REAL-TIME MODIFICATION OF COLLISION-FREE PATHS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Sean Quinlan
December 1994

© Copyright 1995
by
Sean Quinlan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Oussama Khatib
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bernard Roth
(Mechanical Engineering)

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

The modification of collision-free paths is proposed as the basis for a new framework to close the gap between global path planning and real-time sensor-based robot control. A physically-based model of a flexible string-like object, called an *elastic band*, is used to determine the modification of a path. The initial shape of the elastic is the free path generated by a planner. Subjected to artificial forces, the elastic band deforms in real time to a short and smooth path that maintains clearance from the obstacles. The elastic continues to deform as changes in the environment are detected by sensors, enabling the robot to accommodate uncertainties and react to unexpected and moving obstacles. While providing a tight connection between the robot and its environment, the elastic band preserves the global nature of the planned path.

The greater part of this thesis deals with the design and implementation of elastic bands, with emphasis on achieving real-time performance even for robots with many degrees of freedom. To achieve these goals, we propose the concept of *bubbles of free-space*—a region of free-space around a given configuration of the robot generated from distance information. We also develop a novel algorithm for efficiently computing the distance between non-convex objects and a real-time algorithm for calculating a discrete approximation to the time-optimal parameterization of a path. These various developments are combined in a system that demonstrates the elastic band framework for a Puma 560 manipulator.

Acknowledgments

First, I would like to thank Professor Oussama Khatib for his invaluable help and encouragement during my five years at Stanford. As my advisor, Oussama provided both financial support and unwavering enthusiasm for my research.

I gratefully acknowledge the help of Professors Jean-Claude Latombe and Bernie Roth, from whom I had much wise advise. I also thank them for agreeing to be on my reading committee and their review of drafts of this thesis.

Special thanks go to my fellow students at Stanford: Alan Bowling, KC Chang, Sanford Dickert, Bob Holmberg, Yotto Koga, Diego Ruspini, David Williams, Mark Yim, and many others. These people made Stanford an interesting and exciting place to work.

I am grateful for the financial support provided by Fellowships from the National Science Foundation (NSF) and Achievement Rewards for College Students (ARCS). This work was also supported by funding provided by the Boeing Corporation, Hitachi Construction, and the National Science Foundation (contract CDA-9320419).

Finally, I thank my family for their support and encouragement. Special thanks to my father, Ross Quinlan, who carefully read every line of this thesis, and pointed out numerous errors.

Contents

Abstract **iv**

Acknowledgments **v**

1 Introduction **1**
1.1 Background . . . 1
1.2 Executing Collision-Free Motion Tasks . . . 2
1.3 A New Architecture . . . 3
1.4 Elastic Bands . . . 5
1.5 Other Applications for Elastic Bands . . . 5
1.6 Overview of the Thesis . . . 7

2 Collision-Free Motion of Robots **9**
2.1 Introduction . . . 9
2.2 Basic Concepts . . . 9
2.3 Reactive Control . . . 13
2.4 Motion Planning . . . 16
2.5 Combining Planning and Reactive Control . . . 19

3 Elastic Bands **21**
3.1 Introduction . . . 21
3.2 Relation to Previous Work . . . 22
3.3 A Model for an Elastic Band . . . 22
3.4 Using Potential Functions to Specify Forces . . . 25
3.5 An Elastic Band Made from Idealized Elastic Material . . . 28
3.6 A Problem with the Elastic Band Model . . . 29
3.7 A New Model . . . 30
3.8 A Problem with the New Internal Force . . . 31
3.9 Adding a Constraint Force . . . 32
3.10 Changing the length of the elastic . . . 33
3.11 A Summary of the Model . . . 34

3.12 Modifying a Path using Optimization Theory 35

4 Bubbles of Free Space **38**
4.1 Introduction 38
4.2 Bubbles and Cell Decompositions 40
4.3 Applications of Bubbles 41
4.4 Computing Bubbles 44
4.5 A Non-Rotating Free-Flying Robot in the Plane 46
4.6 Planar Robots with Rotation 48
4.7 Open Chain Manipulators 50

5 Implementation of Elastic Bands **56**
5.1 Representing an Elastic Band 56
5.2 The Forces on a Bubble 58
5.3 Deforming the Elastic 63
5.4 Minimization Procedure 71
5.5 Moving Multiple Particles Simultaneously 73
5.6 Concluding Remarks 74

6 Distance Computation **75**
6.1 Introduction 75
6.2 The Hierarchical Bounding Representation 77
6.3 Execution Time 80
6.4 Computing the Distance between Two Objects 80
6.5 Empirical Trials 82
6.6 Conclusion 85

7 System Integration **87**
7.1 Constructing a Smooth Path 88
7.2 Time Parameterization 93
7.3 Combining Path Modification and Trajectory Generation 100
7.4 A Robotic System 103

8 Conclusion **112**

A The Functional Gradient Operator $\overline{\nabla}$ **114**

B The Curvature Vector **119**

References **122**

List of Figures

1.1 A conventional architecture for a robot system 2
1.2 The proposed architecture for a robot system 4
1.3 An example of an elastic band 6

2.1 An example configuration space 12
2.2 A minimum in a potential 15

4.1 An example bubble 39
4.2 A Bézier curve 40
4.3 A path covered with bubbles. 43
4.4 A bubble for a non-rotating robot 46
4.5 An alternative bubble for a non-rotating robot 47
4.6 A non-convex bubble for a rotating robot 49
4.7 A convex bubble for a rotating robot 49
4.8 A two-degree-of-freedom manipulator 51
4.9 Two example bubbles for a manipulator 52
4.10 A bubble for a two-degree-of-freedom manipulator with joint limits 54
4.11 A simpler bubble for a two-degree-of-freedom manipulator with joint limits 54

5.1 Constructing a path through two bubbles 57
5.2 A path though a series of bubbles 58
5.3 An elastic band for a 3 DOF robot 59
5.4 An elastic band for a 6 DOF manipulator 59
5.5 A situation in which v_{ext} is not differentiable 62
5.6 Deformation of an elastic band for a 2-DOF mobile robot 65
5.7 Deformation of an elastic band for a 3-DOF mobile robot 66
5.8 Deformation of an elastic band for a 6-DOF manipulator 67
5.9 Moving a particle 68
5.10 Removing a particle from an elastic 71

6.1 The bounding tree for an object 79
6.2 A typical configuration of the chess pieces 83

6.3 Search Size vs. Relative Error 84
6.4 Search Size vs. Distance between Objects 85
6.5 Search Size vs. Number of Leaf Nodes 86

7.1 An illustration of the new constraint on the bubbles 89
7.2 An example B-spline 89
7.3 The construction of a B-spline path 92
7.4 A path for two links of a Puma 560 95
7.5 The time-optimal parameterization of a path 96
7.6 The effect of restricting the possible switching points 97
7.7 ODE integrations used to generate the trajectory 99
7.8 The time-parameterization generated when $\Delta t = 0.05$sec 101
7.9 Adding the option of maintaining a constant speed 101
7.10 Three Puma manipulators 104
7.11 The various processes of the system 108

B.1 The osculating circle 119

Chapter 1

Introduction

1.1 Background

One of the fundamental characteristics of an autonomous robot system is its ability to move without collision. An "intelligent" robot should avoid undesirable and potentially dangerous impact with objects in its environment. This apparently simple capability has been the subject of much research in robotics.

The problem of collision-free motion has generally been approached from two directions: planning and reactive control. With planning, before moving, the robot system uses models of the environment and itself to determine a feasible sequence of motions to achieve a specified goal. With reactive control, the motion of the robot for some small time period is determined just before the motion is executed, typically by examining the current sensor data. Such a division is vague and there are many systems that fall somewhere in the middle.

A key advantage of planning is that it enables a robot to achieve complex goals. For a robot to move from one configuration to another in a cluttered environment may require a long sequence of motions in which each step must be performed correctly. A classical example is moving a large piano into a small room; unless one thinks before acting, it is quite possible to end up in a situation where the piano cannot be moved into its desired position and one is forced to take the piano out of the room and start over.

Although planning is essential in many situations, it is generally infeasible to blindly follow the actions of a plan. Building a plan requires the ability to predict the future states of both the robot and the environment. Such predictions will be based on models of the world that inevitably contain errors and these errors may cause the plan to fail. As a consequence, relying solely on planning to avoid collision is not robust.

A reactive approach to robot motion is based on the idea of using sensing to interact dynamically with the environment. By relying on sensing, a robot can use simpler models of the world than required by a planner, and can thereby increase the robustness of its ability to avoid collision. The major limitation of many reactive approaches is their inability to

deal with the global problem of moving to an arbitrary goal.

In this thesis, we propose a novel architecture for combining planning and reaction in a complete robot system for executing motion tasks. A motion planner provides a global solution to move the robot to the goal. During the execution of the planned path, the path is modified in response to changes in the environment and unexpected obstacles.

