration, economic planning, social simulations, and educational activities [Bai07]. These online virtual worlds also provide a unified environment to study complex global behaviors such as traffic flows or evacuations.

At any time, thousands of users are logged on, but they are typically widely distributed over the virtual world. As a result, many portions of these worlds appear sparsely inhabited. In simulations or experiences where interaction with other avatars is essential, this can pose a problem and can detract from the overall immersive experience. One way to overcome this problem is to add virtual avatars to populate these worlds and improve the overall immersive experience. In this context, a virtual agent or avatar is a member of the virtual world whose motion is controlled entirely by a simulation,

Figure 6.11: College campus: (a) Many areas in online virtual worlds, such as this college campus in Second Life®, are sparsely inhabited. (b) We present techniques to add virtual agents and perform collision-free autonomous navigation. In this scene, the virtual agents navigate walkways, lead groups, or act as a member of a group. A snapshot from a simulation with 18 virtual agents (wearing blue-shaded shirts) that automatically navigate among human controlled agents (wearing orange shirts). (c) In a different scenario, a virtual tour guide leads virtual agents around the walkways among other virtual and a real agent.

whereas a human agent is controlled by a human user.

There are several challenges in adding realistic virtual agents to online virtual worlds. First, virtual agents should automatically navigate the environments and act appropriately in a variety of situations, such as maintaining a separation from other agents or moving in a formation. Secondly, networking issues such as limited bandwidth and client-to-server latency add uncertainties into much of the navigation process, resulting in incomplete information about the environments. Finally, performance is a key consideration in evaluating online virtual worlds and adding virtual agents should not greatly degrade the performance of the online virtual world system.

Fortunately, physics-based sampling with GPSF as previously described can adequately the local interactions between virtual agents in an online. However, it makes the assumption that positions and velocities of agents and obstacles are known at any time. Due to latency between hosts, this is not usually the case with online worlds. We include additional constraints which can help to account for problems due to latency.

The local motion model for each virtual agents is based on our concept of generalized social potential fields.

For virtual agents r_i, our base local navigation model includes the components of GPSF as described above. Since all the agents can be represented as cylinders, this reduces the discretization to simply be a point in 2D. To summarize, we can use the following forces applied to the center of mass of each agent. A social repulsion and attraction force ($\mathbf{F}_j^{soc}(r_i)$ and $\mathbf{F}_j^{att}(r_i)$, respectively) determines how much agent r_i wants to avoid or move towards agent r_j. A goal force, $\mathbf{F}^{goal}(r_i)$, encourages an agent to move towards its current goal (See Fig. 6.12(a)).

In order to compute total force applied to an agent, we aggregate these forces as follows:

$$\mathbf{F}^{net}(r_i) = \mathbf{F}^{goal}(r_i) + \mathbf{F}^{agents}(r_i) + \mathbf{F}^{obstacles}(r_i), \tag{6.10}$$

where

$$\mathbf{F}^{agents}(r_i) = \sum_{r_j \in \mathcal{R}, i \neq j} (\mathbf{F}_j^{soc}(r_i) + \mathbf{F}_j^{att}(r_i)),$$

and

$$\mathbf{F}^{obstacles}(r_i) = \sum_{o_k \in \mathcal{O}} (\mathbf{F}_k^{obs}(r_i) + \mathbf{F}_k^{obsat}(r_i)).$$

The primary component forces are defined as:

$$\mathbf{F}^{goal}(r_i) = \frac{\mathbf{v}_d \mathbf{e} - \mathbf{v}_i}{\tau}$$
$$\mathbf{F}_j^{soc}(r_i) = \alpha e^{(r_{ij} - d_{ij})/\beta} \mathbf{n}_{ij}$$
$$\mathbf{F}_k^{obs}(r_i) = A e^{(r_i - d_{ik})/B} \mathbf{n}_{ik}$$

where $\mathbf{v}_d$ is the agent's desired speed, $\mathbf{e}$ is the direction to the agent's current goal, τ is its reaction time, $r_{ij} = r_i + r_j$, $d_{ij} = \mathbf{x}_i - \mathbf{x}_j$, α is a social scaling constant, β is the agent's personal space dropoff constant, $\mathbf{n}_{ij}$ is the normalized direction between r_i and r_j, A is an obstacle scaling constant, B is the obstacle distance dropoff constant, and $\mathbf{n}_{ik}$ is the vector from r_i to the nearest point on o_k to r_i. The attractive forces, $\mathbf{F}_j^{att}(r_i)$ and $\mathbf{F}_j^{obsat}(r_i)$, take the same form as their repulsive counterparts except that they have

different values for the constants and an opposite sign for the scaling term. Since these forces are largely based on the social force model of Helbing et al [HBW03], we refer the reader to this for more details.

Contact Handling: While the repulsive forces generally allow agents to avoid contacts, they do not give guarantee of a collision-free motion.. Social forces with a large magnitude can cause the agents to move directly towards each other. Moreover, network communication latencies can result in incomplete information about the location of other agents, objects or dynamic obstacles in the scene. In these situations, contacts between the agents must be resolved. We assume that the central server is ultimately in charge of resolving intersections between both agents and obstacles. Since latency is a factor even in the absence of virtual agents, it enforces hard constraints on agent positions to prevent intersection and returns that information to the clients.

In order to reduce the impact of future collisions with the currently colliding obstacle or agent, additional forces are applied to each agent in collision. A pushing force, $\mathbf{F}_j^{push}(r_i)$, acts to force a separation between agents and a frictional force, $\mathbf{F}_j^{fric}(r_i)$ simulates the act of slowing down due to a collision. Unlike the associated repulsive force, these forces are only applied when an intersection has taken place.

For agent r_i and an intersecting agent r_j, the following forces are added to Eq 6.10:

$$\mathbf{F}_j^{push}(r_i) = \kappa(r_{ij} - d_{ij})\mathbf{n}_{ij}$$
$$\mathbf{F}_j^{fric}(r_i) = \lambda|\mathbf{F}_j^{push}(r_i)|\mathbf{t}_{ij}$$

where κ is a pushing spring constant, λ is a sliding friction constant, and $\mathbf{t}_{ij}$ is the tangent vector to $\mathbf{n}_{ij}$. Contacts with obstacles are treated in a similar fashion.

Network Modeling: Our virtual agent model is computed based on the information on a client database. This introduces certain amount of uncertainty in the position and velocity of other agents and also in other moving obstacles in two ways. First, some

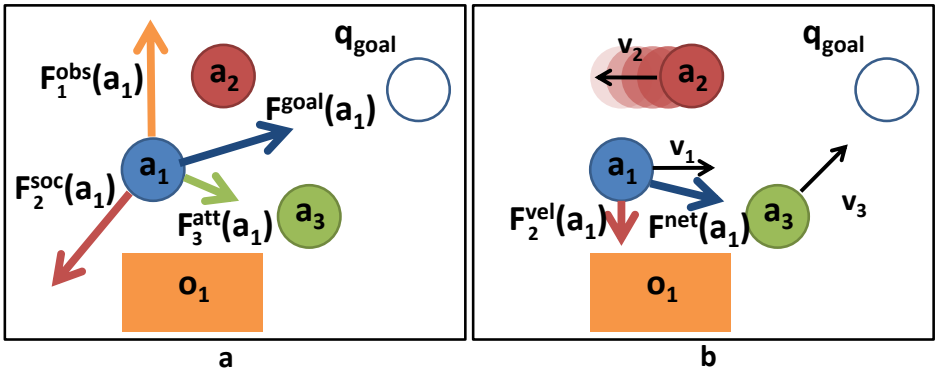

Figure 6.12: Local Navigation via social forces: The colors of the arrows correspond to the specific object that generates the social force. (a) Agent r_1 is acted upon by social repulsive force $\mathbf{F}_2^{soc}(r_1)$ from r_2, attractive force $\mathbf{F}_3^{att}(r_1)$ from r_3, repulsive obstacle force $\mathbf{F}_1^{obs}(r_1)$ from o_1, and goal force $\mathbf{F}^{goal}(r_1)$. Repulsive and attractive forces encourage the agent to move toward or away from agents and obstacles, respectively. (b) An additional velocity bias force $\mathbf{F}_2^{vel}(r_1)$ is computed to account for agent r_2's velocity during inaccurate sensing. By assuming a linear trajectory over a short period of time, we can help to reduce the impact of network latency. No velocity bias force is computed for r_3 since it is heading in the same direction as r_1. The final net force $\mathbf{F}^{net}(r_i)$ reflects the sum of all the forces and serves as the agent's next heading.

agents or obstacles may not have yet been sensed, i.e. their base information has not yet been received by the client from the server. Second, due to low network bandwidth or high network latency, the updated positions and velocities are not received in time by both the server and client.

To account for these issues, we augment the social force model with a velocity bias force, $\mathbf{F}_j^{vel}(r_i)$, to help reduce the impact of latency. Let $V_j = SSV(r_j, \mathbf{v}_j, t_{bias})$ be the spherical swept volume as agent r_j travels along heading $\mathbf{v}_j/||\mathbf{v}_j||$ for time t_{bias}. Then,

$$\mathbf{F}_j^{vel}(r_i) = \gamma e^{(r_i - d(V_j, \mathbf{p}i)/\epsilon)} \mathbf{n}_{(i,V_j)},$$

where γ is a velocity bias scaling factor, ϵ is a bias dropoff distance, and $d(.,.)$ and $n(.,.)$ are the distance and normal direction between the r_i and volume V_j. Intuitively, the force naively assumes that agent r_j will proceed in its current direction for a fixed period

of time and generates a force field around the volume swept of r_j along that heading (See Fig. 6.12(b)). As a result, other agents will tend to move away from their current direction of motion. Combined with $\mathbf{F}_j^{soc}(r_i)$, the resulting force is strongest closest to r_j and reduces further from the swept volume. The net effect of this force is that it gives virtual agents a way to estimate where other agents will go based on the information is available, and thus reduces the impact of latency or bandwidth limitations. Behaviorally, this force has additional benefits in that the agents will tend to slow down and avoid each other as they move toward their goals.

6.5 Motion planning for multi-agent systems

In our physics-based sampling framework, we have decomposed the multi-robot motion planning problem into two portions: a global replanning phase and a local navigation or local avoidance phase. Since both of the these phases are backed by physics-based sampling cores, they will each evolve over time with different objectives. In this section, we detail these planning components and show how they are both used for motion planning of many agents.

6.5.1 Motion Planning with RDRs

The reactive deforming roadmap (RDR) is a dynamic data structure for motion planning of multiple agents, or in dynamic environments. To this point, we have described the data structure, its properties, and how it evolves over time. In this section, we show the various ways in which the RDR can be used for multi-agent planning. This includes providing each agent with its own RDR and also providing a means for a shared RDR when the robots have the same geometry.

Simple Approach

Performing motion planning using RDRs is similar to other roadmap-based methods. For multiple robots, each robot uses its own RDR. In this way, it treats all other robots as dynamic obstacles.

We first connect the each robot's starting and final configurations to the RDR by finding the nearest dynamic milestone to each configuration and attach them via a reactive link. Next, an A* path search algorithm is used to compute a path. As the path deforms in response to the dynamic obstacles, the robot's motion follows the deformed path naturally.

It is possible that a path can become invalidated during execution. This occurs when a link has been removed due to obstacle motion. When these events occur, first we determine if the robot is still connected to the roadmap, and then replanning is performed via an A* search. If no path is found, links are added as previously mentioned.

Global RDR Approach

In certain circumstances with multiple robots, each robot is identical to the other robots. When each robot is entirely rigid (such that $\mathcal{C} \subset \mathbb{R}^2 \times S$ or $\mathcal{C} \subset \mathbb{R}^3 \times SO_3$), we can exploit their similarity to improve th performance of RDRs. Rather than maintaining a separate RDR for each of these robots, we can use a single RDR (see Fig. 6.13). In this case, fewer particles are needed, resulting in a better overall performance. To note, this method will also hold for certain articulated bodies but it does not easily generalize to higher-dimensional configuration spaces.

In order to use this single RDR for multiple robots, we make a few modifications to our basic algorithm. Robots need to be able to apply forces to the RDR and additional coordination is required.

Robot Force

Paths for separate robots can be disjoint but nearby each other. In this case, a robot

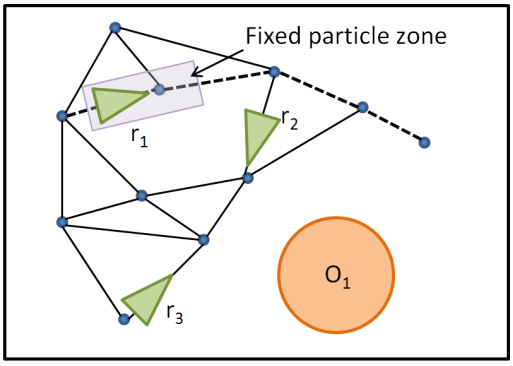

Figure 6.13: Multiple robots on the RDR: For proper avoidance, each robot applies forces on the RDR except on the particles that are in its fixed particle zone.

would need to apply forces to the single RDR and to the the other robot to ensure avoidance with other robots. The force a robot applies on the roadmap is the same as that for an obstacle (Eqn. 6.3). One caveat is that a robot would also by applying forces to its own path along the roadmap, possibly causing unnecessary motion in this direction. To prevent this, we define a "fixed particle zone" for a robot i with path P as:

$$\mathcal{Z} = \{\mathbf{p} | (\mathbf{p} \in \mathcal{P}, d(\mathbf{x}, \mathcal{P}) < \epsilon_p) \wedge d(\mathbf{x}, \mathbf{q}(t)) < \epsilon_r\},$$

where $\mathbf{x}$ is the position of particle $\mathbf{p}$, $d(\mathbf{x}, \mathcal{P})$ is the distance from $\mathbf{x}$ to the path link, ϵ_p is a path distance threshold, and ϵ_r is a robot distance threshold. This region may enclose the particles either on or near the path link which the robot is currently traversing, as well as those near the robot (See Fig. 6.13).

Coordination

With multiple robots on a single RDR, it may be necessary to coordinate their motion to prevent deadlock or collisions with other robots.

Coordination is handled through altering agent velocities and creating additional routes. The main issues arise when the robots arrive at a milestone at the same time

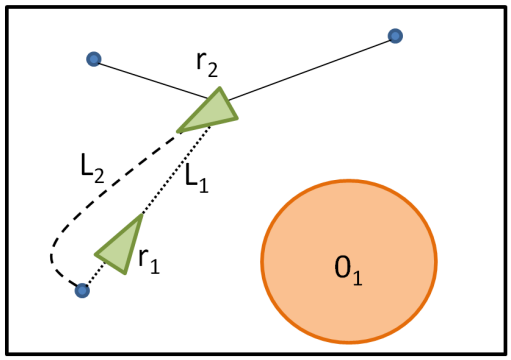

Figure 6.14: Adding additional links between milestones: As the robot r_2 approaches a link occupied by r_1, an additional link L_2 is added for r_2 to traverse.

and a robot is trying to use a link that is already occupied by another robot. When robots are near each other, one robot further away from the milestone is instructed to slow down until the robot ahead is sufficiently clear of the milestone and other robots. To handle link contention, it is possible to make use of the reactive properties of the link itself. Rather than coordinating which robot uses the link at a given time, an additional link is added between the milestones as a reactive link (See Fig. 6.14). This additional link will react to the motion of the other robot that is already on the link and vice versa for the existing link. This approach also provides an effective mechanism for the robots to avoid each other.