1.2 Executing Collision-Free Motion Tasks

Suppose we desire to build a complete system to move a robot autonomously from an initial configuration to a goal configuration. The conventional architecture [38] for such a systems is illustrated in Figure 1.1. The user specifies a task and the planner, using a model of the world, generates a collision-free path which is converted to a feasible trajectory. A motion controller tracks the trajectory using feedback from sensory information about position and velocity. We note two points about such an architecture.

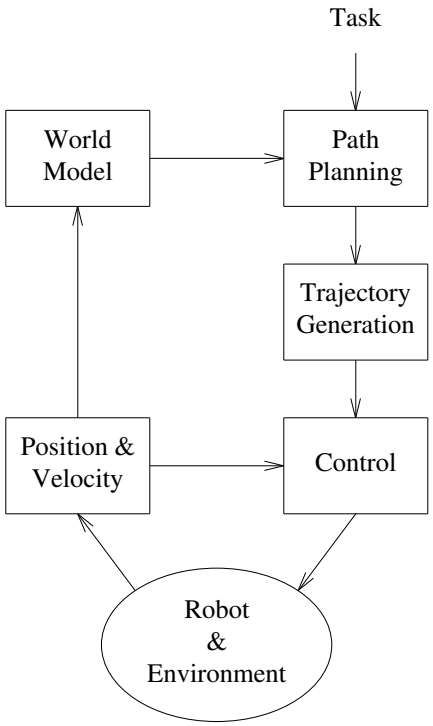


Figure 1.1: A conventional architecture for a robot system.

First, suppose a change in the environment is detected while moving along the path. One way to react to this situation is to halt the robot and plan a new path. In such an approach,

the responsiveness of the robot to changes would be dominated by the time needed to plan a path. Given the complexity of motion planning in general, this time may be rather large. An alternative is to add some type of reactive behavior to the controller. If the controller is to implement some sort of real-time obstacle avoidance scheme then it must be able to deviate from the path. Once the robot is off the path, it is unclear how the controller should use the original path to reach the goal.

Second, path planners are often designed to find any feasible path, with little attention to its suitability for execution. Often the path is represented as a sequence of linear segments and thus will contain many discontinuities in the direction of motion. For the robot to track such a path it must come to a complete stop at each of these discontinuities, greatly increasing the time to reach the goal. Other problems include paths that are unnecessarily long, or maintain little clearance from the obstacles.

1.3 A New Architecture

We propose a new architecture for building systems for executing collision-free motion tasks in which an intermediate level is interposed between the planning and control levels. At this intermediate level the path generated by the planner is treated as a deformable object. In real time, the path is modified in response to changes in the environment and to improve properties such as smoothness, length, and clearance from obstacles. The architecture is illustrated in Figure 1.2.

We can view this architecture as a hierarchy of three feedback loops with the environment. These feedback loops operate with reaction times that decreases from slow (at the planning level) to fast (at the control level). The three levels are:

- Path planning: A world model is used to generate global solutions to specified tasks.
- Path modification: The path from the planner is modified in real time to handle changes in the environment detected by sensors and to improve the path.
- Control: A motion controller is used to move the robot along the modified path.

This framework has the advantage of providing reactivity without limiting the robot's ability to achieve global goals. By modifying the path when changes in the environment are detected, the framework avoids the cost in time of recalling the path planner. The robot can react in real time to information obtained by sensors; however, while performing local motions a complete collision-free path to the goal is maintained.

Given a collision-free path, what characteristics do we desire from an incrementally modified version of the path?

The most important criterion is that the new path should be collision-free. Maintaining this condition, especially for robots with many degrees of freedom, is one of the major issues we address in later chapters.

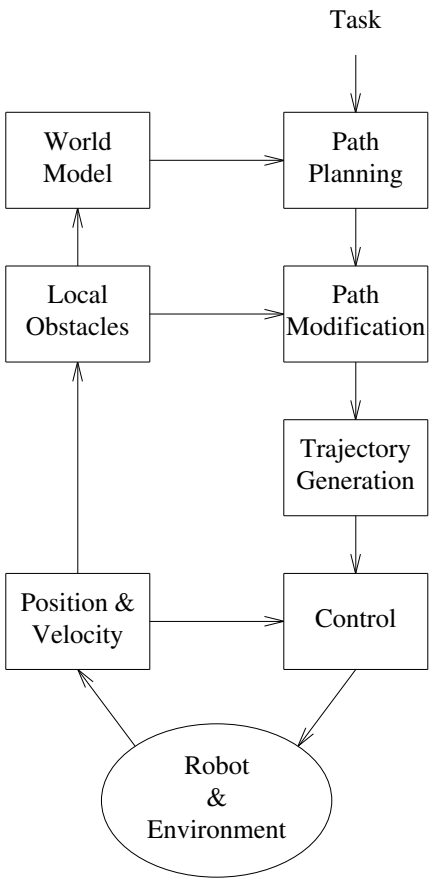


Figure 1.2: The proposed architecture for a robot system.

One extension to the collision-free criterion is to require the path to provide a certain amount of clearance from obstacles. Such a property recognizes that the underlying control system will not perfectly track a specified path. A situation may arise where the robot must come close to the obstacles, for instance, to squeeze through a small opening—more than minimal clearance is beneficial but not essential.

Other desirable properties of the path are smoothness, low curvature, and shortness. These properties will tend to reduce the amount of time the robot takes to reach the goal position. We should emphasize, however, it is a difficult problem to determine the path whose total execution time is minimal [6]; we aim to create paths that are reasonably good.

Depending on the application, there may be other desirable properties for the path. For example, we may desire the robot to stay well clear of internal joint limits. Although we limit our discussion to the basic properties mentioned above, the general approach we propose can handle other characteristics.

1.4 Elastic Bands

As the basis for determining the modification of a collision-free path, we use a physically-based model of a flexible string-like object subjected to internal and external forces. We call such an object an *elastic band* [50].

To illustrate the basic behavior of elastic bands, consider a planar robot that can translate, but not rotate, in an environment with obstacles. Suppose a motion planner has computed a path for the robot to move between two positions as depicted in Figure 1.3a. A controller would experience difficulty moving the robot along the path since there are discontinuities in the direction of the path which would require the robot to come to a complete stop.

To improve the shape of the path, two forces are applied: an internal contraction force and an external repulsion force. The contraction force simulates the tension in a stretched elastic band and removes any slack in the path. To counter the contraction force and to give the robot clearance around the obstacles, the elastic band is subject to a repulsion force from the obstacles. The two forces deform the elastic until equilibrium is reached as shown in Figure 1.3b.

These two types of forces also enable the elastic band to handle changes in the environment. The appearance of new obstacles or the detection of uncertainties in the environment change the forces on the elastic, causing it to deform to a new equilibrium position. In our example, Figures 1.3c and 1.3d depict the deformation of the path in the presence of a new moving obstacle.

Obviously, if the changes in the environment are large, the elastic band could fail to deform to a collision-free path, even if one exists. An example of this would be closing the door through which a robot had planned to move. A different path may exist, say through a different door, but finding such a path may require a global search. This problem is typical of local collision avoidance methods and is the primary reason that path planning is needed. In such a situation, the failure can be detected and a new path found by re-planning. However, for small changes, the elastic band is expected to deform to a good path that reflects the new state of the environment.

1.5 Other Applications for Elastic Bands

With elastic bands, collision-free paths are treated as dynamic objects that are continuously responding to the applied artificial forces. We developed elastic bands for the robot architecture shown in Figure 1.2, but one can conceive of other useful applications for this capability. By bringing collision-free paths to "life," we avoid the tedious task of specifying exactly the motion of the robot; rather, we specify an initial rough path and let the path evolve in real time towards a path that is better suited for the robot.

One simple application of elastic bands is as a post-processor for paths generated by motion planners or specified by users. In this scenario, we specify forces on the path that

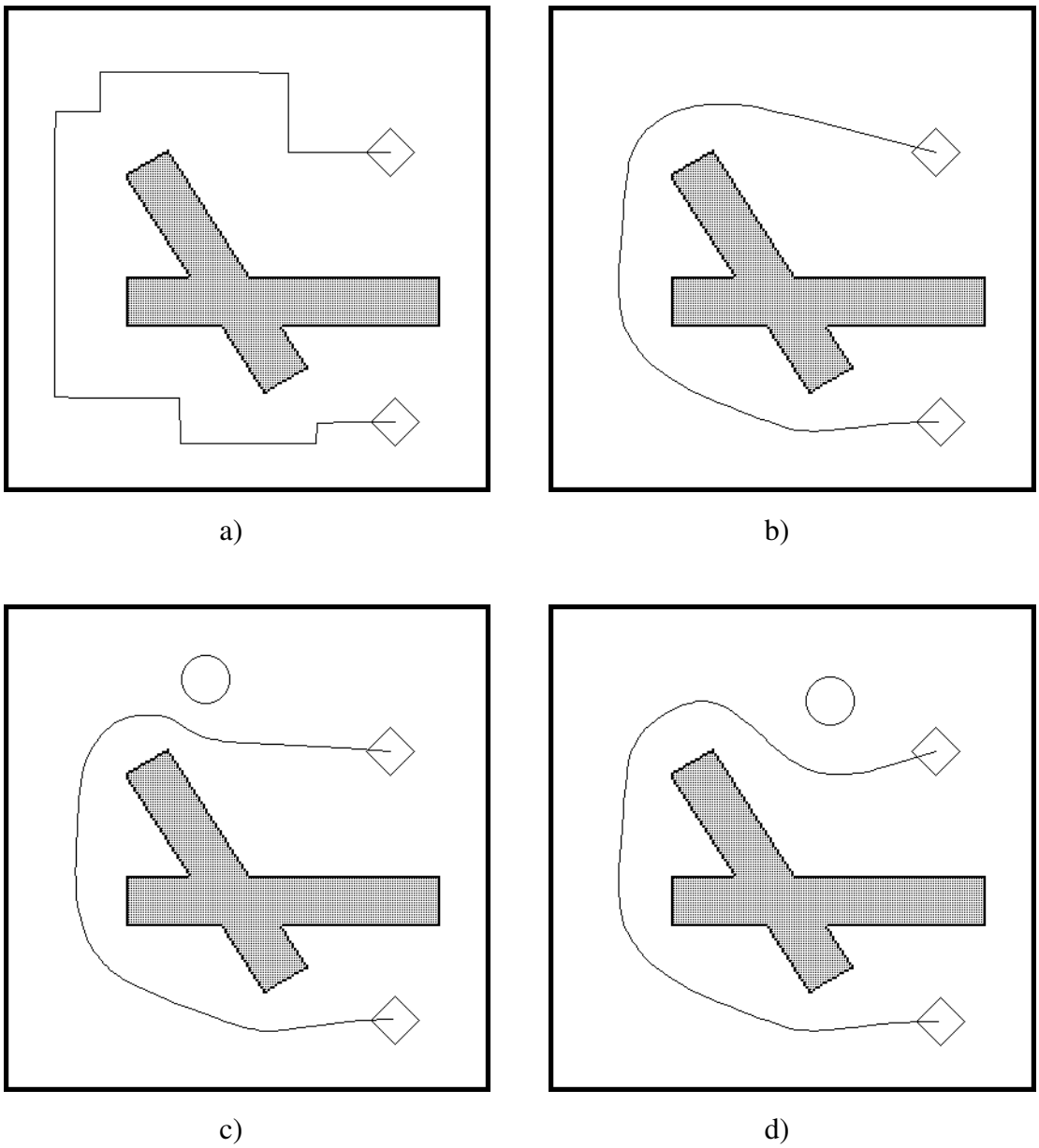


Figure 1.3: **a)** A path generated by a planner. **b)** Applying both an internal contraction force and an external repulsion force. **c, d)** A new moving obstacle deforms the path.

improve its shape and let the elastic deform until a static configuration is achieved. Such an application of elastic bands does not take advantage of the real-time nature of the path modifications, and more traditional optimization techniques may be equally suited to this task.

A more exciting application of elastic bands involves the interface between a robot and the user. Traditionally, users create paths for a robot by specifying a sequence of via points. Suppose instead that the user moves the robot roughly along the desired path. For a physical robot, this motion may be achieved by teach pendant or by floating the robot and moving it by hand. For off-line programming systems, the path may be entered by moving a computer model of the robot through the environment.

To record the motion, one end of an elastic band is attached at the start configuration and the other end to the robot. As the robot moves, the elastic band will be dragged out along the path. The elastic will not record the exact motion generated by the user, but instead will deform in real-time to an improved path.

Suppose, after viewing the motion along the elastic band, the user desires some modifications. By providing a mechanism to influence the forces on the elastic band, the user can simply push and pull the path into the desired shape. One possibility for providing this mechanism would be to allow the user to specify points that either attract or repel the path. This mechanism to interact with a deformable path seems both powerful and intuitive.

One may also envision using elastic bands to aid in updating paths when minor modifications are made to a robot work-cell. Rather than having to re-program all the robot motions, one could let the paths deform as the various fixtures and parts in the work cell are changed.

1.6 Overview of the Thesis

The greater part of this thesis deals with the design and efficient implementation of elastic bands. This topic has many interesting aspects, and along the way we shall propose several ideas which, we feel, have relevance in other areas of robotics.

An emphasis in the work described throughout this thesis is a desire for real-time performance. By real-time, we do not always mean that the algorithms we develop have bounded execution time; many do not. Rather, real-time is taken in the sense of interactive and applicable on-line using computer hardware that is available at the present time. Such emphasis has directly contributed to the development of ideas presented in this thesis.

Another requirement we placed on this research is that it should apply to robots with many degrees of freedom. Numerous problems in robotics have special-case solutions for robots with only a few degrees of freedom. Although they have relevance to real world problems, these solutions often do not extend to systems with a large number of degrees of freedom. We have tried to avoid these special case solutions and directly solve the more

general problem.

A brief outline of the various chapters in the thesis follows:

In Chapter 2, previous work on the collision-free motion of robots is reviewed. Basic concepts, such as generalized coordinates, configuration space, configuration space obstacles, and collision-free paths are summarized. Using these tools, approaches in reactive control, path planning, and hybrid systems are described.

In Chapter 3, a theoretical model for the elastic band is developed. The model is loosely based on the physics of a flexible string-like object in a field of forces.

In Chapter 4, the concept of *bubbles of free-space* is introduced. We derive methods for computing bubbles for a variety of robots including open-chain manipulators with many degrees of freedom such as the PUMA 560.

In Chapter 5, the implementation of elastic bands using bubbles is described.

In Chapter 6, a novel algorithm for distance computation between non-convex objects is presented.

In Chapter 7, a complete robot system we have built that uses elastic bands is described. One of the main points of interest is a real-time incremental algorithm for the generation of time-optimal trajectories along a given path.

In Chapter 8, we conclude with a summary of the contributions of this thesis.

In the Appendix A, the functional gradient operator $\overline{\nabla}$ is developed. This operator is used to determine the forces on a elastic band, given a potential functional over the space of possible paths.

In the Appendix B, the curvature vector is described.

Chapter 2

Collision-Free Motion of Robots

2.1 Introduction

The task of moving a robot to a desired position while avoiding collision with objects in the environment has been extensively studied. In this chapter, we briefly describe some of this previous work.

In Section 2 of this chapter, some of the basic concepts used throughout the robotics literature and this thesis are reviewed; these include: generalized coordinates, configuration space, free-space, configuration-space obstacles, paths, and trajectories.

In Section 3, a few of the approaches used for reactive control, including: artificial potential fields, the subsumption architecture and boundary following are presented. These methods provide real-time collision-avoidance in the presence of uncertain and changing environments.

In Section 4, an overview of the various approaches to path planning is provided. Using path planning and appropriate models of the world, it is possible to generate a collision-free path that the robot can follow to reach a specified goal.

In Section 5, several attempts to combine planning and reactive control in a hybrid system are described.

2.2 Basic Concepts

Throughout the robotics literature on collision-free motion, a few basic concepts appear repeatably. One needs to be able to describe where a robot is located, the set of possible locations that the robot could be, and a sequence of locations that the robot should move through to reach a desired goal. In this section, the conventional methods of formalizing these ideas are presented.

Generalized coordinates

In robotics, it is often required to describe the *configuration* of a robot with respect to the environment. Typically, a robot can be described as a collection of rigid bodies; for manipulator-type robots, these bodies are referred to as *links*. The configuration of a robot is the position and orientation of its various constituent bodies.

It is well known that the position and orientation of a rigid body in a three-dimensional space can be represented by six parameters [26]. For a robot that is composed of N rigid bodies, a configuration can be described by $6N$ parameters.

For most robots, there are constraints that limit the motion of the various constituent bodies; for example, links of a manipulator may be connected by joints. In general, each constraint can be viewed as placing restrictions on the allowable set of $6N$ parameters. A detailed study of the various possible types of constraints is beyond the scope of this thesis and we refer the interested reader to the excellent text of Lanczos [36].

A common form of constraint, called *holonomic*, can be eliminated by the appropriate selection of a set of n *generalized coordinates* $q_1 \ldots q_n$. These n parameters are sufficient to describe all valid configurations of the robot and are minimal in that no set of less than n parameters can describe all configurations. A holonomic robot with n *generalized coordinates* is referred to as an n*-degree-of-freedom* robot. A configuration of the robot is represented by the vector notation $\mathbf{q}$.

Consider the case of a six link manipulator such as the Puma 560. To represent a configuration of the robot, we describe the relative angle between neighboring links at each joint, a single number per joint. For such a robot there are six joints and thus there are six degrees of freedom. In contrast, to specify the position and orientation of each link in a three-dimensional space would require thirty-six parameters.

There are certain types of constraints that cannot be eliminated by the appropriate selection of coordinates. These constraints, referred to as *non-holonomic*, result in a situation where the number of parameters needed to describe the configuration of a robot is greater than the instantaneous number of degrees of freedom of motion. In this thesis, we restrict our attention to robots that are free of non-holonomic constraints.