It should be noted that these solutions have their limitations. It is necessary to be careful about how the agent velocities are altered in order to not create a deadlock situation or a situation in which a single robot is stalled and cannot proceed. The additional link is not practical when there is insufficient space for two separate robots to pass through each other safely.

6.5.2 Integrated local and global planning

In this section we describe our approach to compute collision-free paths, particularly for a large number of agents with coordination constraints using RDRs. In order to

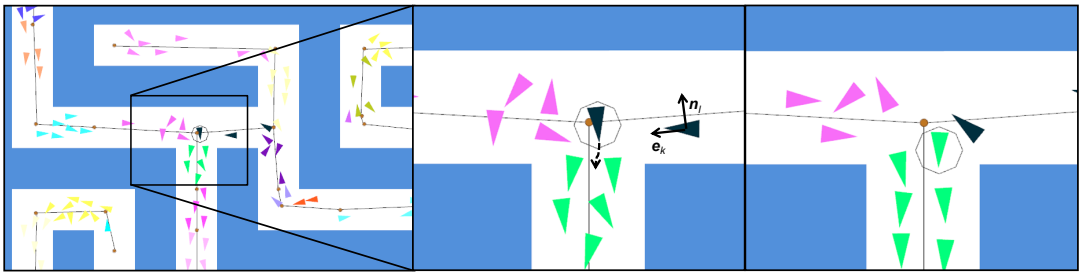

Figure 6.15: Agent clusters for fast proximity computation: (Left) Each color corresponds to a unique cluster of nearby agents. Link band boundaries are shown as dashed orange lines. (Center) Zoomed-in view of the clusters in the boxed region, with a single agent highlighted and circled. The roadmap force components are shown for another agent with the central milestone as next intermediate goal. $\mathbf{n}_l$ denotes the unit normal vector from agent to the link l. $\mathbf{e}_k$ denotes the unit vector direction towards next goal. (Right) Cluster updates are highlighted, as an agent crosses a link band boundary.

allow the agents to occupy the entire free space for navigation, we relax the restriction of constraining the agent's position to the links of the roadmap. Rather, we introduce *link bands* defined by each link of RDR, and use them for path planning as well as local dynamics simulation of each agent. The architecture for our complete navigation system is given in Fig.6.16. In this section we introduce link bands (Section 6.5.2), use them for global path planning (Section 6.5.2), for defining agent clusters (Section 6.5.2) and for local dynamics computation (Section 6.5.2).

Link Bands

The *link band* associated with a link of the roadmap is the region of free space that is closer to that link than to any other link on the roadmap (Fig.6.9(a)). Formally, the link band of a link l_i is given by $B(i) = \mathcal{V}(l_i|\mathcal{L}) \cap \mathcal{F}$. The width of the link band is the minimum clearance from the link to the obstacles in the environment, $B_w(i) = \min_{\mathbf{o}_j \in \mathbf{O}}(d(l_i, \mathbf{o}_j))$. The link bands form a spatial partitioning based on proximity of a point in free space to the nearest link. Each link band specifies a collision free zone in a well-defined neighborhood of each link of the roadmap. Additionally, link bands

provide the nearest link, which is required for path search (Section 6.5.2), distance to the roadmap that is used to compute guiding forces to advance the agent along the path (Section 6.5.2), and a spatial decomposition used for agent clustering (Section 6.5.2).

In a dynamic environment, an agent might be required to recompute its path since its original path could become invalid or otherwise change. We use link bands to detect such events. In particular we keep track of an agent's motion across each link band boundary. We classify points on a link band boundary into *milestone boundary* and *intermediate boundary* (see Fig. 6.9(b)). A point on the milestone boundary belongs to two adjacent link bands whose links share a common milestone. On the other hand, the intermediate boundary is all points on the boundary that do not belong to the milestone boundary. Formally, the milestone boundary of a link l_i, $B_m(i) = B(i) \cap B(j),\ \forall l_i \cap l_j \neq \emptyset$, and the intermediate boundary of a link l_i, $B_i(i) = \delta B(i) \setminus B_m(i)$. In the next section we show how the link bands are used for global path planning and to detect replanning events.

Path Planning

We use the RDR for global path computation for each agent. Since an agent is not constrained to the roadmap, we initially compute the link band it belongs to (i.e. the link band containing the position $\mathbf{x}_i$ of agent r_i). This link is set as the source link. Similarly the link band containing the goal position is computed and the corresponding link is set as the goal link. We assign a weight to each link as a combination of the link length, the reciprocal of the minimal link band width, and the agent density on the link as:

$$w(l_i) = \begin{cases} \infty & \text{if } 2B_w(i) < r_a, \\ \alpha|l_i| + \beta\frac{1}{B_w(i)} + \gamma\frac{n}{|l_i|B_w(i)} & \text{otherwise,} \end{cases}$$

where α, β, γ are constants,$|l_i|$ is the length of link l_i, and n is the number of agents on link $B(i)$. The third term approximates the agent density on the link and causes the agents to plan a path with less crowded regions of freespace. The relative values of the

constants are determined by the behavior characteristics of individual agents. A high relative value of α allows for the choice of shortest path, a high value of β avoids narrow passages and a high value of γ demonstrates preference of less crowded passages. In our experiments, we used a high value of α for slow agents, whereas aggressive agents are assigned a higher value of γ. Given a weighted roadmap, an A* graph search is performed to compute the minimum weight path from the source to goal link band, which is stored by the agent. Once the agent reaches its goal link band, it proceeds to its goal position within the band.

As the simulation progresses, the nearest link to an agent may change. We determine the events that require a path recomputation based on the boundary type, since these define whether the event was due to progress along a path or due to a path changing. Crossing a milestone boundary indicates agent motion along its path on the adaptive roadmap, and does not require a path recomputation. However, it is also possible for an agent to cross over the intermediate boundary. This typically occurs as a result of roadmap modification. In this case, it is possible that an alternative path to the goal exists and a path recomputation is required. However, we also allow the agent to move back to its previous path since this should be its path of least cost.

Local Dynamics Computation

To use link bands with our local navigation or social forces, we add a constraint that is defined over each link band.

We modify the social force model, to include a force term $\mathbf{F}^r$ that guides an agent along a link band on the roadmap. In addition, there is a repulsive force $\mathbf{F}^{soc}$ to the nearby agents, an attractive force $\mathbf{F}^{att}$ to simulate the joining behavior of groups, and a repulsive force from dynamic obstacles $\mathbf{F}^{obs}$. Let the point $\mathbf{p}$ (representing agent r_i)

belong to link band $B(k)$, then the force field at a point $\mathbf{p}$ is given as

$$\mathbf{F}(\mathbf{p}) = \sum_j \left[\mathbf{F}_j^{soc}(\mathbf{p}) + \mathbf{F}_j^{att}(\mathbf{p})\right] + \mathbf{F}_k^r(\mathbf{p}) + \Sigma_o \mathbf{F}_o^{obs}(\mathbf{p}), \tag{6.11}$$
$$r_j \in \mathcal{R}, j \neq i, o \in \mathbf{O}$$

where,

$$\begin{aligned}
\mathbf{F}_j^{soc}(\mathbf{p}) =& A_i \exp^{(2r_a - \|\mathbf{p}-\mathbf{x}_j\|)/B_i} \mathbf{n}_j(\mathbf{p}) \\
& \left(\lambda_i + (1-\lambda_i)\frac{1+\cos(\phi_j(\mathbf{p}))}{2}\right), \\
\mathbf{F}_j^{att}(\mathbf{p}) =& -C_j \mathbf{n}_j(\mathbf{p}) \\
\mathbf{F}_o^{obs}(\mathbf{p}) =& A_i \exp^{(r_a - d(\mathbf{p},o))/B_i} \mathbf{n}_o(\mathbf{p}) \\
& \left(\lambda_o + (1-\lambda_o)\frac{1+\cos(\phi_o(\mathbf{p}))}{2}\right) \\
\mathbf{F}_k^r(\mathbf{p}) =& \frac{\mathbf{v}_k^d(\mathbf{p}) - \mathbf{v}_i}{\tau_i} + D_i d^4(\mathbf{p}, l_k)\mathbf{n}_{l_k}(\mathbf{p})
\end{aligned}$$

where $\phi_j(\mathbf{p})$ is the angle between p and agent r_j, $\phi_o(\mathbf{p})$ is the angle between the nearest point on obstacle o and p, A_i and B_i denote interaction strength and range of repulsive interactions and C_j is the strength of attractive interaction, which are culture-dependent and individual parameters. λ_i reflects the anisotropic character of pedestrian interaction. The obstacle force field $\mathbf{F}^{obs}$ simulates the repulsion of the agents from other obstacles in the environment. Since the obstacles may be dynamic, we introduce an additional anisotropic term which biases the repulsive forces along the motion of the obstacles. This effect has also been modeled in other approaches by creating a 'discomfort zone' in front of dynamic obstacles [TCP06]. For efficient computation of repulsive force $\mathbf{F}^{soc}$ and obstacle force $\mathbf{F}^{obs}$, we compute forces to agents and obstacles within a radius B_i.

The roadmap force field $\mathbf{F}_k^r$ guides the agent along the link l_k. The link band $B(k)$

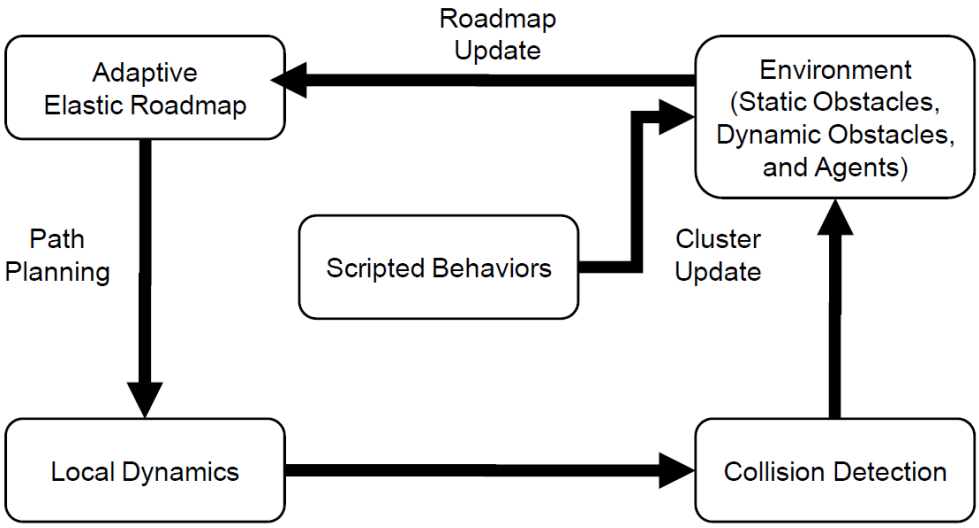

Figure 6.16: Navigation System: Given a description of the environment, an RDR is computed and updated. This is used in conjunction with our local dynamics model to simulate the motion of each agent.

is used to define the region which is used to compute the force field for l_k. The first term in $\mathbf{F}_k^r$ makes the agent achieve a desired velocity along the link, whereas the second term attracts the agent within the link band. $\mathbf{n}_{l_k}(\mathbf{p})$ is the unit normal from point $\mathbf{p}$ to the closest point on l_k, $d(\mathbf{p}, l_k)$ is the distance from $\mathbf{p}$ to l_k. The desired velocity $\mathbf{v}_k^d(\mathbf{p}) = v_{max}\mathbf{e}_k(\mathbf{p})$, where $\mathbf{e}_k(\mathbf{p})$ is a unit vector orthogonal to $\mathbf{n}_{l_k}(\mathbf{p})$. The direction of the normal is chosen such that $\mathbf{e}_k(\mathbf{p})$ points along the roadmap towards the next milestone on an agents path (see Figure 6.15(center)). D_i is a weighting term and the attractive force term keeps an agent inside the link band, reducing toggling across intermediate boundaries.

Resolving Agent Collisions

While our local dynamics model generally avoids intersections between the agents well, collisions are still possible on a generated physics-based sample. Large forces resulting from a dense group of agents or a relatively large simulation time step can lead to this situation. We employ a collision elimination scheme similar to that of Lakoba et al.

[LKF05].

For each intersecting agent r_{col}, we place it in a priority queue $\mathcal{Q}$, ordered by extent of intersection with both agents and obstacles. We determine the amount of intersection by the total penetration depth r_{col} with all other agents and obstacles. This can easily be determined by computing the distance to the agent's center and comparisons with the agent's radius or obstacles.

To remove overlaps, we first dequeue an agent r_i from $\mathcal{Q}$. If r_i intersects an obstacle, we first move it away from the obstacle until it is collision free. Then, for each other agent r_j that intersects r_i, we move it by a distance

$$\epsilon(r_i + r_j - ||\mathbf{x}_i - \mathbf{x}_j||)$$

away from r_i's center to remove any overlap, where $\epsilon > 1$ is a safety threshold, r is an agent's radius, and $\mathbf{x}$ is its position. The effect of ϵ is to move them into a safer, non-intersecting state. Note that for larger ϵ, it becomes more difficult to remove all of the collisions. Any agent which is intersecting with other agents as a result of this process are inserted into $\mathcal{Q}$. This process is repeated until no agents are in collision.

Intuitively, this position modification algorithm for agents treats them as they were in contact throughout some portion of a time step and that they remain in a near collision state until the end of timestep. We do note that in degenerate cases, such as a densely packed room, this process could take a long time and possibly result in an invalid state. However, in practice this collision resolution algorithm finishes very quickly and has not resulted in such a state in our benchmarks. This is likely due to the fact that local forces help prevent such cases when agents are nearby each other.

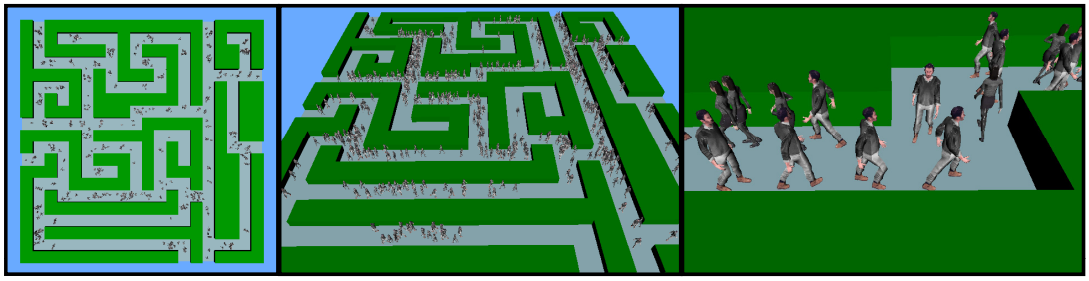

Figure 6.17: Maze environment: Left: Navigation of 500 virtual agents in a maze consisting of 8 entrance and 8 exit points. Center: Each agent computes an independent path to the nearest exit using adaptive roadmaps. Right: Our local dynamics simulation framework based on link bands captures emergent behavior of real crowds, such as forming lanes. We perform real-time navigation of 500 agents at over 200fps.

Agent Clustering for Fast Proximity Computation

Proximity computation is a key step in force computations within our local dynamics model. Prior work for local dynamics computation used a spatial hash function based on regular spatial decomposition [SGA+07]. While this provides a simple acceleration structure, it has some drawbacks primarily in scaling to large heterogeneous environments and increased storage overhead. However, in order to achieve optimum performance, the size of a grid cell in the spatial hash has to be carefully chosen for each environment.

In this section we introduce an acceleration structure based on link bands to efficiently compute proximity information for $\mathbf{F}^{soc}$ and $\mathbf{F}^{r}$ in Eq (6.11). Link bands provide a partitioning of the free space, and are used for global path computation for each agent. Therefore, we use the link band data structure to define a clustering of agents for efficient proximity computations.

Given the set of agents $\mathcal{R}$, we use simple techniques to cluster them into groups. An agent cluster is a set of agents that belong to same link band and are sufficiently close to each other. Formally, a set of agents $\mathcal{C}_i$ is a cluster on link l if $\forall r_j, r_k \in \mathcal{C}_i$,

$$\mathbf{x}_j, \mathbf{x}_k \in B(l)$$

and

$$d(r_j, r_k) \leq \delta_1$$

where δ_1 is a predefined constant governing the maximum size of a cluster. An example of agent clusters for a subset of the Maze environment(Fig. 6.17) is shown in Fig. 6.15.

The first condition (same link band $B(l)$) ensures that two agents within the same cluster are in close spatial proximity as defined by the roadmap. Furthermore, all agents within a cluster have one of two intermediate goals given by the two end milestones of the link l. Thus the roadmap force $\mathbf{F}_k^r$ for all agents r_k within a cluster $\mathcal{C}_i$ is defined by normal vector field $\mathbf{n}_l$ to same link l. The second condition (inter-agent distance) bounds the maximum spatial extent of a cluster. This is useful in heterogeneous environments which contain long links in the roadmap. The maximum size of a cluster δ_1 is proportional to the interaction range of repulsive interactions among agents (the term B_i in Eq (6.11)).

We now describe our algorithm for computing and updating the agent clusters. Initially for each agent r_i the nearest link l_k is computed and agent r_i is assigned to cluster $\mathcal{C}_k$ corresponding to link band $B(k)$. The axis-aligned bounding box (AABB) of a link band $B(k)$ is computed. If the diagonal of the AABB is greater than δ_1, then $\mathcal{C}_k$ is split into equal sized clusters $\mathcal{C}_{k1}, \mathcal{C}_{k2}, \ldots, \mathcal{C}_{kn}$ such that the diagonal of the AABB of each sub-cluster is less than δ_1.

As the simulation progresses, the agent clusters are updated incrementally. We update the agent set contained in each link band, as well as the clusters within each link band. The agent set is updated when an agent crosses a link band boundary. If a link band boundary is crossed, then the clusters are updated by moving the agent to the appropriate adjacent agent set. As the clusters are updated, the AABB of all agents within the cluster is updated. If the AABB size exceeds δ_1, then the clusters within the link band are recomputed.

Given the set of agent clusters, the proximity computation for computing the repulsive and roadmap forces is efficiently performed. For each cluster, the nearest link

is determined from the link band and used to compute the roadmap force $\mathbf{F}^r$. For an agent within a cluster $\mathcal{C}_i$, the set of adjacent clusters is computed using the link adjacency information. The repulsive force $\mathbf{F}^{soc}$ is computed against all agents in the set of adjacent clusters.

6.6 Results and analysis

In this section, we highlight our results on a variety of benchmarks involving multiple more complex robots or thousands of simple human agents in a complex scheme. For each method, we present results as well some discussion on the performance and comparisons approach.

6.6.1 Global navigation

RDR Results

In this section, we analyze the performance of our planner on some benchmarks and also highlight its performance.

Results

We highlight the performance of our planner on planar robots undergoing translational and rotational motion, as shown in Fig. 6.18 and in Fig 6.19. In this first benchmark, we demonstrate our algorithm on motion planning of 15 star-shaped robots (indicated in different colors) using reactive-deforming roadmaps. Each robot has 3-DOF (2T+R) and acts as a dynamic obstacle for the other 14 robots. The goal for each robot is represented as a thick point of same color in the figure. Thin gray segments represent the initial roadmap, thick colored segments denote the path for each robot. Fig. 6.18 (b) and (c) show the snapshots of two timesteps of our planning algorithm. Only the deforming paths are shown. Fig. 6.18 (d) shows the final configuration of the robots, the reactive-deforming roadmaps, and the corresponding paths. Our planner

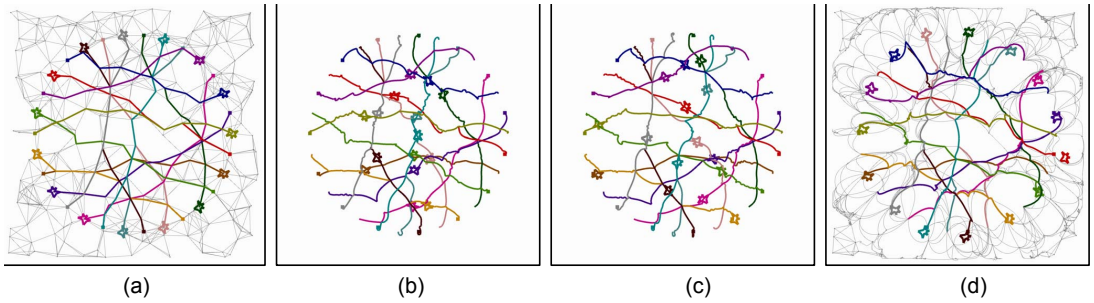

Figure 6.18: Multi-robot motion planning of 15 **star-shaped robots in different colors using reactive-deforming roadmaps**: (a) Initial configuration. Each robot has 3-DOF (2T+R) and acts as a dynamic obstacle for the other 14 robots. The goal for each robot is represented as a thick point of same color. Thin gray segments represent the initial roadmap, thick colored segments denote the path for each robot. (b)-(c) Two timesteps of our planning algorithm. Only the deforming paths are shown. (d) The final configuration showing the reactive-deforming roadmaps and the corresponding paths. Total number of simulation steps = $1,525$, average time per step = 10.6 msec on a 2.1GHz Pentium Core2 CPU.

takes a total of $1,525$ simulation steps, with an average time per step of 10.6 msec on a 2.1GHz Pentium Core2 CPU.

The second benchmark incorporates an application of multi-robot planning with dynamic obstacles to crowd simulation. Fig.6.19 shows an urban environment with 4 dynamic obstacles (cars) and 3-DoF human agents. Each agent has a unique goal near one of the buildings, which is updated as an agent reaches the goal. The dynamic obstacles invalidate links which forces agents to replan. As links become clear, the roadmap is repaired. For an environment with 100 agents, the average time per step is 11.5msec.

Performance

In this section, we analyze the performance of different components of our planner. Most of the time in RDR computation is spent in roadmap update, force computations, and roadmap maintenance. Roadmap update needs to be performed for each particle and the cost of this step is $O(n)$ for n particles. Internal forces are computed for each spring.

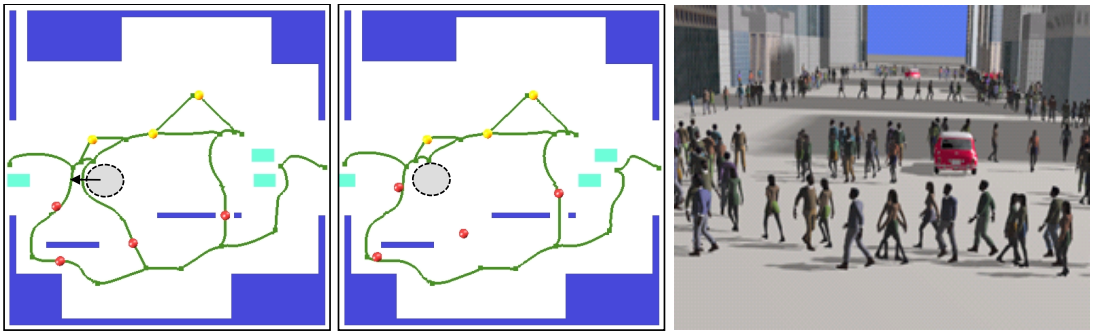

Figure 6.19: Application of N-body motion planning using reactive deforming roadmaps to complex crowd simulation with human agents and polygonal dynamic obstacles: *(a)-(b) An instructive example with* 4 *agents (in red) and goals (in yellow). The static obstacles are in dark blue and dynamic obstacles (cyan). The reactive deforming roadmap is shown with green links. The dynamic obstacles represent cars. As the highlighted car (circled) moves, the affected link in the roadmap is removed. (c) A real-time simulation of motion planning for* 100 *human agents and* 4 *cars in same environment. The average time for roadmap update and motion planning per frame* $= 11.5ms$ *on a* $2.1Ghz$ *Pentium Core2 CPU.*

Based on our existing construction algorithm, the number of springs is generally $O(n)$. If the initial roadmap is very dense, the number of springs can be a super-linear function of n. In the worse case, each obstacle can potentially apply a force to each particle, so for m obstacles, the worst case performance is $O(nm)$. In practice, if each particle only applies forces within a small or localized region, then the expected complexity of roadmap update is nearly linear in the number of particles. It is difficult to derive tight bounds on complexity of roadmap maintenance as it depends on the environment and the motion of the obstacles. The complexity of the algorithms for adding or removing links is also linear in the number of springs. removal methods is on the same order as that of the number of springs. Even though we perform a fixed number of link and milestone additions, the cost of each update is at least $O(n)$, for a nearest neighbor search. So, the total cost is at most $O(n^2 + nm)$. Fortunately, in any practical case, the cost much lower than this.

Analysis and Conclusions

We have presented a physically-based, adaptive roadmap representation that retracts and changes its topology as a function of the dynamic environment. The RDR can be used to plan the motion of a single robot or multiple robots among dynamic obstacles. Our formulation offers several advantages over existing approaches. Most of replanning algorithms remove the links as soon as they are invalidated. Since our links can retract based on the motion of the obstacles or other robots, they are not invalidated as frequently. Thus, the planner removes relatively fewer links and performs fewer milestone and link additions into the initial roadmap. In addition, by maintaining relatively longer links, in terms of path length, roadmap connectivity is not updated as frequently. Moreover, our formulation can easily accommodate moving goals of the robots or changes in the path topology due to dynamic obstacles. It is relatively simple to implement our algorithm using a mass-spring simulation framework, on top of a roadmap-based motion planner.

Our approach also has some limitations. Since we do not make any assumptions on the motion of the obstacles, the generated motion is susceptible to leaving the robot in a state where a collision is inevitable. It is also difficult to give any guarantees on the optimality of the computed paths. If the environment consists of heterogeneous or different robots, we need to maintain and update a separate RDR for each robot which can be prohibitively expensive. Even when there are classes of robots which can occupy a single RDR, the RDR overhead can be the bottleneck in planning.

Currently, we have tested the performance of our planner on complex scenarios with tens of planar 3-DOF robots. As part of future work, we plan to evaluate its performance on more complex scenarios with high-DOF robots (so, the particles would exist in $\mathcal{C}$ rather than in the workspace). Improved coordination could lessen the number of extra lanes, and also improve performance. Multiresolution methods would also help improve the performance. Finally, we want to add a dynamics to the agent motion for more realistic behavior.

Particle sampling

Different sampling schemes have a impact on the overall performance and behavior of the global RDR planner. In our tests, we have observed up to a 3 times speedup in overall roadmap performance. This largely comes from far fewer roadmap particles to be simulated.

There is a clear visual distinction between the different methods as well (See Fig. 6.20). Uniform sampling causes unwanted oscillation along a link due to elastic forces, compression between the link's milestones, and oversampling. Depending on the degree of oscillation, this may effect the agent's goal force. On the other hand, the effect of APSL is to increase link elasticity near obstacles, but also to shorten the link farther away from obstacles. This shortening removes oscillation along a link, helps to smooth agents' goal forces, and greatly improves roadmap update timings. Additionally, the resulting link will have similar sampling density as uniform sampling near obstacles, where more samples are needed.

An additional advantage of the APSL algorithm is that it can be used to reduce the complexity of the input roadmaps, both in terms of number of links and milestones. The APSL algorithm generalizes to higher dimensions (in a manner similar to that proposed by Quinlan et al [QK93]), so it can be applied to higher dimensional RDRs as well. In many cases, the initial roadmap may be overly sampled due to our selection of Voronoi vertices. We can apply the APSL algorithm to the input roadmap by inspecting all milestones with a valence of 2, or those that are only connected to exactly two other milestones. If the milestones connected to this milestone satisfy the APSL safety criterion, then the milestone can be removed (See Fig 6.21). We have applied this to our input roadmaps. The results of this are summarized in Table 6.1. Note that in the case of the maze and city demos, the roadmaps were already relatively clean so this approach did not help much. However, in the case of the tradeshow, the size of our initial roadmap was reduced by a factor of 3.

Demo	Sampling	**Num Milestones**	**Num Links**
Maze	Uniform	99	103
Maze	APSL	94	98
Tradeshow	Uniform	3734	3896
Tradeshow	APSL	1154	1316
City	Uniform	2321	3216
City	APSL	2189	3084

Table 6.1: Uniform Sampling vs. APSL in Roadmap Cleanup. This shows the reduction in input roadmap size based on the APSL sampling metric.

We also note an important consideration when using the APSL sampling method. It tends to oversample when the link is near an obstacle since the safety region for a particle is relatively small. Other approaches, such as those based on curvature or basis functions, may yield coarser samplings in these situations. However, most of these other approaches require the tuning of some parameter which may be non-intuitive based on the situation or may be relatively complex to implement. On this other hand, this approach works well in a wide variety of situations and is simple to implement with the RDR framework and data structures for efficient obstacle proximity queries. Plus, through the application of forces, the links will tend to move away from obstacles, which, over time, will result in the coarsest sampling a link allows based on the obstacle configuration and the APSL sampling criterion.

6.6.2 Local navigation

Generalized Social Potential Field Results

We have completed a preliminary implementation of Generalized Social Potential Fields and tested its performance and scalability on a workstation computer with 2.66 GHz Intel X5355 Xeon CPUs and 3 GB of main memory. To understand the effectiveness of the approach, we have applied it to a variety of local coordination benchmarks.

Benchmarks

We have tested our approach on three different benchmarks, each designed to test different aspects of the coordination technique.

- **Antipodal**: In this benchmark, a fixed number of random robots are placed around the circumference of a circle. Each robot must move across the circle and toward the opposing orientation. This benchmark uses social potential fields between all bodies as well as an attractive force to have the robots cross the circle to its antipodal position. Planning took 125 s to complete the task for 16 robots with an average step time of 8ms. See Fig 6.4.

- **Letters**: In the letters benchmark, 33 randomly placed robots must navigate to form the letters **ICRA'09**. Like the Antipodal demo, only social repulsive and attractive forces are used. The benchmark took 273 s to complete, with an average step time of 5 ms. See Fig 6.23.

- **Guarding and Escorting**: In this benchmark, an important robot is trying to navigate to a position across the environment. Several guard robots form a formation around the important robot and then escort it to its goal while trying to remain in formation. This benchmark has 35 aggressive robots, and uses formation constraints as well as social repulsive and attractive potentials. Planning completed in 133 s with an average step time of 42 ms. See Fig 6.1.