An example of a non-holonomic mechanism is a car on a flat surface. Assuming a global reference frame is specified, the coordinates used to describe a configuration of the car might be the x, y coordinate of the center of the car, and an angle θ between the axis of the car and the x axis. It is not possible to describe a configuration with less than three parameters. On the other hand, at a given configuration, the motion of the car can be described by two parameters, say, the linear speed and the rate of turn; the car has only two degrees of freedom.

Configuration space

A powerful tool for developing solutions to problems in robotics is the *configuration space* representation. A given configuration of a robot is described by the values of a set of n generalized coordinates $q_1, \ldots, q_n$. We can consider these values as the coordinates of a particular point in a Euclidean space where the q's form the n coordinate axes. This n-dimensional space is referred to as the configuration space, denoted by the set C_{space}.

The concept of a configuration space is useful because it allows problems in robotics that deal with many bodies to be transformed into problems that concern only a single point. Configuration spaces were originally developed as a tool for analytical mechanics in the 19th century [26] and were popularized in robotics by Lozano-Pérez [40]. As described below, the basic motion planning problem can be succinctly formulated in terms of configuration spaces.

For a given configuration space, it is quite possible for two points to correspond to the same configuration of the robot. For example, joint angles are a natural set of generalized coordinates for robot manipulators such as the Puma 560. If these angles are expressed in radians, the addition of multiples of 2π to any coordinate will not effect the configuration of the robot. Mathematically, one can remove these redundancies by embedding the configuration space as a manifold in a higher-dimensional Euclidean space; see the text by Latombe [38] for details.

In practice, the non-Euclidean nature of most configuration spaces can be handled by limiting the range of the generalized coordinates and adding special code to handle wraps. In the case of manipulators with revolute joints, we could limit the joint angles to the range $[0, 2\pi)$. Special care must be taken to allow the robot to move from one end of this range and reappear on the other side.

Configuration space obstacles and free space

Typically, a robot moves in an environment that contains obstacles which must be avoided. These obstacles can be mapped into obstacles in a configuration space of the robot.

The configuration space obstacles consist of all points in C_{space} for which the robot at the corresponding configuration intersects with the obstacles in the environment. More formally, suppose the robot at configuration $\mathbf{q}$ and the obstacles are respectively described by the subsets $R_{\mathbf{q}}$ and O of the three-dimensional environment space $\mathbf{R}^3$. The set of configuration space obstacles, denoted by C_{obst}, is given by the equation

$$C_{obst} = \{\mathbf{q} \in C_{space} : R_{\mathbf{q}} \cap O \neq \emptyset\}.$$

Closely related to the configuration space obstacles is the concept of the free-space, denoted by the set C_{free}. The free-space is the region of C_{space} in which the robot is not in collision with the obstacles, i.e., C_{free} is the complement of C_{obst}.

As an example, consider the two-degree-of-freedom manipulator shown in Figure 2.1a surrounded by a few objects. Using joint angles q_1 and q_2 as generalized coordinates, the configuration space for this robot is the region $[0, 2\pi) \times [0, 2\pi)$. Figure 2.1b illustrates both C_{free} and C_{obst} for this robot in the given environment. The shaded region represents C_{obst}, while the white area is C_{free}.

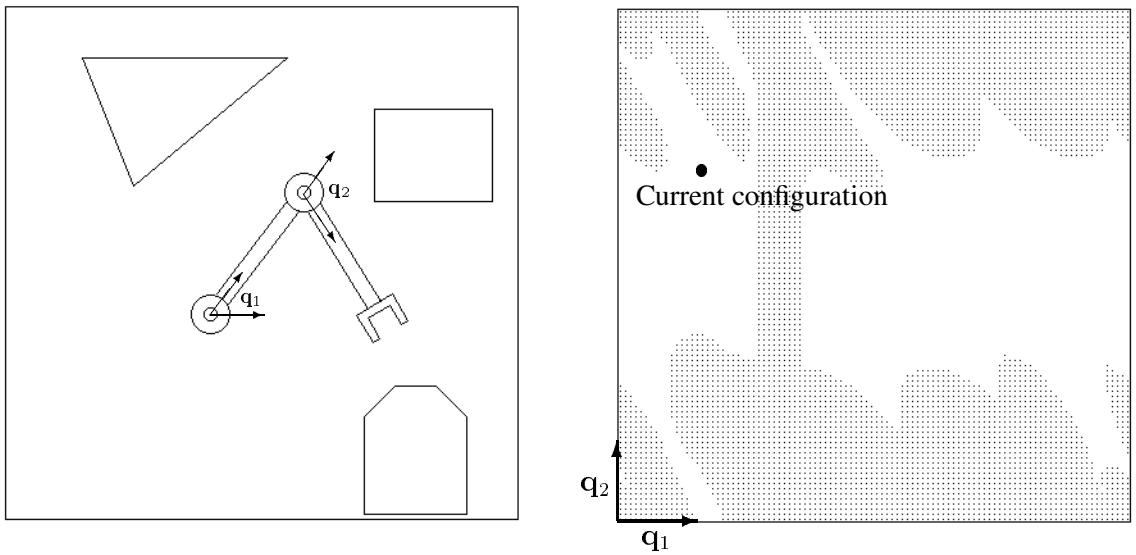


Figure 2.1: **a)** A manipulator with two degrees of freedom in an environment with obstacles. **b)** The joint space configuration space for the manipulator. The shaded region represents C_{obst}, while the white area is C_{free}.

The notions of a path and a trajectory

A *path* specifies a continuous sequence of configurations for a robot. More formally, a path from the initial configuration $\mathbf{q}_{start}$ to a goal configuration $\mathbf{q}_{goal}$ is defined as a smooth parameterized function $\mathbf{c}(s)$, with $\mathbf{c} : [0, 1] \rightarrow C_{space}$, $\mathbf{c}(0) = \mathbf{q}_{start}$, and $\mathbf{c}(1) = \mathbf{q}_{goal}$. The parameterization variable s does not represent any physical quantity, and the range of s is arbitrarily set to $[0, 1]$.

A path specifies the exact configurations the robot will occupy during a motion. Assuming a perfect model of the world is available, and the environment is static, it is possible to check if the motion of the robot is free of collisions. A path $\mathbf{c}(s)$ is collision-free if for

all $s \in [0, 1]$,

$$\mathbf{c}(s) \in C_{free}.$$

A *trajectory* is a path for which the parameterization variable is interpreted as time. A trajectory specifies not only where the robot should move, but also the speed of the motion, and is a standard representation used by robot control systems to specify a desired motion. A path can be converted to a trajectory by the process of time-parameterization; the parameterization variable s of a path $\mathbf{c}(s)$ is represented by a smooth scalar function of time, i.e., $s(t)$.

2.3 Reactive Control

Reactive control is a term we use to describe a wide variety of schemes that have been proposed to enable robots to move without collision. Although the term is vague, what these schemes have in common is a philosophy of determining the desired motion of the robot in real time by examining some up-to-date model of the world. As the model of the world changes, the robot reacts. Typically, the model of the world is determined by the robot's sensors. Also, the model may be local in that it is a function only of the current sensor information and does not contain global state that is determined over time. Reactive control goes under many names such as reactive behaviors, behavior-based control, sensor-based control, and local collision avoidance.

In the following, three reactive control schemes are described: artificial potential fields, the subsumption architecture, and boundary following.

Potential fields

The artificial potential field method proposed by Khatib [32, 33] is a popular approach for implementing real-time obstacle avoidance. The basic idea in this approach is to consider the robot to be moving in a field of forces. The forces are composed of two components: an attraction to the goal configuration, and a repulsion from the obstacles in the environment. Applying these "artificial" forces via the robot's actuators, the robot moves towards the goal while avoiding collision.

The potential field method can be viewed as increasing the capabilities of low level control. Instead of the control system simply tracking a given trajectory, the presence of obstacles is taken into account and the robot responds accordingly. The simple formulation of the avoidance strategy is such that it can be implemented efficiently and incorporated directly in the real-time servo-loop of the control system.

In the potential field method, the force on the robot is specified as the negative gradient of a potential function. More formally,

$$\mathbf{f}(\mathbf{q}) = -\nabla V(\mathbf{q}),$$

where $V(\mathbf{q})$ is a non-negative scalar function over the configuration space of the robot. The function V can be considered as specifying the potential energy of the robot at a configuration and hence the name for the approach.

An advantage of specifying a force as the gradient of a potential is that general statements can be made about the behavior of the system without knowing the exact sequence of configurations the robot will go through. In particular, if the configuration space is bounded and the potential function is time invariant, it can be shown [32] that the robot will move to a local minimum of the potential.

A key requirement for the above analysis is that the potential function is time invariant. In a typical application of the potential field method, the potential is a function of the environment, thus time invariance corresponds to a static environment. Even in the case where this condition is not meet, the potential function approach to defining forces still means that robot is continuously moving in a reasonable direction.

Another important property of potential functions is that they are additive. Potentials can be developed independently to provide capabilities for the robot. Using the sum of these potentials to control the robot will result in a behavior that attempts to combine the desired capabilities. For example, a potential may be developed to move the robot towards a goal, while a separate potential avoids collision with the obstacles. Combining these potentials provides a goal-seeking collision-avoiding behavior. If at a later date we desire the robot to avoid internal joint limit, this capability can be included by designing an appropriate potential and adding it in.