The preliminary results of our approach are very promising. In a mostly unoptimized implementation, we were able to generate collision-free, coordinated motion plans in a relatively short amount of time. Again, there is no global planner present so robots will simply attempt to take the straight-line path to the goal even if the free space is more constrained than other paths. Even though this results in a great deal of traffic, they are frequently able to still reach their goals. Table 6.2 highlights the run-time performance for our benchmarks. We note that in all cases, as the number of robots scales from 16 to 200, the average step time remains at a few to tens of milliseconds (ms) which shows

Demo	Num Robots	**Forces (s)**	**Update (s)**	Avg Step Time (s)	Total Time (s)
Antipodal	16	0.0078	0.00001	0.00781	125.7
Antipodal	32	0.0231	0.00003	0.02313	358.7
Antipodal	48	0.0303	0.00004	0.03034	637.2
Antipodal	64	0.0464	0.00005	0.04645	881.1
Antipodal	80	0.0290	0.00007	0.02907	939.3
Antipodal	100	0.0300	0.00008	0.03008	969.4
Antipodal	200	0.0752	0.00017	0.07537	3748.8
Guarding	39	0.0419	0.00003	0.04193	132.9
Letters	33	0.0053	0.00002	0.00532	273.8

Table 6.2: Planning times for benchmarks: This table shows the timing for each of our benchmarks. Forces (force computation, constraint updates), update (motion equation integration), and step time are averages over the entire planning run. Total time is the amount of time it took to find a planning solution.

that it scales well with the number of robots. For the Antipodal benchmark, there was a drop in the average time as we went beyond 64 robots. This can be attributed to the fact that the size of the circle was increased to accommodate the robots. When the robots are further apart, fewer interactions occur. Fig. 6.22 confirms the scalability with the number of robots, as it shows the average step time as the number of robots increases. The total planning times can be misleading, since these are also dependent on how far each robot must travel. The average step time is a better indicator of the actual performance since total planning time is a function of average step time and average distance traveled per step.

Analysis

There are numerous factors which play a role in the theoretical run-time performance of our approach. The most dominant feature would be the number of robots, N, and their discretization. Each sample on a robot r_i could potentially visit every other sample on every other robot and obstacle at each time step. Thus, in the worst case, our time step complexity approaches $O(M^2)$, for $M > N$ samples. However, by exploiting the fall-off distance parameter in Eq. 6.2, this is typically not the case. Given the maximum search distance of $c\beta$ (as defined in Sec. 6.4.1), each sample visits a finite, bounded

number of other samples within that region. This reduces our run-time complexity to $O(M)$, but it is important to note that while bounded, the maximum number of samples a single sample could visit could still be a large number. In our tests, we did see cases where too fine of a sampling severely degraded performance but these are usually eliminated by the previously mentioned optimizations.

Like other simulations with several parameters, the values of these parameters played a role in the success of the approach. However, since many of the parameters in our approach have real-world interpretations (distance, force in Newtons, etc), we found that giving them realistic values often resulted in successful and stable planning.

Comparison and Limitations

In this section, we provide a qualitative comparison of our work with the large body of work on multi-robot motion planning and note some limitations of our approach.

Comparisons

Like other decentralized methods, our work relaxes completeness for efficiency. Thus, compared to centralized approaches such as Probabilistic Roadmap Methods (PRM) [KSLO96], Rapidly-exploring Random Trees [LK00], or other variations on these methods, our approach scales much better in the number of robots, but does not offer completeness or convergence. Furthermore, additional coordination constraints can greatly restrict the search space, making it more difficult for centralized approaches to find suitable paths. As a result, these methods become impractical for systems with a large number of robots, such as in the scenarios we are attempting to solve.

Of the decentralized methods, we can group them into several categories: incremental planning, decoupled planning and coordination, dynamic replanning, path modification, and potential fields.

Incremental planning methods first determine a priority for each robot. Then, robots plan in the priority order such that they avoid collisions with robots who have already

found a path. Variations on this interleave centralized planning with incremental to improve reliability [SI06]. Since each robot is completely independent other robots, certain coordination constraints such as distance constraints are more difficult to satisfy than in our approach. In contrast, the social potential model in our approach accommodates close proximity between robots and scales well, as the number of robots increases. However, since incremental planning uses a planner for each individual robot, it supports a wider range of robots than is currently supported by our approach and can navigate in environments where a global path is required.

Decoupled planning and coordination is a variation on incremental planning and decomposes multi-robot planning into two phases. In the first phase, paths are planned without consideration of other robots. Then, a coordination phase adjusts the robot's velocity along their path so that no collisions occur [KZ86, SLL99, PA05]. Similarly, graph representations such as coordination graphs, help to combine planned paths to avoid collisions [LGP05]. This approach shares many of the same qualities as incremental planning. Particularly, stiffer coordination constraints increase the difficulty and lower the chance of success as compared to our approach.

Dynamic replanning methods wait until the current path becomes invalidated, and then uses any remaining valid information to determine a new plan [Ste95, KM04, ZKB07, BTK07]. Some replanning methods have been popular choices for higher-DOF multi-robot problems. The key drawback of replanning methods is that the cost of replanning can be high, particularly for a large group of robots or with additional coordination constraints such as distance constraints. Thus, they will not always scale as well as potential field methods. Path modification approaches [QK93, BK02, YB06] attempt to bend or deform the path around moving obstacles, rather than allow them to become invalidated. However, in general these methods cannot handle global changes in connectivity which commonly occur in multiple robot problems.

Various planners also make different assumptions for sensing or exchange of infor-

mation. Communication is a popular choice, and was successfully used for a replanning approach coupled with decentralized planning of multiple vehicles [BTK07]. Several communication configurations assume that sufficiently precise position and trajectory information can be passed to robots within a local region such that the position of nearby robots is known at any point in time as long as the robot follows that trajectory. This works well and provides essentially the same information as needed by SPF. On the other hand, sensing of position information as in SPF can also account for unexpected changes to the paths of other robots or other inaccuracies in sensing.

As discussed earlier, potential field methods [Kha85, War90, RW99, GC02] rapidly update a navigation function to adjust to moving obstacles and robots. Constraint-based Motion Planning (CBMP) [GL02] also falls into this category with iterative methods to satisfy additional constraints. We propose a generalization of the social potential field, which has assisted in generating collision-free motion in crowded settings while offering real-time performance.

Limitations

There are some limitations to our work. There is no guarantee that a robot will find a path even if one exists. However, this has not often been the case in our experiments. While the excitation helps robots escape a local minimum, it is not guaranteed to always work. For instance, a relatively small and light robot will not be able to move past several larger, heavier robots. Since we do not consider points beyond our distance criterion, a fast moving robot or obstacle might collide with a robot before it could react. While dynamic constraints are inherently simulated, they are not explicitly handled and thus it is possible for a robot to reach an inevitable collision state. However, it should be noted that by either bounding relative velocities or by adjusting the social force parameters to give a robot sufficient time to react, these cases can frequently be avoided.

6.6.3 Physics-based planning for multiple agents

Multi-agent planning results

In this section we describe the implementation of our multi-agent navigation system and highlight its performance on various environments. We have implemented our algorithm on a PC running Windows Vista Enterprise operating system, with a two 2.66Ghz Quad-Core Xeon CPUs, 3GB memory and an NVIDIA Quadro NVS 285 GPU. We used Visual C++ 8 programming language and OpenGL as the graphics API for rendering the environments. Since the robots are in these tests have two translational DOFs and a single rotational DOF, the initial Reactive Deformable Roadmap (RDR) for an environment is initialized by computing the Voronoi diagram of the static obstacles in the environment. This computation helps initialize the roadmap with links that are optimally clear of obstacles when the simulation begins. To simulate particle dynamics of the agents, we used a semi-implicit Verlet integrator [Jak01].

Proximity computations to dynamic obstacles are accelerated using a spatial hash data structure to identify the nearby objects. In addition, to accelerate proximity computations to static obstacles, we precompute a discretized Voronoi diagram of the obstacles using the GPUs [SGGM06]. The discrete Voronoi diagram provides proximity information to the nearest obstacle. To get the set of all obstacles within a given radius r, we scan the discrete Voronoi diagram (and distance field) within a window of size $r \times r$ and check if the distance value at the discrete samples is less than r. Thus the proximity computation to static obstacles is reduced to a small number of table lookups.

Benchmarks

We demonstrate our system on three complex scenarios.

- **Maze**: The maze scenario considers the case of multiple agents navigating a maze. The maze has 8 entry and 8 exit locations, and 1288 polygons. The initial roadmap consists of 113 milestones and 281 links. By using RDR, they have complete

knowledge of how to navigate the maze despite its complexity and thus are able to quickly move toward their goals. See Fig. 6.17.

- **Tradeshow**: The tradeshow scenario is an indoor environment of an exhibit hall in a trade show. The exhibit consists of 511 booths and 110K polygons. The initial roadmap consists of 3996 links and 5996 milestones. Numerous agents walk around and visit multiple booths. The goals for each agent are updated as the agent arrives at a booth. Some agents stop when they reach their goal in order to simulate observation of a particular point of interest. After a certain amount of time, the agents will resume walking toward their next goal. Also, certain booths have fixed agents whose orientation changes according to passing agents. See Fig. 6.30.

- **City**: The city scenario is an outdoor scene consisting of multiple city blocks. The model consists of 924 buildings and 235K triangles. The initial roadmap for the environment consists of 4K links. The environment also consists of 50 moving cars as dynamic obstacles. As the cars move through the urban setting, links on the path deform around the obstacles and get invalidated. We add a higher potential in front of the cars along their direction of motion, which decreases the probability of the agents from selecting paths in front of moving obstacles. The environment is populated with a non-uniform density of agents moving along the side walks or crossing the streets. Additional behavior characteristics of each agent are assigned at run-time. These individualized behaviors includes updating the goals, varying the maximum speed, and changing interaction range of the agents. See Fig. 6.29.

We highlight the performance of our crowd simulation algorithm on the complex benchmarks. Our approach can perform real-time simulation of crowds with up to $1,000$ independent agents at interactive rates using APSL sampling and agent clustering – ranging from 54 to 307 frames per second, depending on the scene complexity and the

Demo	Agents	Sim	Path Search	RDR Update	Total Time
Maze	500	3.01	0.006	0.24	3.256
Maze	1000	7.98	0.013	0.26	8.37
Trade Show	500	2.51	1.02	2.21	5.74
Trade Show	1000	6.52	2.21	2.22	10.95
City	500	3.18	2.03	5.88	11.09
City	1000	8.11	3.85	6.07	18.03

Table 6.3: Performance on each scenario. Timings reported here are the average simulation time per frame (step) broken down into the time for simulating local dynamics (**Sim**), performing path search (**Path Search**), and updating RDR on the fly (**RDR Update**). All timings are in milliseconds.

crowd density. The performance of our algorithm in the environments (with different complexity) and varying number of agents is highlighted in Table 6.3.

We further highlight our results by comparison with prior approaches. Figures 6.24, 6.25, and 6.26 break down our algorithm based on which novel features it includes. As can be seen, both APSL sampling and agent clustering improve the total time per simulation step. We note that in the maze scenario, very little path search and RDR update improvement is visible. This is largely because the roadmap was already fairly optimized to begin with.

Analysis and Discussion

In this section we analyze the time complexity of stages of our algorithm, and provide a qualitative comparison with prior work.

Analysis Performance of our approach depends on a number of factors. At each time-step, RDR's complexity is $O(|M| + |E|)$, or linear in the number of particles and edges. The tasks per timestep include force computations, numerical integration, as well as path search and roadmap maintenance. Agent motion also depends linearly in the number of agents, but each agent also performs a path search, thus making the agent portion of computation complexity $O(N + N|E|lg|E|)$. In all of these cases, the performance scales linearly with the number of agents or the complexity of the roadmap.

Furthermore, particle simulation is easily parallelizable, so this approach should be able to scale well to a large number of agents.

Comparison and Limitations

We compare some of the features of our approach with prior algorithm and highlight some of its limitations. Our adaptive roadmap based agent navigation algorithm is designed to perform real-time global navigation for a large number (e.g. hundreds or thousands) of independent or *heterogeneous* agents, each with different goals. We also take into account dynamic obstacles and the local dynamics among the agents. RDR continuously adapts to dynamic obstacles and is used to compute collision-free paths in dynamic environments. As compared to local or potential field methods, RDR can compute a global path for each agent. In addition, we use elastic bands and link bands to augment the local dynamics and resolve collisions among multiple agents. Due to lazy and incremental updates to the roadmap and efficient computation of guiding-path forces with link bands, this approach can scale well to hundreds or thousands of agents.
Comparisons: Our work is complementary to several existing works on crowd simulation and multi-agent planning. Continuum Crowds [TCP06] targets navigation for a small umber $(2-5)$ groups of human agents, where each group consists of many (up to thousands) agent with *identical* goals and behavior characteristics. Moreover, this approach uses local collision avoidance and its accuracy is governed by the underlying grid resolution. This approach has not been shown to extend well to a large number of groups or when there are challenging narrow passages in the free space, as shown in our maze and trade show benchmarks.

Graph based approaches [PLT05, LD04, LG07] use proximity graphs to capture the connectivity of the navigable space and use it for agent coordination. However, the navigation graphs are precomputed and thus are mainly restricted to static environments. Multi-agent Navigation Graphs [SAC+07] efficiently compute dynamic navigation graphs for simple agents. However, this approach is limited to a few hundred agents and does

not scale with the number of agents. It cannot guarantees coherent and smooth paths. Corridor Maps [GO07] use similar proximity ideas as link bands to define navigable space, and can adapt to dynamic obstacles. However, corridor maps cannot easily handle dynamic topology of the roadmap and model emergent behaviors like agents following each other in lanes. Local agent-based and potential-field methods [Rey87, ST05] perform well for a large number of agents and exhibit interesting crowd-like behaviors, but cannot provide same guarantees in path finding as global approaches.

Limitations: Our approach has some limitations. Our current implementation addresses collision-free navigation of a large number of 3-DoF agents, therefore our work does not produce realistic motion in situations where each human is modeled as a high DoF avatar. Although RDR uses a global roadmap at each given time step for path computation, the local dynamics formulation to update the links can potentially result in an agent getting stuck in a local minimum across space-time. In other words, our work may not be able to provide completeness on the existence of a collision-free path for each agent in all environments. Furthermore, we currently treat each agent as an individual agent and do not exploit all the behavior-related characteristics of real crowds such as grouping.

Autonomous agents in online virtual worlds

In this section we describe our implementation, show results of our virtual agent model, and discuss possible issues with the approach.

Implementation

A preliminary implementation of his approach has been developed for Second Life, based on the LibSecondlife (LibSL) framework. LibSL is used to reverse engineer the SL network protocol. With this, it is possible to create a SL client session without using the official SL viewer resulting in avatars that appear as full clients to the server. It includes data structures for avatars and objects as well as events for when data has been

updated. Support for simultaneously controlling multiple avatars also exists. While the number of avatars per computer is limited, additional computers can be used to host additional clients. In this way, the algorithm can scale easily to multiple computers as long as there exists a single host server. The avatar motion control model along with the local agent dynamics has been implemented on top of this framework. A forward Euler integrator is used to estimate a heading and the velocity is locally computed. For increased interactivity, the ALICE chatterbot engine has also been integrated into virtual agents. This allows them to have small conversations with users and to define goals based on the location of agents which a virtual agent is chatting with.

For testing, one PC ran only virtual agents while a second PC ran both virtual agents and a viewer. These computers were placed on different networks, a university network connection and an at home cable modem respectively.

We have tested our approach in a two scenarios:

- **Populating a city block**: This city block scenario includes 2 streets, a fountain and a patio of a building. Virtual agents are added to the city block to improve realism (Fig. 6.27). Virtual agents randomly select points of interest while interacting with each other through repulsive forces. Each agent randomly selects a goal and travels to that location, after which it selects another goal. All agent and obstacle interactions are with repulsive forces. Basic avoidance as well as a little bit of lane formation is apparent. The server managed 18 virtual agents and 3 real agents. The 17 virtual agents were distributed between two PCs.

- **Campus tour guide**: To show a wider variety of behaviors, we added agents to a university campus. This scene included three classes of virtual agents: wandering students, a tour group, and a tour guide. Grouping and following along with the effects of both attractive and repulsive behaviors and following are observable in this scene (See Fig. 6.11(c)). There are a total of 20 agents, 2 real and 18 virtual agents, in the scene. As before the virtual agents were simulated on two client

PCs.

The overall run-time performance in both scenarios was about the same. Virtual agents were distributed over 2 client computers. After tuning the computers, we were able to achieve interactive performance. So, it is likely that this approach will scale with additional client computers.

While the simulation computation was relatively cheap, the biggest limiting factor was the available bandwidth. Each agent, even when hosted on the same client computer, sends and receives a extensive amount of information to the server. In our experiments, each PC could support about 8 to 15 agents without noticeable degradation in performance. It should be noted that our limit of 18 agents is unrelated to the performance. Instead, it was the number of accounts on Second Life we were able to use. In the future, we expect to use many more agents.

6.7 Conclusions

We present a potential field based method for local coordination of multiple robots. For local avoidance and navigation, social potential fields were generalized for convex and non-convex polyhedra and adapted to fit within a physics-based motion planning framework. Overall, the results are promising based on our preliminary implementation. We are able to plan for many robots, several of which are often in tight spaces, in a relatively short amount of time.

There are several additional directions for future work along with the items mentioned in the limitations. It would be useful to integrate this method with a global planner to allow it to function in larger and more complex environments. A similar approach may be possible for holonomic vehicles as well as for articulated robots. For a more realistic simulations, sensing and communication can be considered. Better models of discretization can help to reduce the number of samples per robot while also preserv-

ing accuracy. Since the approach is parallelizable, better implementations are likely to improve performance.

We present a novel approach for real-time navigation of independent agents in complex and dynamic environments. We use adaptive roadmaps and present efficient algorithms to update them. These roadmaps are augmented with link bands to resolve collisions among multiple agents. The algorithm has been applied to complex indoor and outdoor scenes with hundreds or thousands agents and dynamic obstacles. Our preliminary results are encouraging and the algorithm can compute collision-free paths for each agent towards its goal in real time.

There are many avenues for future work. First of all, we would like to incorporate agent clustering in a multi-resolution simulation framework [OFL01] to handle a very large number of agents, e.g. 10-20K independent agents at interactive rates. Secondly, we would like to use better models for local dynamics and behavior modeling that can result in more realistic crowd-like behavior. Instead of its current simple model, we would like to use higher DoF articulated models for each agent to generate more realistic motion. However, this would increase the dimensionality of the configuration space and significantly increase the complexity of the navigation algorithm. Finally, it may be useful to extend these results to generate truly heterogeneous crowd behavior [Bon95], using example based models to guide the simulation [LCL07].

Finally, we have applied our planning and collision avoidance to virtual agent navigation in online virtual worlds. In our implementation, each virtual agent navigates using an established pedestrian model augmented to take sensing latencies into account. Our preliminary results are promising and the algorithm can control and simulate several virtual agents from a single computer.

While the approach works well, there are some limitations. The approach generally requires more bandwidth, which can become a bottleneck. Moreover, server-based solutions probably require modifications on the server if not already supported and thus

may be less practical. As with many "potential field" planners, there is no guarantee of success in reaching a goal. Finally, we hope to scale our work to include much larger crowds in virtual worlds.

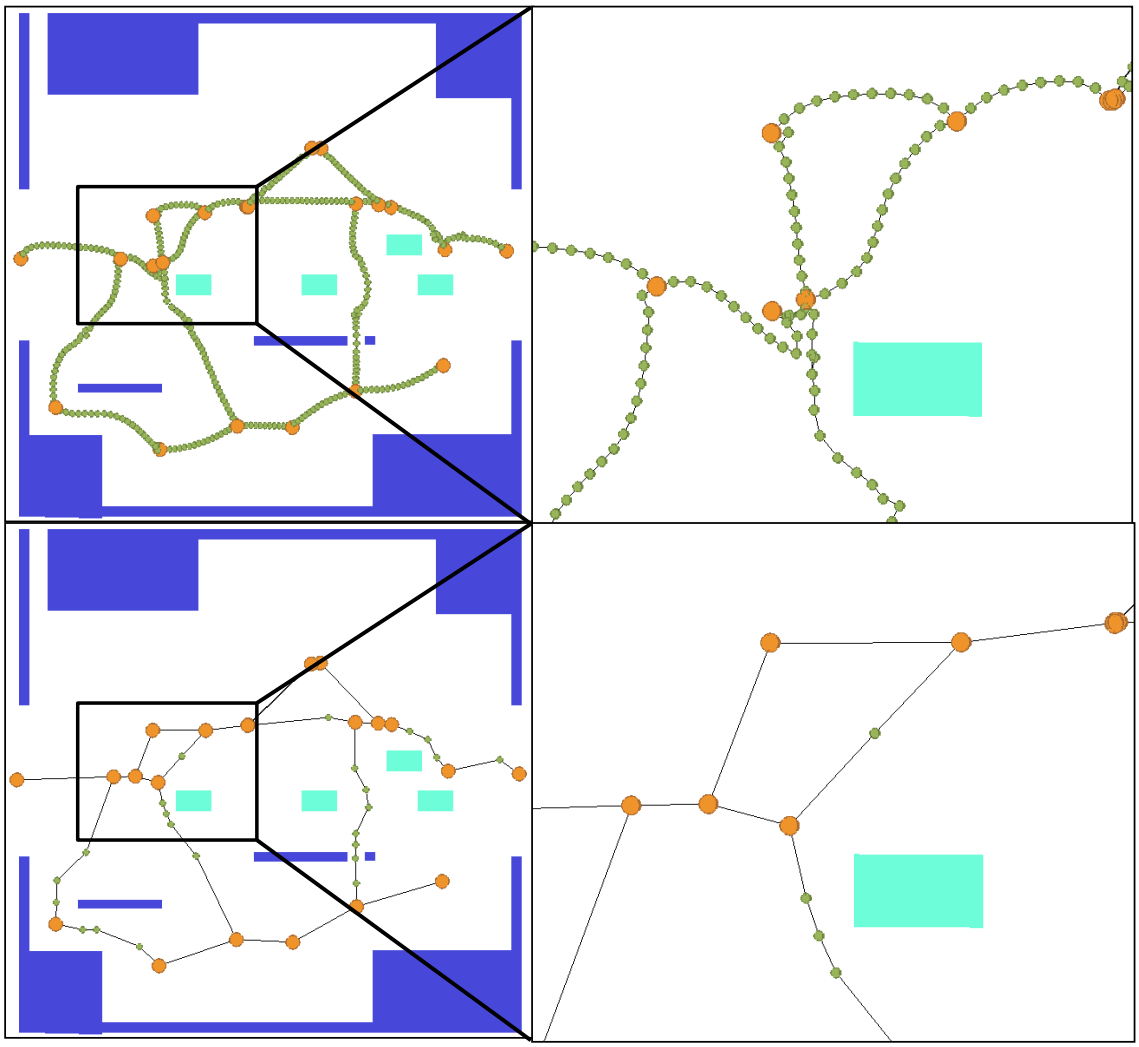

Figure 6.20: APSL vs. Uniform Sampling: The simple city scene after 700 simulation steps. Larger, orange circles are milestones, while the light green circles are the internal samples along a link. (Top row) Uniform sampling effectively covers the links to provide smooth deformations. (Bottom row) APSL provides straighter links and can provide equivalent sampling in the most deformed state of a link. (Right column) A zoomed in view of the boxed region is shown to highlight the differences as well as unnecessary bends and oscillation in the Uniform case.

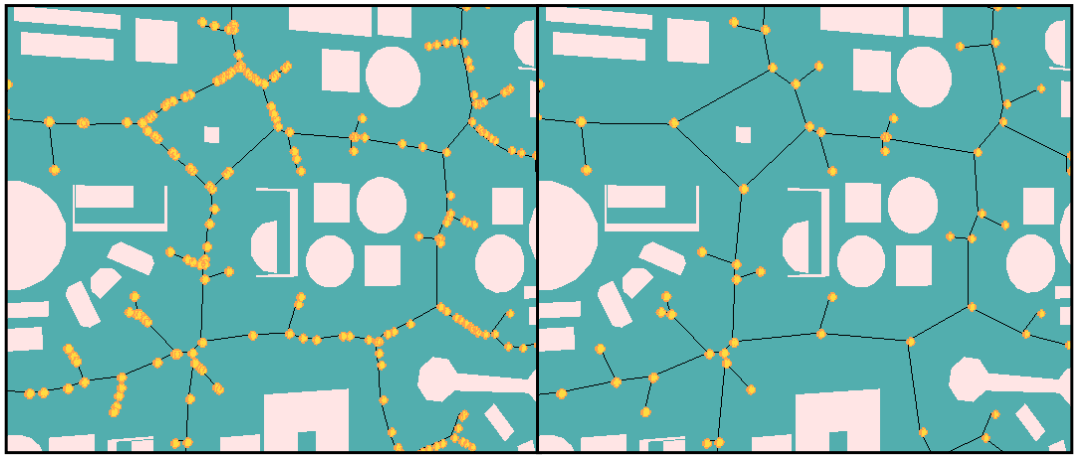

Figure 6.21: APSL Roadmap Cleaning: (Left) A zoomed in portion of the input roadmap for the tradeshow scenario, where roadmap links are the black lines and milestones are the yellow circles. (Right) By applying the APSL algorithm, many redundant milestones have been removed which helps to improve RDR performance.

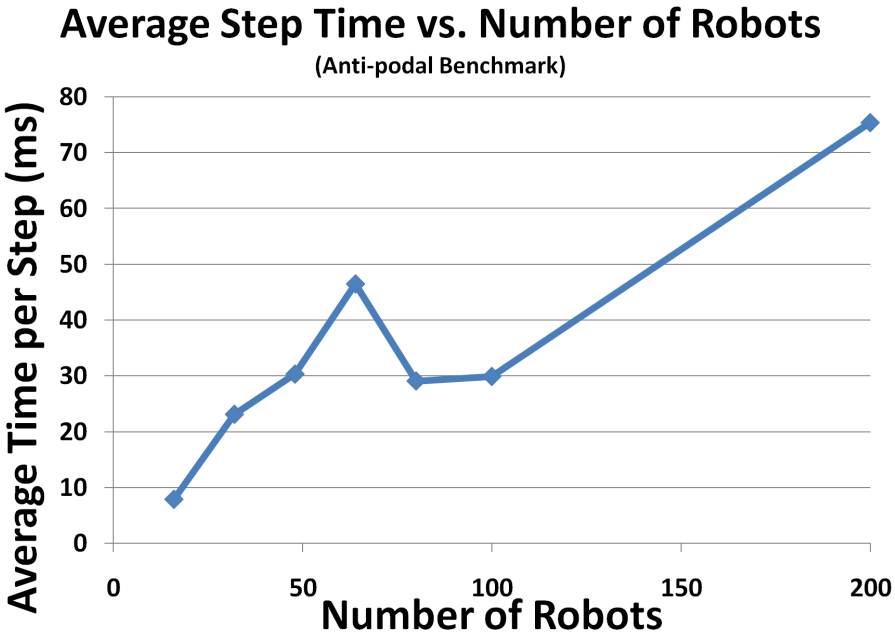

Figure 6.22: Antipodal benchmark timings: To test scalability, the number of robots was increased in the antipodal benchmark.

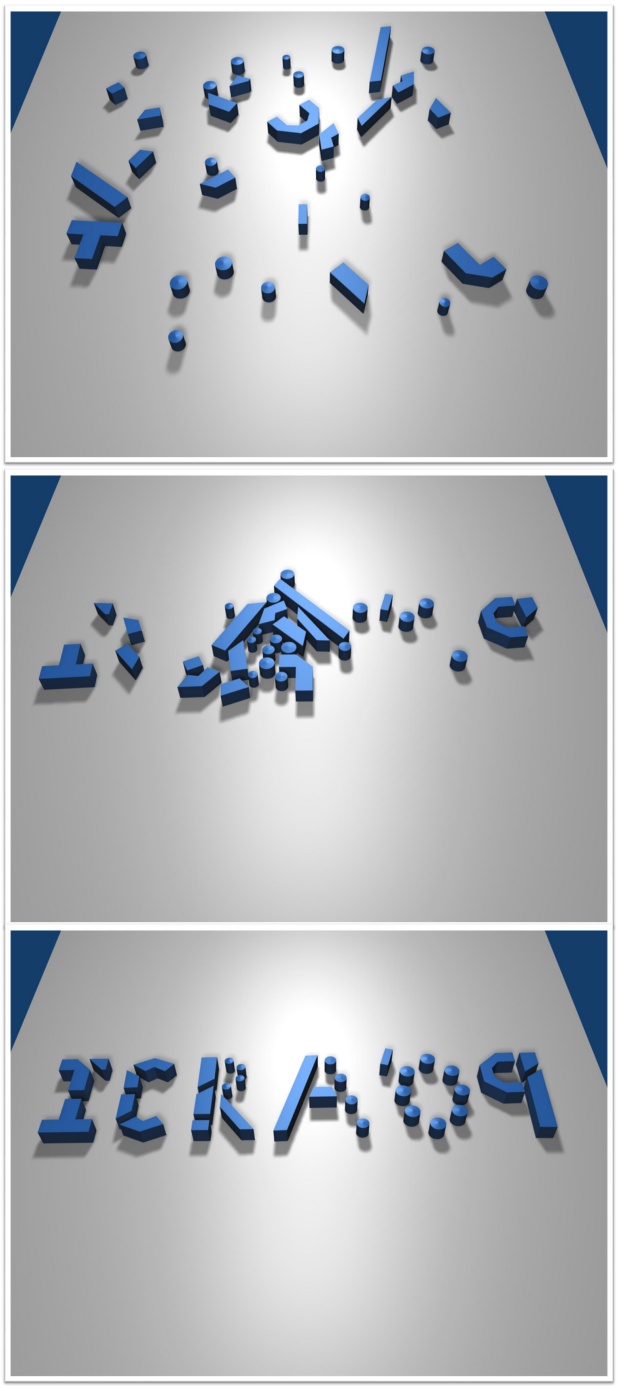

Figure 6.23: Letters: The letters benchmark consists of 33 convex and non-convex robots in an assembly-like situation. (a) The letters start off at random positions and orientations. (b) Repulsive and attractive forces allow the robots to move toward their goals but to also avoid collisions with each other. (c) The robots arrive at their final location.