Adding potentials that have conflicting goals may cause the robot to become trapped in a local minimum. There may exist configurations in which the forces associated with the potentials sum to zero and consequently the robot does not move. For example, suppose the robot is attracted to a goal configuration and repelled from the obstacles. The combination of these behaviors will not enable the robot to achieve the goal in a situation such as depicted in Figure 2.2.

Local minima prevent the potential field method from directly solving the problem of collision-free motion of robots. The potential field method, however, has been the basis from which successful approaches to motion planning have been developed (see Section 4).

Subsumption architecture

Another popular method of building reactive systems is the subsumption architecture proposed by Brooks [12]. In this architecture, a robot control system is decomposed in terms of a hierarchy of behaviors or *levels of competence*. The idea is that additional levels can be added to an existing system to obtain more complex and sophisticated abilities. For example, Brooks proposes collision avoidance as the lowest level of control. The next higher level builds on this ability to wander aimlessly around without hitting objects. Next, the robot should be able to "explore" by moving in directions that seem "interesting."

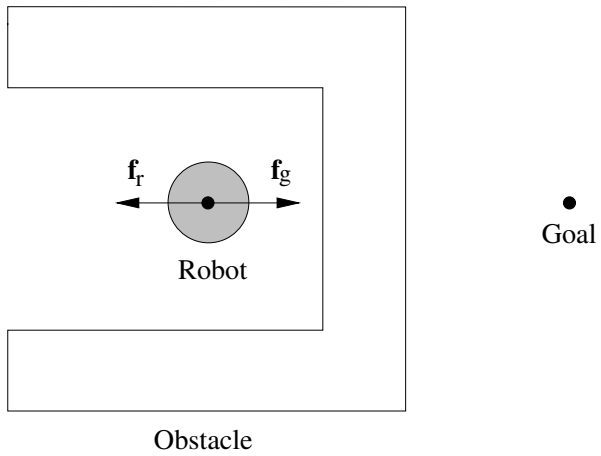


Figure 2.2: At the shown configuration, the attraction force $\mathbf{f}_g$ toward the goal and the repulsive force $\mathbf{f}_r$ are equal and opposite. As a result, the robot will not move. This corresponds to a local minimum in the potential function.

The behavior at each level of Brooks' architecture is intended to be self sufficient. Higher level behaviors are integrated into the system by a mechanism that can override the actions of the lower level; they can subsume the lower levels behaviors. The advantage of this approach is that the robot system can be built incrementally. Simple behaviors can be developed and debugged as a complete robot system. These simple behaviors can continue to operate, unaware that more complex behaviors are being built on top.

The individual layers are constructed from finite state machines connected by asynchronous communication lines. In addition, outputs from the state machines may be inhibited by other state machines, and inputs may be suppressed. This design for the layers enables them to be implemented by a loosely coupled set of simple microprocessors, which has advantages for mobile robots. The design can also be implemented efficiently on more conventional computer equipment.

One difficulty with the subsumption architecture is analyzing the global behavior of the robot. In particular, one would like to ensure stability and convergence towards the desired goals. Some progress has been made in developing tools for such analysis, for example Brock and Salisbury [10] use *behavior diagrams* to examine the behavior of a robot in all possible situations.

Boundary following

For the special case of a robot with a two-dimensional configuration space, Lumelsky [41] developed a reactive-control method that is guaranteed to converge to the goal configuration. The method does not require any knowledge about the configuration space obstacles; it only

assumes that the robot is equipped with a position sensor and a touch sensor. The basic strategy is as follows:

- Move towards the goal location in a straight line.
- If an obstacle is encountered, record the current configuration then start to move along the obstacle's boundary in a predetermined direction, say clockwise.
- As the robot moves along the boundary, keep track of the configuration that is closest to the goal. Continue until the robot returns to the original configuration at which the object was encountered, indicating the robot has circled the object.
- Moving along the object boundary, return to the point that was closest the goal. Resume moving towards the goal location.

Unfortunately, it is difficult to envision how this algorithm may be extended to situations in which the configuration space has dimension greater than two.

2.4 Motion Planning

One of the classical problems of robotics is the basic motion planning problem: given a perfect model of a robot in a static environment and two configurations, find a collision-free path between the configurations.

Solving the basic motion planning problem, often referred to as *path planning*, is important. The problem ignores many issues that appear when actually moving a robot, but any robot that can autonomously execute motion tasks will have to solve this problem to some degree (one could always simulate such a robot system and use the resulting motion as a solution path). The problem is well defined, and is one of the most active areas of research in this field.

It has been shown that a complete solution to the basic motion planning problem is computationally very expensive [51, 14]. Much progress has been made, however, in producing fast planners for practical situations by considering schemes that are not complete, i.e., may fail to find a path when one exits.

In this section, some of the basic approaches to path planning are described. We follow Latombe [38] and divide these approaches into three broad categories: roadmap, cell decomposition, and potential field.

Roadmap

The *roadmap* approach to path planning represents the free-space for a robot as a collection of connected collision-free paths. This set of collision-free paths, called a roadmap, is used

to plan a path as follows. A path is constructed from the start configuration to some part of the roadmap. Similarly, a path is constructed from the goal configuration to the roadmap. Next, using standard graph algorithms, the roadmap is searched for a path between the two points of connection on the roadmap. For a static environment, the roadmap is constructed once, and can be used to solve multiple planning problems.

The many variations of the roadmap approach differ mainly in the method for constructing the roadmap. These variations include: visibility graphs [43], Voronoi diagrams [44], freeways networks [11], and randomized roadmaps [31, 27]. The first three of these variations are among some of the earlier attempts to build path planners, however, they are applicable only for simple mobile robots with two or three degrees of freedom. Recently, there has been renewed interest in the roadmap approach in which the roadmap is constructed using randomized techniques. Such randomized roadmaps have been found experimentally to capture the structure of a robot free-space in a surprisingly efficient manner, even for complex robots with many degrees of freedom.

Cell decomposition

Until recently, the most common approach to path planning was based on constructing a cell decomposition of a robot's free-space. A *cell* is a region of the free-space with a simple shape such that a path can be easily constructed between any two configurations within the cell. By describing the free-space as a collection of cells, path planning can be reduced to a search of the graph representing the adjacency relationship between cells. Cell decompositions can be exact or approximate.

Exact decompositions use cells that can precisely represent the free-space. For all but the simplest configuration spaces, such cells must be described by complex analytical expressions. Planners based on exact cell decompositions tend to be more of theoretical interest as they are complex to implement and are extremely inefficient [55]. On the other hand, such planners prove the existence of algorithms that can exactly solve the path planning problem.

Approximate decompositions use some simple cell shape, typically a rectanguloid, to represent the free-space up to a given resolution. The regular shape of the cells results in simplified algorithms for generating and representing the decomposition. A planner that uses an approximate cell decomposition may fail to find a path when one exists, however, such failure occurs only when the robot must move through a region of the free-space that is smaller than the resolution of the decomposition, a parameter the user can control.

The classical approximate cell decomposition algorithm, proposed by Brooks and Lozano-Pérez [13], is as follows. The configuration space is divided into rectanguloid cells at successive levels of approximation. Cells are divided into three categories: EMPTY, FULL, and MIXED. EMPTY cells are completely contained in C_{free}, FULL cells are completely contained in C_{obst}, and the remaining cells are MIXED. The planner iterates

between searching for a sequence of adjacent EMPTY cells connecting the start and goal configurations, and dividing MIXED cells into smaller cells that are then classified into one of the three categories. The planner terminates when a sequence of EMPTY cells is found, or when some level of resolution is reached. The many variations of this algorithm differ mainly on how MIXED cells are selected for division and how they are divided [38].

The major limitation of cell decomposition planners is that the representation of the cells tends to grow exponentially with the dimension of the configuration space. This property effectively limits the planners to robots with no greater than perhaps four degrees of freedom.

Potential field

The artificial potential field method was described earlier in this chapter as an approach for implementing reactive control. With potential fields, the robot is attracted to the goal position and repulsed from the obstacles in the environment. Such a scheme can be implemented to operate in real time, and can be incorporated directly into a robots control system.

For many simple situations, the potential field method will successfully move a robot to the desired goal configuration. Since this capability is achieved without costly representations of the free-space, there has been much interest in adapting the potential field method to path planning. The major difficulty that must be overcome is the presence of local minima in the potential (see Section 3).

One approach to overcoming the problem of local minima is the design of special potential functions called *navigation functions*. Navigation functions have only one minimum and hence the robot will move to the desired configuration. Analytical navigation functions have only been developed for restricted environments; for example, Rimon and Koditschek [52, 53] present such potentials for environments consisting of disjoint spherical and star shaped objects. Numerical navigation functions can be developed over a discrete representation of the configuration space [4]. These numerical navigation functions, however, require an exponential amount of memory with respect to the dimension of the configuration space, and are only practical when the dimension is not greater than say four.