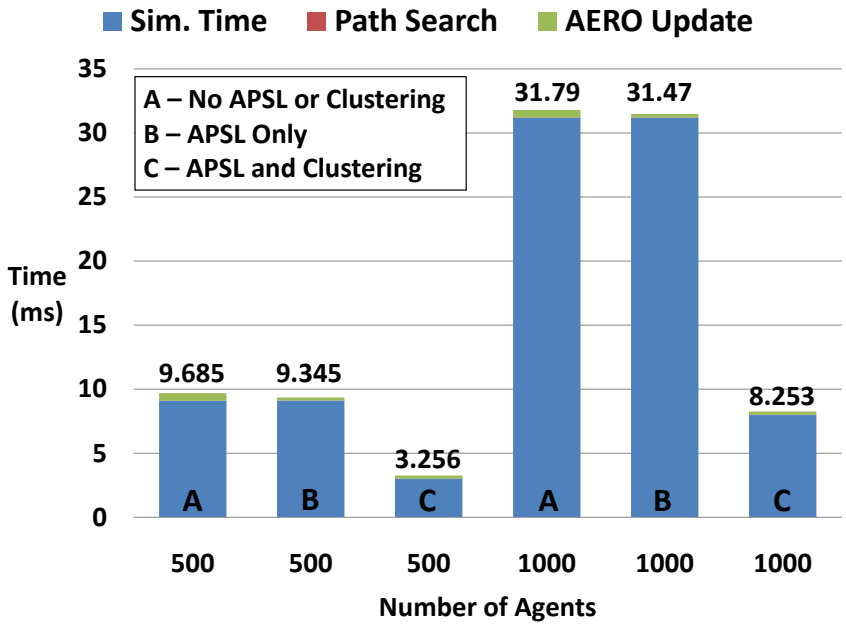

Figure 6.24: Maze - Performance vs. Number of Agents: This chart compares the performance of including APSL and agent clustering into a crowd system for the maze scenario.

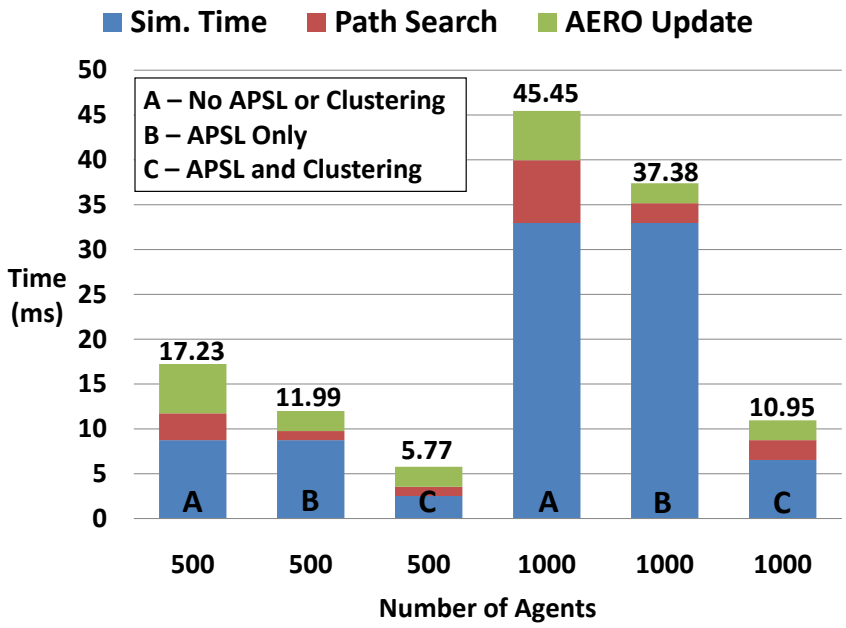

Figure 6.25: Trade Show- Performance vs. Number of Agents: This chart compares the performance of including APSL and agent clustering into a crowd system for the trade show scenario.

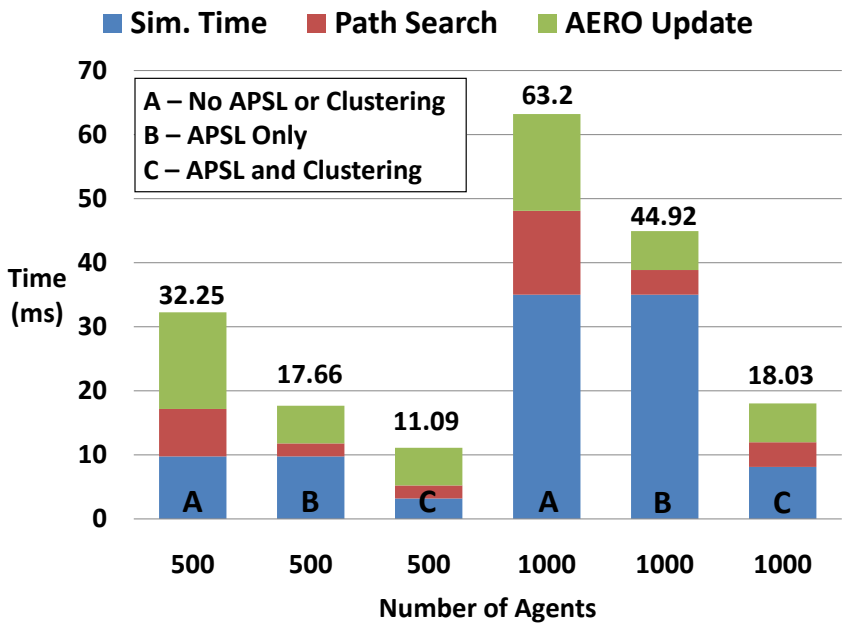

Figure 6.26: City - Performance vs. Number of Agents: This chart compares the performance of including APSL and agent clustering into a crowd system for the Cityscenario.

Figure 6.27: City block: The sequence of images follows the evolution of virtual agents in a small city block. Several virtual citizens (circled in red and wearing shirts in various shades of blue) move around the streets and occasionally stop by the fountain. Virtual agent behaviors include exploration, attraction to the fountain (for a period of time), and avoidance. Agents r_1 and r_2 are identified to demonstrate their avoidance of each other as they cross near the fountain. A human controlled agent (orange) acts as an obstacle and must be avoided. The scene has 18 virtual agents distributed over 2 PCs.

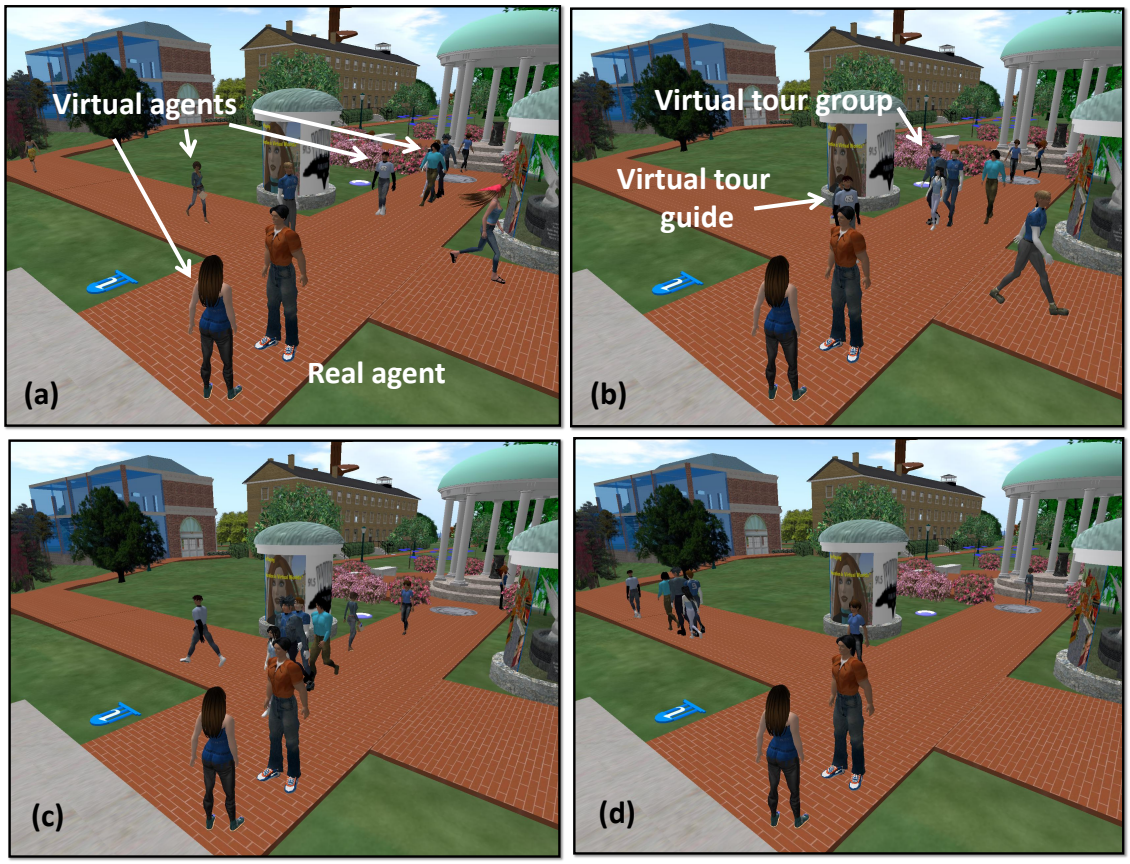

Figure 6.28: Campus tour guide: A sequence of still images following the evolution of virtual agents in a campus scene, at the University of North Carolina's Second Life campus. To better simulate the real world, several virtual agents (wearing shirts with various shades of blue) have been added to the scene. Agent behaviors include leading a group, following, being part of a group, selecting random goals for exploration, and avoidance. This models a virtual tour guide leading a small tour group while also avoiding the human controlled agents. There are 18 virtual and 2 real agents in this scene.

Figure 6.29: Crowd simulation in an urban landscape: A street intersection in a virtual city with 924 buildings, 50 moving cars as dynamic obstacles and $1,000$ pedestrians. We show a sequence of four snapshots of a car driving through the intersection. As the car approaches a lane of pedestrians (top), the lane breaks (middle two images) and the pedestrians re-route using alternate links on the adaptive roadmap. Once the car leaves the intersection (bottom) the pedestrians reform the lane using the adaptive roadmap. We are able to perform navigation of $1,000$ pedestrians in this extremely complex environment at 54fps on a 2.6Ghz Dual Processor PC.

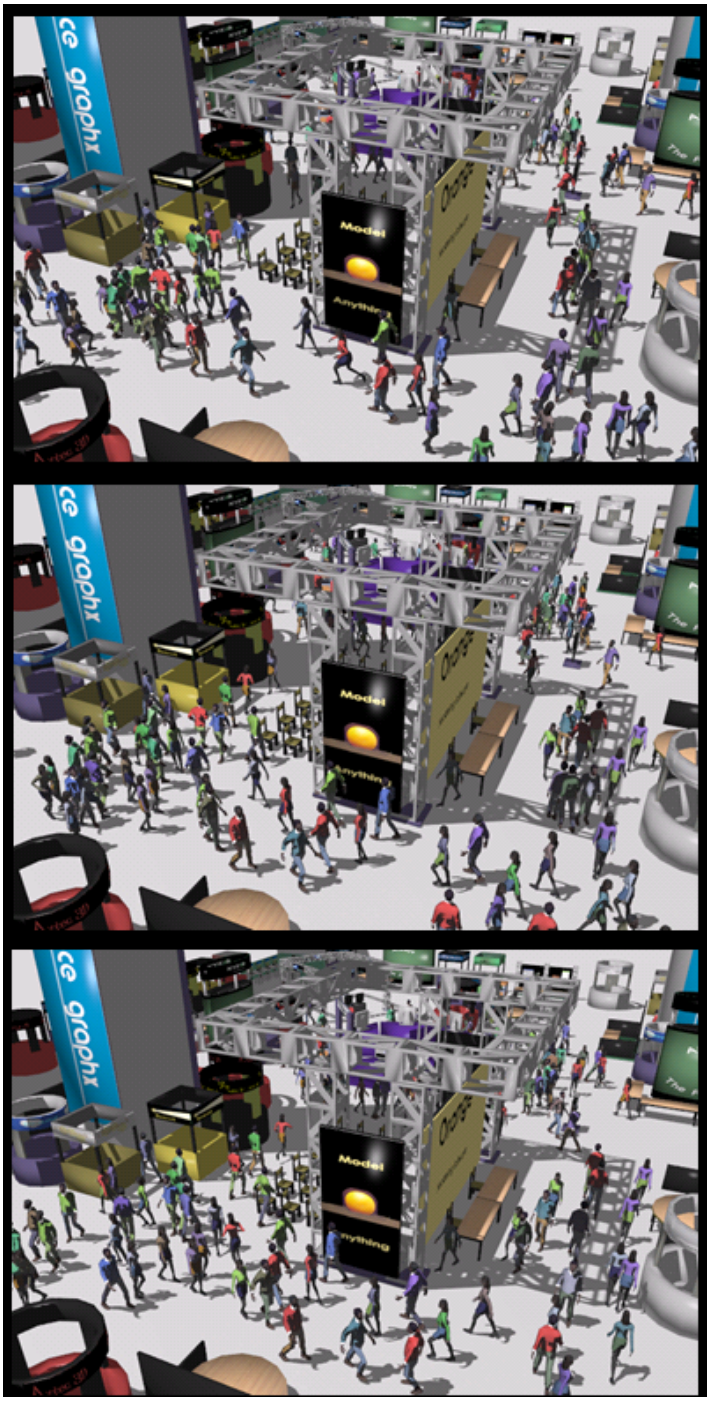

Figure 6.30: Sequence of 3 snapshots from Tradeshow demo: The environment contains 511 booths with 110K polygons. The agents move toward different booths and avoid each other using link bands.

Chapter 7

Conclusion

In this dissertation, we have developed a novel framework for motion planning using physics-based sampling and showed how it can be applied to several realistic situations by expanding the list of constraints either explicitly as part of the robot's motion model, including planning with deformable soft-body robots, highly articulated serial linkages, and finally for a large number of agents in dynamics environments. In practice, the method performs well in that the physics simulation core which generates the samples is close to "real-time" and scales well to a large number of bodies and complex situations. Many optimizations to the physics simulation components, specifically for our motion planning problems, further help with the performance. Thus, the total planning time required is usually proportional to how long the task would take in real life. Also, since the underlying physics model usually solves for second order motion equations (i.e. considering acceleration), the resulting motion tends to be smoother than other planning algorithms which are based on straight-line linkages between milestones.

It is important to note that there are a couple of noteworthy limitations to the approach. While the physical equations are modeled for each scenario, not all kinodynamic constraints are considered in all cases. Thus, for certain bodies and trajectories, not all motions will always be physically-plausible. Variations on our guiding path, such as that proposed by [RLA06a], could included to help alleviate these problems. More fun-

damentally, there is no guarantee that a robot will find a path even if one exists, such as when the body becomes "stuck" or get snagged on some part of the environment (in the case soft-bodies) or when the robot reaches a local minima in the artificial force field. However, in practice this has not been the case in our experiments. Excitation in the local motion model can help overcome some of the limitations of local minima, but is not guaranteed to work in all cases. For instance, a relatively small and light rigid body will not be able to move past several larger, heavier robots. With respect to dynamic obstacles, a fast moving obstacles may still collide with a robot since the robot may not always have enough time to react. A velocity-bias also helps with this situation, by essentially projecting the moving obstacle forward in time to a position which a robot can reason about. This situation can also usually be overcome by placing velocity-limits on the obstacles or otherwise tuning the local avoidance parameters. Regarding parameters, many of the simulation components require tuning of parameters to have a realistic and stable simulation. Some algorithms exist which can auto-time some of these parameters, but to the best of our knowledge no approach does this well in a generic manner.