A more successful approach to using potential fields for path planning, proposed by Barraquand and Latombe [4], utilizes random walks to escape local minima. The idea is to follow the gradient of a potential function until a minimum is reached. If the minimum is not the goal configuration, a random walk is performed, followed by another gradient descent of the potential until a new local minimum is reached. If the new local minimum is still not the goal, the process is repeated from the lower of the two local minima. The planner continues until the goal is attained or some time limit is reached. One can view such a planner as using the potential field as a heuristic. A local minimum corresponds

to a situation where the heuristic cannot suggest any further course of action, thus random motion seems appropriate.

The randomized path planner has been used to plan paths for complex, many-degree-of-freedom robots. By its nature, however, the performance of such a planner can only be determined empirically. The planner does not guarantee to find a path if one exists, and there are many parameters, such as the length of each random walk, that influence the efficiency of the planner in a specific situation. Of particular importance is the specification of the potential function. Barraquand and Latombe [4] describe a variety of potentials that they found to be successful. First, numerical navigation potentials are constructed for various *control points* on the robot; these potentials are of dimension two or three, and thus are computational feasible. Next, the potential for a configuration of the robot is calculated by combining the low-dimensional navigation potentials using an *arbitration function*.

The paths produced by the randomized path planner consist of an alternating sequence of gradient descent and random walks. Obviously, it is advantageous to smooth these paths before moving the robot. As mentioned in the Chapter 1, elastic bands could provide an efficient method of performing such a post-processing.

2.5 Combining Planning and Reactive Control

When building a robot system, we ideally would like to combine both path planning and reactive control. Path planning provides the ability to move to specified goal positions, even in the presence of complex obstacles. Reactive control provides robust performance in order to deal with uncertainties and unexpected obstacles while executing the planned path.

The problem with combining planning and reactive control is that a path specifies exactly where a robot should move to reach the goal. Apart from tuning the velocity along the path to avoid collision with moving obstacles [28], a reactive controller will need to move off the original path. Deviating from the path, however, may or may not have severe repercussions with respect to reaching the goal position; a path provides no information about the degree of flexibility in the planned motions.

One common approach to combining planning and reaction involves replacing paths as the specification of the planned motion of a robot. By designing a representation that reduces the level of commitment inherent in a path, a reactive controller can adapt the motion of the robot in response to information obtained during execution while still following the original plan. Note that the architecture based on elastic bands, proposed in Chapter 1, does not fit into this category—a path is still used to represent the planned motion, however during execution, the path itself is modified.

Krogh and Thorpe [35] propose replacing a path with a sequence of critical points. The potential field method is then used to move the robot through the sequence of points. As

the robot moves between points, the potential field controller allows the robot to deviate from the original path. The drawback of this approach is that the freedom provided by the representation is difficult to control.

An improved representation, proposed by Choi et al. [57, 15], is the *channel*—a sequence of adjacent cells of free-space connecting the start and goal configurations. As the robot moves through a channel, the robot has a region of the free-space in which it is free to move. By designing a suitable potential field within the channel, the robot can use this freedom to avoid unexpected obstacles.

A drawback of the channel method is that the range of contingencies possible during the motion of the robot is artificially restricted. The boundary of the channel is established before the robot begins moving, and the placement of the boundary is more a result of the planning algorithm used, rather than representing some actual constraint on the robot's motion. During the motion of the robot, the environment may evolve in such a way that the channel boundary becomes an unnatural constraint. As an extreme example, suppose all the obstacles in the environment were to be removed. Ideally, the robot should then move directly to the goal position. This will not occur with channels.

The channel method was designed with the implicit assumption that the path planning system would be based on a cell decomposition. This restriction is not inherent in the approach, as one could envision building a channel directly from a collision-free path. In fact, in Chapter 4, we present such an algorithm for performing this construction.

Chapter 3

Elastic Bands

3.1 Introduction

In this chapter we develop a theoretical model for *elastic bands*. The model is loosely based on the physics of flexible materials under the influence of external forces. At the end of the chapter we compare this approach with an alternative point of view based on optimization theory.

The model we develop is theoretical in the sense that it is continuous and cannot be directly implemented on a computer. The continuous case is interesting as paths are defined, at least formally, as functions. To specify an arbitrary path is equivalent to describing an arbitrary function, which cannot be achieved with a discrete representation. In subsequent chapters we utilize the theoretical model to develop a computationally feasible discrete model that approximates the continuous case.

As stated in the introduction chapter, we use physics as the inspiration for the development of elastic bands. The rationale for this approach is its intuitive appeal. The behavior of objects in the real world is well understood and can be exploited to assist in the development of an algorithm that solves a specific problem.

The development of physical-based algorithms is quite common in computer science. A classic example is the optimization technique of simulated annealing [34]. Simulated annealing models the physics of cooling metals and freezing liquids to effectively find near-optimal solutions to difficult problems such as the traveling salesman and circuit layout. These problems are characterized by many spurious local minima and were considered difficult for traditional optimization techniques.

One should remember that we are using a model of a physical process only as the foundation for an algorithm. As a model of elastic bands is developed, we will slowly depart from an exact physical model. This is not a drawback; we are not interested in accurately simulating some physical system as is the case with many engineering applications. For example, simulated annealing is based on the behavior of thermodynamic systems, but it is

not a true simulation in the sense that it does not accurately model the actual behavior of metal annealing or freezing liquids. When it seems appropriate, the underlying physics of the simulation may be modified to suit the task at hand.

3.2 Relation to Previous Work

In previous research, models of deformable materials have been used for applications in computer vision and graphics. One well know example is the *snake*, as developed by Witkin et al [30]. A snake is a piece of stretchy, springy wire moving in a field of forces generated from digital images. The image forces move the snake towards features of interest in the image such a contours while the internal forces maintain smoothness.

The elastic band model adopts the same basic approach as snakes. Elastic bands, however, are designed for a different application and as such share few implementation details. A key difference is the condition that an elastic band must represent a collision-free path. Such a constraint is difficult to maintain and leads to the development of an innovative idea called *bubbles*.

The elastic band model is also similar to the artificial potential field method described in the previous chapter. As in potential fields, artificial forces are used to influence the behavior of the robot. The principal difference is that the robot is influenced indirectly via the modification of a path. In this way, global information about how to achieve the goal can be maintained while reacting to changes in the environment. In other words, we can integrate the powerful reactive capabilities of the potential field method with the goal information necessary to achieve the motion to a global configuration in a complex environment.

3.3 A Model for an Elastic Band

An elastic band is a deformable collision-free path under the influence of forces. To build a model for an elastic band we need to consider the nature of the applied forces and their effect on the shape of the path. In this section these two issues are considered.

Recall from the last chapter the mathematical description of a path. A parameterized path $\mathbf{c}(s)$ is defined as a smooth function $\mathbf{c} : [0,1] \rightarrow C_{space}$, with $\mathbf{c}(0) = \mathbf{q}_{start}$, and $\mathbf{c}(1) = \mathbf{q}_{goal}$.

We intend to consider a path as a physical object. The first thing to notice is that a path is specified as a continuous function and thus, in general, cannot be described by a finite or even countably infinite configuration space. In physics, the motion of such objects is the subject of continuum mechanics.

Suppose we consider a path to define the configuration of a one-dimensional continuous solid where each value of s in the range $[0,1]$ corresponds to an infinitesimal particle in the

object. The position of each particle is given by the function $\mathbf{c}(s)$. Note that the particles of an elastic band move in an n-dimensional configuration space that, as we saw in Chapter 2, may or may not be Euclidean.

At a given instant in time, we wish to apply a force to each particle of the elastic band. The force on a given particle can be represented as an n-dimensional vector in which each element corresponds to the force applied along the corresponding axis in the n-dimensional configuration space. In general the force will vary along the elastic band, and thus we represent the force $\mathbf{f}(s)$ as a smooth function $\mathbf{f} : [0, 1] \rightarrow \mathbf{R}^n$, which is referred to as a force function.

After determining the force function on the elastic, we need to determine the evolution though time of the shape of the path. Instead of attempting to simulate the full dynamics of a true physical system, a pseudo-static approach is used.

Internal and external forces

The force applied to a particle of an elastic band can be decomposed into the sum of component forces, the most important division being between an *internal force* and an *external force*. We can consider the particles of the elastic band as a system. The internal force is the result of interaction between particles within the system while external force is due to sources outside the system. The total force function can be written

$$\mathbf{f}(s) = \mathbf{f}_{int}(s) + \mathbf{f}_{ext}(s)$$

where $\mathbf{f}_{int}$ and $\mathbf{f}_{ext}$ represent the internal and external force functions respectively.

The division of the force function into internal and external components emphasizes the relationship between the elastic band model and the potential field method. We can consider each of the particles of the elastic band to be under the influence of a potential function represented by the external force. Each particle moves under the influence of the external force; however, the internal force maintain the constraint that the particles form a continuous curve.

The internal force determines the properties of the artificial "material" that an elastic band is made of. In the example from the introduction chapter, the internal force was described as a contraction force similar to that of a piece of elastic rubber, and hence the name for our model. As the elastic band is a one-dimensional curve, we restrict the interaction between particles of the system to those that are within an infinitesimal distance of each other; such interaction will be a function of the local shape of the elastic. Mathematically, this corresponds to restricting the internal force to be a function of the derivatives of the path with respect to the spatial parameterization variable s. The internal force can thus be written as

$$\mathbf{f}_{int}(\frac{d\mathbf{c}}{ds}, \frac{d^2\mathbf{c}}{ds^2}, \frac{d^3\mathbf{c}}{ds^3}, \ldots),$$

or more concisely

$$\mathbf{f}_{int}(\mathbf{c}', \mathbf{c}'', \mathbf{c}''', \ldots).$$

The external force reflects the effect of the environment on the path. For each particle, the force is independent of the other particles in the elastic. We restrict the external force so that it is dependent on only a particle's position in the configuration space; the external force has the form

$$\mathbf{f}_{ext}(\mathbf{c}(s)).$$

The external force specifies how we want each particle to move regardless of the other particles. This is not to say that the external force on a particle will have no effect on neighboring particles. As particles move, the shape of the path will change. The internal force, which is a function of the shape, will cause neighboring particles to be affected.

Motion of an elastic band

After the forces on the elastic band are determined, the resulting motion of the path needs to be calculated. Mathematically, the shape of the elastic band and the applied force can be considered as function of both a spatial parameterization variable s and a temporal parameterization variable t, in other words, $\mathbf{c}(s,t)$ and $\mathbf{f}(s,t)$ respectively. The motion of the elastic band can then be described by a set of partial differential equations relating $\mathbf{c}$ and $\mathbf{f}$.

From a physical point of view, the motion of the elastic band can be determined by an application of Newton's laws of motion. For each infinitesimal particle s on the elastic, a mass $\mu(s)$ is specified. Applying the classical $\mathbf{f} = m\mathbf{a}$ law gives

$$\mathbf{f} = \mu \frac{\partial^2 \mathbf{c}}{\partial t^2}. \tag{3.1}$$

Assuming there are sufficient initial conditions, the partial differential equation (3.1) can be integrated twice with respect to time to determine the motion of the elastic band. Such an approach is equivalent to a *dynamic simulation* of our artificial elastic material under the influence of forces.

We use a pseudo-static model to determine the motion of an elastic band. To integrate equation (3.1), the time rate of change $\partial \mathbf{c}/\partial t$ of the elastic band must be known. Instead, we assume that the elastic band is at rest. A Taylor series expansion of $\mathbf{c}$ gives

$$\mathbf{c}(s, t + \Delta t) = \mathbf{c}(s,t) + \Delta t \frac{\partial \mathbf{c}}{\partial t} + \frac{1}{2}\Delta t^2 \frac{\partial^2 \mathbf{c}}{\partial t^2} + \cdots$$

If at time t the particle is at rest, i.e., $\partial \mathbf{c}/\partial t = 0$, and high order terms are ignored, the initial instantaneous motion of each particle will be in the direction of the acceleration $\partial^2 \mathbf{c}/\partial t^2$.

From equation (3.1) it can be seen that direction of acceleration is also the direction of the force, thus mathematically we have

$$\Delta \mathbf{c} \propto \mathbf{f}. \tag{3.2}$$

The behavior of such a system can be approximated by the partial differential equation

$$\frac{\partial \mathbf{c}}{\partial t} \propto \mathbf{f}. \tag{3.3}$$

Such an approach ignores the dynamic components of the motion of the elastic band, but retains the same steady state configurations or low frequency behavior. The analogous physical situation is to imagine that the elastic band is immersed in a viscous fluid.

The use of pseudo-static motion rather than simulating the full dynamics is the same simplification that is used when applying the potential field method to path planning. The justification for this approach is that for path planning and elastic bands, the dynamic behavior of the simulated objects is of little interest, thus it appears unnecessary to spend effort simulating it.

3.4 Using Potential Functions to Specify Forces

In the artificial potential field method, the force on the robot is specified using the negative gradient of a potential function, that is

$$\mathbf{f}(\mathbf{q}) = -\nabla V(\mathbf{q}).$$

Under suitable conditions, subjecting the robot to a force defined in this manner will result in the robot moving to a local minimum in the potential. This is a powerful property—a potential can be designed to meet various criteria, and even though the effect of the resulting forces is extremely complex, the global behavior of the system can still be analyzed.

We would like to specify force function acting on an elastic band in a similar fashion. The difficulty is that the configuration of the elastic band is specified by the parameterized function $\mathbf{c}(s)$ and not some finite number of coordinates. The potential should considers the entire configuration of the elastic band and returns a scalar value. As the state of the elastic band is itself a function, the potential is a function of a function, often referred to as a functional. The gradient operator can not be applied to functionals.

Luckily, we can utilize variational calculus to develop an operator that has many of the same properties as the gradient operator ∇, but applies to functionals. Although this development is interesting, it is rather mathematical in nature and adds little to an understanding of the elastic band idea. For this reason, the development of the operator appears in Appendix A; the following few paragraphs state the relevant results.

One natural way to specify the potential functional is to define a potential density $v(s)$ for each particle on the elastic band and then integrate this density over the elastic. Like the

force functions considered in the previous section, the energy density can be divided into internal and external components. The internal component is a function only of the shape of the elastic, while the external component is a function of the position of a particle in the configuration space. Mathematically we have

$$v(s) = v_{int}(\mathbf{c}', \mathbf{c}'', \mathbf{c}''', \ldots) + v_{ext}(\mathbf{c}),$$

where v_{int} and v_{ext} are the internal and external potential density functions respectively. The potential functional $V[\mathbf{c}]$ over the elastic is given by

$$V[\mathbf{c}] = \int_0^1 v(s)\, ds.$$

For potential functionals such as V, we can derive a functional gradient operator $\overline{\nabla}$ given by

$$\overline{\nabla} v = \frac{\partial v}{\partial \mathbf{c}} - \frac{d}{ds}\frac{\partial v}{\partial \mathbf{c}'} + \frac{d^2}{ds^2}\frac{\partial v}{\partial \mathbf{c}''} - \cdots + (-1)^n \frac{d^n}{ds^n}\frac{\partial v}{\partial \mathbf{c}^{(n)}} + \cdots. \tag{3.4}$$

This operator has many analogous properties to the well-known gradient operator as applied to functions of several variables, some of which are listed below:

1. Given a function $g(\mathbf{x})$ and a unit vector $\mathbf{n}$, the derivative of g in the direction $\mathbf{n}$ is defined as
$$\frac{dg(\mathbf{x} + t\mathbf{n})}{dt} = \nabla g \cdot \mathbf{n}.$$
In a similar fashion, suppose we let $\boldsymbol{\eta}(s)$ be a function that maps the interval [0, 1] into the same n-dimensional space as $\mathbf{c}(s)$, and has the property that
$$\int_0^1 \|\boldsymbol{\eta}\| = 1.$$
We can view $\boldsymbol{\eta}$ as a unit function, and define the *derivative of* V *in the direction of* $\boldsymbol{\eta}$ as
$$\frac{dV(\mathbf{c} + t\boldsymbol{\eta})}{dt}.$$
Using the functional gradient operator, the direction derivative is given by the expression
$$\int_0^1 \overline{\nabla} v \cdot \boldsymbol{\eta}\, ds.$$

2. For a function $g(\mathbf{x})$, the direction of maximal increase is given by the vector ∇g. Similarly, the direction that causes the greatest increase in V is given by the function $\overline{\nabla} v(s)$.

3. If x is a function of time t, the change in the function $g(\mathrm{x}(t))$ is given by the equation

$$\dot{g} = \nabla g \cdot \dot{\mathrm{x}}.$$

Similarly, if $\mathbf{c}$ is a function of both s and t, then $V[\mathbf{c}] = V(t)$ is a function of t. We may now consider the derivative of V with respect to t. Using the functional gradient, this quantity can be expressed as

$$\dot{V}(t) = \int_0^1 \overline{\nabla} v \cdot \dot{\mathbf{c}}\, ds.$$

4. The vector x is a stationary value of g if

$$\nabla g(\mathbf{x}) = 0.$$

An elastic band $\mathbf{c}$ is in a stationary configuration if, for all values of s,

$$\overline{\nabla}\mathbf{c}(s) = 0.$$

By stationary, we mean that any infinitesimal changes in $\mathbf{c}$ will not result in a change in $V[\mathbf{c}]$.

5. A potential function can be considered as defining an energy function over the configuration space. If the force on a particle is specified by

$$\mathbf{f}(\mathbf{x}) = -\nabla g,$$

then the function g can be viewed as the potential energy of the particle. In particular, the work done in moving a particle from configuration x_0 to x_1 in the presence of the force f is given by the expression

$$g(\mathbf{x}_0) - g(\mathbf{x}_1).$$

Similarly, if the force on a particle of the elastic is specified by

$$\mathbf{f}(s) = -\overline{\nabla} v,$$

then the functional $V[\mathbf{c}]$ defines a potential energy over the space of possible paths. The work done in deforming a path $\mathbf{c}_0$ to a path $\mathbf{c}_1$ in the presence of the force function $\mathbf{f}$ is given by the expression

$$V[\mathbf{c}_0] - V[\mathbf{c}_1].$$

As V is the integral of v over the path, we label v an energy density function.