Along with the variety of limitations, there are several avenues for future work. We would like to obtain better simulation coefficients through physical measurements, particularly in the case of deformable robots and catheters. This could improve the modeling and interaction of the soft-bodies with the surrounding environment. In these cases, it would be then worthwhile to compare the results to real-life procedures to help improve our model along with the decisions made when performing the procedures. We could further improve these simulations by include a deformable environment, much like the internals of human-body. In fact, this situation would work extremely well with RDR, since paths would rarely be invalidated (the global connectivity should not change) and only local adjustments would need to be made to the path itself. At the moment, we often rely on a guiding path generated from the medial axis or a Voronoi diagram of

the workspace. While the property of being far away from obstacles helps to reduce the chance of collisions, it does not always procedure the most efficient final paths (in terms of time needed or total work necessary to execute the path). We would like to see if the effectiveness of the approach changes with different models of guiding paths or roadmaps. In real life certain robots, such as the snake-like robots, also have unique modes of locomotion. It would be interesting to improve our physics model to capture this sort of locomotion and incorporate that in our physics-based sampling search. Much of the current framework assumes perfect knowledge of position and velocity. It would also be interesting to consider sensing and communication between robots, much like the work of [BTK07]. Finally, in most cases, these implementations are still not completely optimized. Many of these algorithms can be trivially parallelized or can be implemented with more efficient data structures, especially in the physics simulation core and its supporting components. These optimizations could help tremendously with performance. In a similar manner, faster, more robust components in this core could also allow for the robots to take larger steps over time, also helping the overall performance.

Finally, we highlight the results of our work by showing a variety of uses of the framework for realistic and challenging situations which would be difficult for many other motion planning algorithms, including a medical procedure called liver chemoembolization, using snake-like robots for industrial pipe inspection and for search-and-rescue operations, for social planning of crowds of agents in an open city environment which includes vehicles, to estimate pedestrian traffic in a large trade-show environment, and finally for populating online virtual worlds. Each of these tasks can include complex robots, a large number of robots, a complex dynamic environment, or real-time constraints in the form of network latency.

Appendix A

Numerical Integration

As posed in this work, the motion equations in physical simulation are in the form of (coupled) ordinary differential equations (ODEs). In general, these are of the form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, t)$$

where f is a known function and $\mathbf{x}$ is the state of the bodies in the simulation.

In order to evolve the state of the simulation, or otherwise advance time, it is necessary to solve these ODEs. This is often referred to as either numerically solving the ODEs and also numerical integration. There are many different approaches for numerically solving ODEs, each with various benefits and trade-offs. We will detail the most popular methods as well as some trade-offs for each one here.

A.1 Euler's method

Perhaps the simplest method for solving ODEs is Euler's Method. This approach approximates the solution by using the first two terms in the Taylor expansion of x,

$$\mathbf{x}(t + h) = \mathbf{x}(t) + hf(\mathbf{x}(t), t)$$

In more general terms, x is approximated by evaluating the tangent of the curve at t and then moving along that slope by some amount h.

For example, consider the state of a particle $\mathbf{p}(t)$ with state $\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix}$. If we differentiate this with respect to time and note that $\dot{v} = \mathbf{a} = \frac{\mathbf{f}}{m}$, then

$$\begin{pmatrix} \dot{x} \\ \dot{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \frac{\mathbf{f}}{m} \end{pmatrix}$$

yields a pair of coupled first order differential equations.

To apply Euler's method, we first select some fixed time step size h. Then,

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}$$

and

$$\mathbf{v}(t+h) = \mathbf{v}(t) + h\mathbf{a}(t) = \mathbf{v}(t) + h\frac{\mathbf{f}}{m}$$

In order to take a step in time using Euler's method, it is first necessary to solve for the total force acting on the particle, $\mathbf{f}$.

The error in this approximation can be expressed as the remaining terms in the Taylor expansion.

$$\frac{1}{2}h^2 f''(t) + O(h^3)$$

Since the number of steps required to solve a solution (over a range of t) is proportional to $\frac{1}{h}$, the accumulated error is proportional to h. Thus, Euler's method is considered a first order approximation to f.

Additional analysis reveals that Euler's method has a relatively small region of stability (with respect to h), and thus is considered *unstable* particularly with stiff differential equations. This forces the use of smaller values of h, which can significantly increase the number of steps needed for a solution. However, due to its simplicity and relatively quick per-step performance (it only requires a single force computation), it is a popular

choice in many physical simulations.

A.1.1 Backward Euler

To overcome the stability problem, many simulations make use of implicit methods. These are known to have much larger stability regions and in some cases are unconditionally stable. The implicit version of Euler's method, or familiarly backward Euler uses the approximation:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + hf(\mathbf{x}(t+h), t+h)$$

To use this method, $\mathbf{x}(t+h)$ must be found or approximated ahead of time. Various methods can be used to approximate it.

Since backward Euler requires this additional computation, it is generally slower than (the explicit) Euler's method. Furthermore, implicit methods generally are a little more complex in implementation and are often less accurate.

A.1.2 Higher order explicit methods

Euler's method is known to be relatively unstable, expensive in that the step sizes are typically very small, and relatively inaccurate. More terms in the Taylor expansion of $\mathbf{x}$ could be used to simultaneously improve accuracy and improve stability. However, this comes at the cost of additional derivative computations. The second-order method, or the midpoint method, has the form

$$\mathbf{x}(t+h) = \mathbf{x}(t) + hf(\mathbf{x}(t) + \frac{h}{2}f(\mathbf{x}(t), t), t + \frac{h}{2})$$

Intuitively, this uses Euler's method to first approximate the value of $\mathbf{x}(t + \frac{h}{2})$. Then, it computes the final value of $\mathbf{x}(t+h)$ based on the tangent at this point. This method

can generalized as in the class of Runga-Kutta methods. If stability and accuracy are a concern and if the computational cost of additional derivative computations is acceptable, then higher order methods may be a good choice for simulations.

A.2 Verlet integration

Verlet integration was proposed specifically to integrate Newton's equations of motion. In general, it offers greater stability than Euler integration while only requiring a single derivative computation. Furthermore, it has been shown to have some properties such as time-reversibility which is beneficial to physical simulations.

Verlet integration reduces the error by calculating the position at the next time based on the positions at the previous and current time step. It is derived from the addition of two Taylor expansions of $\mathbf{x}$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t) + \frac{h^2}{2}\mathbf{a}(t) + \frac{h^3}{6}\dot{\mathbf{a}}(t) + O(h^4)$$

and

$$\mathbf{x}(t-h) = \mathbf{x}(t) - h\mathbf{v}(t) + \frac{h^2}{2}\mathbf{a}(t) - \frac{h^3}{6}\dot{\mathbf{a}}(t) + O(h^4)$$

Note that both the $\mathbf{v}$ and $\dot{\mathbf{a}}$ terms cancel out, leaving a $O(h^4)$ error term.

After adding and re-arranging the terms, Verlet integration is given by:

$$\mathbf{x}(t+h) = -\mathbf{x}(t-h) + 2\mathbf{x}(t) + h^2\mathbf{a}(t)$$

If needed, the instantaneous velocity at time t can be found by using the mean value theorem and the computed positions.

Bibliography

[ABD+98] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. *Proceedings of WAFR*, pages 197–204, 1998. 38

[ABG06] Ron Alterovitz, Michael Branicky, , and Ken Goldberg. Constant-curvature motion planning under uncertainty with applications in image-guided medical needle steering. *Algorithmic Foundation of Robotics VII*, 2006. 67

[AdBvds+99] Boris Aronov, Mark de Berg, A. Frank van der stappen, Petr Svestka, and Jules Vleugels. Motion planning for multiple roobts. *Discrete and Computational Geometry*, 1999. 44

[ALA+01] K. Ashida, S. J. Lee, J. Allbeck, H. Sun, N. Badler, and D. Metaxas. Pedestrians: Creating agent behaviors through statistical analysis of observation data. *Proc. Computer Animation*, 2001. 45

[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. 39, 41

[Bai07] William Sims Bainbridge. The scientific research potential of virtual worlds. *Science*, 2007. 163

[Bar91a] David Baraff. Coping with friction for non-penetrating rigid body simulation. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 31–40, July 1991. 43

[Bar91b] Jerome Barraquand. Automatic motion planning of complex articulated bodies. Technical report, 1991. 41

[Bar96] David Baraff. Linear-Time simulation using lagrange multipliers. In *SIGGRAPH 96 Conference Proceedings*, pages 137–146, 1996. 42

[BB05] Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. *Proceedings of Robotics: Science and Systems*, 2005. 39

[BBGN07] J. L. Baxter, E. K. Burke, J. M. Garibaldi, and M. Norman. Multi-robot search and rescue: A potential field based approach. *Autonomous Robots and Agents Series: Studies in Computational Intelligence*, 2007. 45

[BBT02] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 2002. 45

[BK99] Oliver Brock and Oussama Khatib. Elastic strips: A framework for integrated planning and execution. *Proceedings of the 1999 International Symposium on Experimental Robotics*, 1999. 48

[BK00] Oliver Brock and Oussama Khatib. Real-time replanning in high-dimensional configuration space using sets of homotopic paths. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000. 48

[BK02] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *Int. Journal of Robotics Research*, 18(6):1031–1052, 2002. 189

[BL91] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robotic Research*, 1991. 10

[BLA02a] O. B. Bayazit, H. Lien, and N. Amato. Probabilistic roadmap motion planning for deformable objects. *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2002. 39, 91

[BLA02b] O. B. Bayazit, J.-M Lien, and N. M. Amato. Better group behaviors in complex environments with global roadmaps. *Int. Conf. on the Sim. and Syn. of Living Sys. (Alife)*, pages 362–370, 2002. 43

[BLA02c] O. B. Bayazit, J.M. Lien, and N. M. Amato. Probabilistic roadmap motion planning for deformable objects. *Proceedings of International Conference on Robotics and Automation*, pages 2126–2133, 2002. 26

[BLA02d] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy Amato. Roadmap-based flocking for complex environments. *Proceedings of Pacific Conference on Computer Graphics and Applications (PG)*, 2002. 43

[BLP85] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. Syst*, SMC-15:224–233, 1985. 7

[Bon95] Gustave Le Bon. *The Crowd: A Study of the Popular Mind*. 1895. Reprint available from Dover Publications. 198

[BSA00] O. B. Bayazit, G. Song, and N. Amato. Enhancing randomized motion planners: Exploring with haptic hints. *Proceedings of ICRA*, pages 529–536, 2000. 73

[BTK07] K. Bekris, K. Tsianos, and L. Kavraki. A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007. 45, 189, 190, 210

[BW02] G. Baciu and S.K. Wong. Image-based techniques in a hybrid collision detector. *IEEE Trans. on Visualization and Computer Graphics*, 2002. 40

[Can88a] J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988. 7

[Can88b] John Canny. The complexity of robot motion planning. 1988. 14, 37

[CB90] G. S. Chirikjian and J. W. Burdick. An obstacle avoidance algorithm for hyper-redundant manipulators. *IEEE Transactions of Robotics and Automations*, 1990. 97

[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. 97

[CB94b] H. Choset and J. Burdick. Extensibility in local sensor based planning for hyper redundant manipulators (robot snakes). *AIAA/NASA CIRFFSS*, 1994. 97

[CB94c] H. Choset and J. Burdick. Extensibility in local sensor based planning for hyper redundant manipulators (robot snakes). *AIAA/NASA CIRFFSS*, 1994. 97

[CB96] H. Choset and J. Burdick. Sensor based planning: The hierarchical generalized voronoi graph. *Workshop on Algorithmic Foundations of Robotics*, 1996. 75

[CBS+05] O. C. Cordeiro, A. Braun, C. B. Silveria, S. R. Musse, and G. G. Cavalheiro. Concurrency on social forces simulation model. *First International Workshop on Crowd Simulation*, 2005. 46

[CLH+05] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005. 37

[CP03] M. Cline and D. Pai. Post-stabilization for rigid body simulation with contact and constraints. *IEEE Int. Conf. on Robotics and Automation*, pages 3744–3751, 2003. 43

[CRL03] C. Clark, S. Rock, and J. Latombe. Motion planning for multiple robot systems using dynamic networks. *Proc. of IEEE Conf. on Robot. & Autom.*, 2003. 43

[CSL02] J. Cortes, T. Simeon, and J. P. Laumond. A random loop generator for planning the motions of closed kinematic chains using prm methods. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002. 41

[Don84] B. R. Donald. Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab., 1984. AI-TR-791. 7

[DXCR93] B. R. Donald, P. Xavier, J. Canny, and J. H. Reif. Kinodynamic motion planning. *Journal of ACM*, 40(5):1048–1066, November 1993. 38

[ELP86] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Proc. of IEEE Conf. on Robot. & Autom.*, pages 1419–1424, 1986. 45

[Eri04] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2004. 40

[Fea87] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer, Boston, MA, 1987. 42, 101, 105

[Fea99a] 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. 42, 99, 107, 109

[Fea99b] Roy Featherstone. A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 2: Trees, loops, and accuracy. *International Journal of Robotics Research 18(9):876-892*, 1999. 42, 107, 109

[FGLM01] 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. 38, 73, 75

[FKS06] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006. 47

[FO00] R. Featherstone and D. E. Orin. Robot dynamics: Equations and algorithms. *IEEE Int. Conf. Robotics and Automation, pp. 826-834*, 2000. 42, 102

[Fra93] T. Fraichard. Dynamic trajectory planning with dynamic constraints: A "state-timespace" approach. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, July 1993. 47

[FTT99] J. Funge, X. TU, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proc. of ACM SIGGRAPH*, pages 29–38, 1999. 43

[GC02] S. S. Ge and Y. J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 2002. 45, 190

[GHK99] L. Guibas, C. Holleman, and L. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of IROS*, pages 254 – 259, 1999. 38, 39

[GHSC96] J. Gates, G. Hartnell, K. Stuart, and M. Clouse. Chemoembolization of hepatic neoplasms: Safety, complications, and when to worry. *Radiographics*, 19:399–414, 1996. 66, 93

[GKJ+05] N. Govindaraju, D. Knott, N. Jain, I. Kabal, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha. Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 24(3):991–999, 2005. 40

[GKX07] Russell Gayle, Kristopher Klingler, and Patrick Xavier. Lazy reconfiguration forest (lrf) - an approach for motion planning with multiple tasks in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 39

[GL02] M. Garber and M. Lin. Constraint-based motion planning using voronoi diagrams. *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002. 57, 59, 157, 190

[GLM06] R. Gayle, M. Lin, and D. Manocha. Adaptive dynamics with efficient contact handling for articulated robots. *Proc. of Robotics: Systems and Science*, 2006. 101

[GO07] Roland Garaerts and Mark H. Overmars. The corridor map method: Real-time high-quality path planning. In *ICRA*, pages 1023–1028, Roma, Italy, 2007. 44, 139, 195

[GRC05] A. Greenfield, A. Rizzi, and H. Choset. Dynamical ambiguities in frictional rigid-body systems with applications to climbing via bracing. *Proc. of ICRA*, 2005. 97

[GRLM03] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 25–32, 2003. 40, 74, 86, 88

[Gro03] Parasol Research Group. Motion planning benchmarks. http://parasol.tamu.edu/groups/amatogroup/benchmarks/mp, 2003. 90, 91, 92

[GRS+07] Russell Gayle, Stephane Redon, Avneesh Sud, Ming C. Lin, and Dinesh Manocha. Efficient motion planning of highly articulated chains using physics-based sampling. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 157

[GSLM05a] R. Gayle, P. Segars, M. Lin, and D. Manocha. Path planning for deformable robots in complex environments. Technical report, University of North Carolina-Chapel Hill, 2005. Proc. of Robotics: Science and Systems. 41

[GSLM05b] R. Gayle, P. Segars, M. Lin, and D. Manocha. Path planning for deformable robots in complex environments. *Proc. of Robotics: Science and Systems*, 2005. 97

[HA00] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. *Proceedings of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000. 41

[Hal02] D. Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002. 7

[HBJW05] D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science*, 39:1–24, 2005. 47

[HBW03] D. Helbing, L. Buzna, and T. Werner. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*, 2003. 46, 143, 157, 166

[HCK+99] Kenneth E. Hoff, III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Computer Graphics Annual Conference Series (SIGGRAPH '99)*, pages 277–286, 1999. 74

[HH03] S. Hirsch and D. Halperin. Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane. *Sringer Tracts in Advanced Robotics*, pages 239–256, 2003. 7

[HHC98] W. Henning, F. Hickman, and H. Choset. Motion planning for serpentine robots. *Proc. of ASCE Space and Robotics*, 1998. 97

[HKL+98] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, pages 25–32, 1998. 38

[HKLR02] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 2002. 15, 38, 48

[HKW98] C. Holleman, L. Kavraki, and J. Warren. Planning paths for a flexible surface patch. *IEEE Int. Conf. Robot. Autom. (ICRA)*, 1998. 39

[HLB+00] S. P. Hoogendoorn, S. Luding, P.H.L. Bovy, M. Schrecklenberg, and D.E. Wolf. *Traffic and Granular Flow*. Springer, 2000. 46

[Hol80] J.M. Hollerbach. A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No. 11*, 1980. 42

[HTG03] B. Heidelberger, M. Teschner, and M. Gross. Real-time volumetric intersections of deforming objects. *Proc. of Vision, Modeling and Visualization*, pages 461–468, 2003. 40

[Jak01] T. Jakobsen. Advanced character physics. In *Game Developer's Conference*, 2001. 191

[JP04] D. L. James and D. K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH*, pages 393–398, 2004. 40

[JS04] L. Jaillet and T. Simeon. A PRM-based motion planning for dynamically changing environments. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004. 47

[Kha85] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1985. 45, 48, 143, 147, 190

[Kha86] O. Khatib. Real-time obstable avoidance for manipulators and mobile robots. *IJRR*, 5(1):90–98, 1986. 10

[KL00] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings of the IEEE Intnational Conference on Robotics and Automation (ICRA), San Francisco, CA*, April 2000. 37, 39, 98

[KL02] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002. 47

[KM04] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, April 2004. 47, 189

[KO04] A. Kamphuis and M. Overmars. Finding paths for coherent groups using clearance. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 19–28, 2004. 43

[Kok04] V. Kokkevis. Practical physics for articulated characters. *Proc. of GDC*, 2004. 43, 55, 101, 116, 118

[KP03] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, pages 73–80, 2003. 40

[KSLO96] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580, 1996. 8, 37, 73, 98, 188

[KvdP06] Maciej Kalisiak and Michiel van de Panne. RRT-blossom: RRT with a local flood-fill behavior. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006.

[KZ86] K. Kant and S. Zucker. Towards efficient trajectory planning: The path-velocity decomposition. *Int. Journal of Robotics Research*, 5(3):72–89, 1986. 15, 189

[LAM01] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. *Eurographics*, pages 325–333, 2001. 40

[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. 37, 82, 83

[LaV06a] S. M. LaValle. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006. 43, 45

[LaV06b] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. 37, 156

[LBL04] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics*, December 2004. 48

[LC03a] T.-T. Li and H.-C. Chou. Motion planning for a crowd of robots. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 3:4215–4221, 2003. 64

[LC03b] T-Y. Li and H-C. Chou. Motion planning for a crowd of robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003. 44

[LCL07] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum (Proceedings of Eurographics)*, 26(3), 2007. 198

[LD04] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real-time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):509–518, 2004. 43, 194

[LG07] Yi Li and Kamal Gupta. Motion planning of multiple agents in virtual environments on parallel architectures. In *ICRA*, pages 1009–1014, Roma, Italy, 2007. 44, 194

[LGLM99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999. 87

[LGP05] Yi Li, Kamal Gupta, and Shahram Payandeh. Motion planning of multiple agents in virtual environments using coordination graphs. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005. 45, 189

[LH00] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. *Proceedings of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000. 47

[LK99] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *In Proc. IEEE Int'l Conf. on Robotics and Automation*, 1999. 38, 61, 104, 133

[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. 8, 73, 188

[LK01] F. Lamiraux and L. Kavraki. Path planning for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001. 39

[LK04] A. Ladd and L. Kavraki. Using motion planning for knot untangling. *International Journal of Robotics Research*, 23(7-8):797–808, 2004. 41, 97

[LK05] A. M. Ladd and L. E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. *Proc. of the Sixth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2005. 38

[LKF05] Taras I Lokoba, D. J. Kaup, and Neal M. Finkelstein. Modifications of the helbing-molnr-farkas-vicsek social force model for pedestrian evolution. *Simulation*, 2005. 177

[LL05] Stephen R. Lindemann and Steven M. LaValle. Incrementally reducing dispersion by increasing voronoi bias in RRTs. *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2005. 39

[LM03] M. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003. 40, 42, 83

[LMM03] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. In *Theory and Practice of Computer Graphics (TPCG'03)*, pages 122–129, 2003. 46

[LP83] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, 1983. 5

[LPW79] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979. 7

[LRMA05] Jyh-Ming Lien, Samuel Rodriguez, Jean-Phillipe Malric, and Nancy Amato. Shepherding behaviors with multiple shepherds. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005. 43

[LS02] T-Y. Li and Y-C. Shie. An incremental approach to motion planning with roadmap management. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002. 47

[LYK99] S. LaValle, J. Yakey, and L. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. *Proceedings of the IEEE Intnational Conference on Robotics and Automation (ICRA)*, 1999. 41

[Mir96] B. V. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, 1996. 43, 101, 106, 112

[MK04] M. Moll and L. Kavraki. Path planning for minimal energy curves of constant length. *Proceedings of International Conference on Robotics and Automation*, 2004. 41

[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. 41

[MM03] A. Mueller and P. Maisser. A lie-group formulation of kinematics and dynamics of constrained mbs and its application to analytical mechanics. *Multibody System Dynamics, vol. 9, no. 4, pp. 311-352(42)*, 2003. 42

[MO03] M. Matyka and M. Ollila. A pressure model for soft body simulation. *Proc. of SIGRAD*, pages 325–333, 2003.

[MT97] S. R. Musse and D. Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation*, pages 39–51, 1997. 46

[NK97] H. Nakagaki and K. Kitagaki. Study of deformation tasks of a flexible wire. *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1997. 39, 41

[NM96] W. Ngugen and J. Mills. Multi-robot control for flexible fixtureless assembly of flexible sheet metal auto parts. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2340–2345, 1996. 39

[OFL01] D. O'Brien, S. Fisher, and M. Lin. Simulation level of detail for automatic simplification of particle system dynamics. *Proc. of Computer Animation*, pages 210–219, 2001. 198

[OLP89] P.A. O'Donnell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. *Proc. IEEE International Conference on Robotics and Automation*, pages 484–489, 1989. 15

[PA05] Jufeng Peng and Srinivas Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal on Robotics Research*, 2005. 189

[PBC+05] E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(6):597–608, 2005. 39

[PF05] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005. 48

[PHLM00] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000. 38

[PK06] E. Plaku and L. E. Kavraki. Quantitative analysis of nearest neighbor search in high-dimensional sampling-based motion planning. *Proceedings of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2006. 98

[PLT05] J. Pettre, J.-P Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *First International Workshop on Crowd Simulation*, 2005. 43, 44, 194

[POSB05] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*, 2005. 46

[QK93] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. *Proc. of IEEE Conf. on Robotics and Automation*, 1993. 45, 48, 184, 189

[Ree03] Douglas A Reece. Movement behavior for soldier agents on a virtual battlefield. *Presence*, 2003. 46

[Rei79] John Reif. Complexity of the mover's problem and generalization. *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979. 14

[Rey87] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Comput. Graph.*, 21(4):25–34, 1987. Proc. SIGGRAPH '87. 46, 195

[Rey06] Craig Reynolds. Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 113–121, New York, NY, USA, 2006. ACM Press. 45

[RGL05] S. Redon, N. Galoppo, and M. Lin. Adaptive dynamics of articulated bodies. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 24(3), 2005. 29, 42, 101, 109, 110, 111

[RJKD91] G. Rodriguez, A. Jain, and K. Kreutz-Delgado. Spatial operator algebra for manipulator modelling and control. *Int. J. Robotics Research, vol. 10, no. 4, pp. 371-381*, 1991. 42

[RK92] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Trans. on Robotics and Automation*, 8(5):501–518, 1992. 10

[RK97] D.C. Ruspini and O. Khatib. Collision/contact models for the dynamic simulation of complex environments. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1997. 42

[RKC02] S. Redon, K. Kheddar, and S. Coquillart. Gauss' least constraints principle and rigid body simulation. *Proceedings of International Conference on Robotics and Automation*, 2002. 43

[RKLM04] S. Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004. 42

[RLA06a] S. Rodriguez, J. Lien, and N. Amato. Planning motion in completely deformable environments. *Proc. of IEEE Int. Conf. Robot. Autom. (ICRA)*, 2006. 208

[RLA06b] S. Rodriguez, J-M Lien, and N. M. Amato. Planning motion in completely deformable environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006. 38, 39

[RS85] John Reif and Micha Sharir. Motion planning in the presence of moving obstacles. *IEEE Symposium on Foundations of Compuer Science*, October 1985. 47

[RW99] John H. Reif and Hongyan Wang. Social potential fields: A distribued behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 1999. 45, 143, 157, 190

[SAC+07] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha. Real-time path planning for virtual agents in dynamic environments. *Proc. of IEEE VR*, pages 91–98, 2007. 44, 194

[Seg01] William P. Segars. *Development of a new dynamic NURBS-based cardiac-torso (NCAT) Phantom.* PhD thesis, University of North Carolina at Chapel Hill, 2001. 95

[SGA+07] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. *Proc. of ACM VRST*, 2007. 149, 178

[SGC04] M. Sung, M. Gleicher, and S. Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3 (Sept)):519–528, 2004. 46

[SGGM06] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006. 191

[SI06] M. Saha and P .Isto. Multi-robot motion planning by incremental coordination. In *Proc. of IROS*, pages 5960–5963, 2006. 16, 45, 135, 189

[SKG05] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. *Proc. of SCA 2005*, pages 291–300, 2005. 43

[SL02] G. Sanchez and J. Latombe. Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In *ICRA*, pages 2112–2119, 2002. 135

[SLL99] T. Simeon, S. Leroy, and J. Laumond. Path coordination for multiple mobile robots: a geometric algorithm. *Proc. of IJCAI*, 1999. 189

[SLN00] T. Simeon, J. P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000. 38

[SNH01] Y. Sugiyama, A. Nakayama, and K. Hasebe. 2-dimensional optimal velocity models for granular flows. In *Pedestrian and Evacuation Dynamics*, pages 155–160, 2001. 46

[SS83] J. T. Schwartz and M. Sharir. On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351, 1983. 7

[SS01] M. Schreckkenberg and S. D. Sharma. *Pedestrian and Evacuation Dynamics*. Springer, 2001. 45, 46

[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. 39, 41

[SSL02] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, Dec. 2002. 150

[ST05] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2005. ACM Press. 45, 195

[Ste95] A. Stentz. The focussed D* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995. 47, 189

[Sur92] Mark C. Surles. An algorithm with linear complexity for interactive, physically-based modeling of large proteins. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 221–230, July 1992. 97

[Tal94] P. Le Tallec. Numerical methods for solids. *Handbook of Numerical Analysis*, 1994. 79

[TCP06] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. *Proc. of ACM SIGGRAPH*, pages 1160 – 1168, 2006. 47, 175, 194

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 19(1):61–81, 2005. 40

[TOCD06] D. Thalmann, C. O'Sullivan, P. Ciechomski, and S. Dobbyn. *Populating Virtual Environments with Crowds.* Eurographics 2006 Tutorial Notes, 2006. 45

[TPK03] M. Teodoro, G. N. Jr. Phillips, and L. E. Kavraki. Understanding protein flexibility through dimensionality reduction. *The Journal of Computational Biology*, 10(3-4):617–634, 2003. 41

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pages 43–50, 1994. 46

[TTP01] J. C. Trinkle, J. A. Tzitzouris, and J. S. Pang. Dynamic multi-rigid-body systems with concurrent distributed contacts: Theory and examples. *Philosophical Transactions of the Royal Society*, Series A: Mathematical,Physical, and Engineering Sciences(August 15), 2001. 43

[US03] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2003. 39

[vdB97] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997. 40

[vdBLM08] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008. 45

[vdBO05] J. P. van den Berg and Mark H. Overmars. Prioritized motion planning for multiple robots. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005. 45

[vdBPS+08] Jur van den Berg, Sachin Patil, Jason Sewall, Dinesh Manocha, and Ming C Lin. Interactive navigation of individual agents in crowded environments. *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, 2008. 45, 46

[VM05] Gokul Varadhan and Dinesh Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. 7

[vWV98] T. van Walsum and M. Viergever. Deformable b-splines for catheter simulation. Technical report, Image Science Institute, Utrecht University, 1998. 66, 93

[War90] C. W. Warren. Multiple path coordination using artificial potential fields. *Proc. of IEEE Conf. on Robotics and Automation*, pages 500–505, 1990. 45, 190

[WAS99a] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. *IEEE Conference on Robotics and Automation*, pages 1024–1031, 1999. 75

[WAS99b] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. *Proc. of the 15th Annual ACM Symposium on Computational Geometry (SoCG'99)*, pages 173–180, 1999. 38

[WB97] A. Witkin and D. Baraff. *Physically Based Modeling: Principles and Practice*. ACM Press, 1997. Course Notes of ACM SIGGRAPH. 51, 52, 55, 57

[Wil99] Steven Wilmarth. *A Probabilistic Method for Rigid Body Motion Planning Using Sampling from the Medial Axis of the Free Space*. PhD thesis, Texas A&M University, Dec 1999. 38

[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. 97

[WTF05] R. Weinstein, J. Teran, and R. Fedkiw. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Trans. on Visualization and Computer Graphics*, 2005. 43

[Xav97] P. Xavier. Fast swept-volume distance for robust collision detection. In *Proc. 1997 IEEE Int'l Conf. on Robotics and Automation*, pages 1162–1169, Albuquerque, NM, April 1997. 42

[Xav02] P. Xavier. Implicit convex hull distance of finite-screw-swept volumes. In *Proc. 2002 IEEE Int'l Conf. on Robotics and Automation*, pages 847–854, Washington, DC, May 2002. 42

[YB06] Yuandong Yang and Oliver Brock. Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation. *Proceedings of Robotics: Science and Systems*, August 2006. 45, 48, 189

[YJSL05] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005. 39

[YN02] K. Yamane and Y. Nakamura. Efficient parallel dynamics computation of human figures. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages pp. 530–537, 2002. 42

[ZKB07] M. Zucker, J. Kuffner, and M. Branicky. Multipartite rrts for rapid replanning in dynamic environments. *Proc. IEEE Int. Conf. on Robotics and Automation*, 2007. 39, 44, 189

[ZKM07] Liangjun Zhang, Young J Kim, and Dinesh Manocha. A hybrid approach for complete motion planning. *In Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, 2007. 7