3.5 An Elastic Band Made from Idealized Elastic Material

Suppose the elastic band is made of an idealized elastic material with the following two properties:

1. Its unstrained state corresponds to zero length.
2. When deformed, the strain energy takes the standard form $\frac{1}{2}kx^2$, where k is the constant of elasticity and x is the magnitude of the deformation of the material.

For such a model of the elastic band, we can specify the internal energy density by the equation

$$v_{int} = \frac{1}{2}k\|\mathbf{c}'\|^2. \tag{3.5}$$

To see how equation (3.5) is derived, notice that $\|\mathbf{c}'\|$ can be thought of as describing the deformation of the infinitesimal particle s; a large value of $\|\mathbf{c}'\|$ indicates the particle is "stretched out" and "thin". These terms, of course, are intended only as mental aids for visualizing the mathematical equations.

Using the functional gradient operator, the corresponding force can be determined for a particle of the elastic band. As v_{int} in equation (3.5) contains only the first derivative of $\mathbf{c}$, $\overline{\nabla}$ has the simplified form of

$$\overline{\nabla}v = -\frac{d}{ds}\frac{\partial v}{\partial \mathbf{c}'}. \tag{3.6}$$

Applying (3.6), it can be seen that the contraction forces f_{int} is given by,

$$\begin{aligned} \mathbf{f}_{int}(s) &= -\overline{\nabla}v_{int} \\ &= \frac{d}{ds}\frac{\partial v_{int}}{\partial \mathbf{c}'} \\ &= k\mathbf{c}''. \end{aligned} \tag{3.7}$$

Now consider the external force. As stated above, the external force for a particle on the elastic is restricted to be a function of only its position in the configuration space. Given this restriction, it is easy to see that the external energy density must also be a function only of the particle's position. For such an energy function, the operator $\overline{\nabla}$ is identical to the standard gradient operator ∇, giving

$$\begin{aligned} \mathbf{f}_{ext}(s) &= -\overline{\nabla}v_{ext}(\mathbf{c}) \\ &= -\nabla v_{ext}(\mathbf{c}). \end{aligned} \tag{3.8}$$

In other words, the external force on a particle of the elastic band is specified in the same fashion as the force on a robot in the artificial potential field method. This is to be expected as an elastic band can be viewed as an infinite number of robots, each under the influence of an

external potential field while the internal force provide the connection between neighboring robots.

In summary, our first model for an elastic band specifies the force on a particle of an elastic by the equation

$$\mathbf{f}(s) = k\mathbf{c}''(s) - \nabla v_{ext}(\mathbf{c}(s)),$$

where k is the constant of elasticity and v_{ext} is a user-specified potential over the configuration space of the robot.

3.6 A Problem with the Elastic Band Model

The model that we have developed above represents the forces on an elastic band made of an idealized linear elastic material. Such a material seems physically appealing as we expect it to behave in a manner similar to actual physical objects. Of course, the property that the material has an unstrained state of zero length is somewhat unrealistic, but this can be seen as the extreme case of materials that have short unstrained lengths that are then stretched considerably.

Unfortunately, the model based on an idealized elastic band has a serious problem when used for our application of path modification. The problem results from the variation in the tension in the elastic band as the total length of the elastic changes.

Consider stretching a physical object that is similar to our elastic band. Intuitively, as the length increases, the internal tension becomes greater. One result of an increase in tension is that it becomes more difficult to deform the shape of the object.

In terms of our elastic band model, increasing the length of the elastic will decrease the deformation due to a given external force. In particular, if the external force is used to repel the elastic from the obstacles in the configuration space, the distance from the obstacles at which the repulsive force and the contraction force balance each other will depend on the length of the elastic.

This property is undesirable for two reasons. First, the contraction and repulsion forces are designed to produce an elastic that is short, smooth, and has clearance from the obstacles. As discussed previously, these criteria conflict with each other. By allowing the length of the elastic to influence the amount of external force needed to modify the shape, it becomes difficult to design the external force and set the constant of elasticity k such that a suitable compromise is achieved.

Second, changes in the environment may cause non-local changes in the path. Suppose a new obstacle appears and starts to deform an elastic. If this deformation substantially changes the length of the elastic, the tension will increase and we expect the entire elastic band to change shape. In the case of a repulsion external force, changes in one segment of the elastic may cause the entire path to move closer to the obstacles. Such global deformations of the elastic seem undesirable.

3.7 A New Model

We now consider a new model for elastic bands that is slightly less physical. In this model, the internal contraction force is designed so that the tension in the elastic is always constant. It is difficult to imagine an actual material with this property and such an artificial "material" has its own problems. However, we feel this model is well suited to the task of modeling a deforming collision-free path.

Consider a material that has constant tension when deformed. The strain energy for such a material has the form kx, instead of the usual $\frac{1}{2}kx^2$, where k is the constant of elasticity and x is the magnitude of deformation. For a model of the elastic band, internal energy density can be specify by the equation

$$v_{int} = k\|\mathbf{c}'\|.$$

Using the functional gradient operator $\overline{\nabla}$, the corresponding force on a particle of the elastic can be determined. Applying (3.6), the contraction force f_{int} is given by

$$\begin{aligned} \mathbf{f}_{int}(s) &= -\overline{\nabla} v_{int} \\ &= \frac{d}{ds}\frac{\partial v_{int}}{\partial \mathbf{c}'} \\ &= k\frac{d}{ds}\frac{\mathbf{c}'}{\|\mathbf{c}'\|} \\ &= \frac{k}{\|\mathbf{c}'\|}\left(\mathbf{c}'' - \frac{\mathbf{c}''\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}'\right). \end{aligned} \tag{3.9}$$

The force described by equation (3.9) is almost equivalent to the *curvature vector* of $\mathbf{c}$. In Appendix B we describe the notion of a curvature vector in detail, however, the basic idea is the direction and extent that the curve bends at a given point is given by the vector

$$\boldsymbol{\kappa}(s) = \frac{1}{\|\mathbf{c}'\|^2}\left(\mathbf{c}'' - \frac{\mathbf{c}''\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}'\right). \tag{3.10}$$

This equation can be interpreted as follows. First notice that the expression

$$\frac{\mathbf{c}''\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}' \tag{3.11}$$

represents the component of $\mathbf{c}''$ along the unit vector $\mathbf{c}'/\|\mathbf{c}'\|$, which is the direction of the path at the given point. Subtracting (3.11) from $\mathbf{c}''$ gives the component of the acceleration that is not along the path. Scaling this vector by one over the square of the magnitude of the velocity along the path gives the curvature of the path. The curvature vector for a point on a curve is a function only of the shape of the curve at that point and not the particular parameterization that is used to describe it.

Comparing equations (3.9) and (3.11), we see that the internal force can be expressed as

$$\mathbf{f}_{int}(s) = k \, \|\mathbf{c}'\| \, \boldsymbol{\kappa}(s).$$

The force $\mathbf{f}_{int}$ has the same direction as $\boldsymbol{\kappa}$, but its magnitude is scaled by $\|\mathbf{c}'\|$ and the constant k. Recall that for an elastic $\mathbf{c}$, the quantity $\|\mathbf{c}'\|$ represents the local amount of stretch in the elastic. In other words, the magnitude of the internal force is still dependent on how stretched the elastic is, and the problem described in section 3.6 still applies. However, the degree of coupling is much reduced from the original force defined by equation (3.7).

Given that the tension in the elastic band is constant, it maybe surprising that the internal force varies with the degree of stretch in the elastic. One way to picture the situation is as follows. Each infinitesimal particle experiences a force from its two neighboring particles. As the tension is constant, these forces are of equal magnitude The direction of the two forces, however, are not quite opposite due to the curvature of the path; the discrepancy in the directions results in a non-zero resultant force. If the elastic band is stretched in some region while maintaining the same shape, the neighboring particles will move further apart and the discrepancy in the directions of the two forces will increase. For the elastic to be in equilibrium, the magnitude of the external force must increase.

Ideally, we would like an internal force that is a constant multiple of the curvature vector $\boldsymbol{\kappa}$. Such a force is appealing as it is proportional to the degree of bending in the elastic and is dependent only on the elastic's shape and not on the distribution of particles along it. Unfortunately, we have not found an energy density that describes such a force. Without such an energy density, it is difficult to make any statements about the global stability of the elastic when the force is applied.

Later in the chapter, we will return to the issue of decoupling the internal force from the amount of stretch in the elastic band. We now discuss another problem that arises when we use our new internal force.

3.8 A Problem with the New Internal Force

The internal force described in the last section is based on a model in which the elastic band is constructed from a material that has constant internal tension. Although this model is less "physical" than the original model based on an idealized elastic material, it does have advantages as a model of a deforming path. There is, however, a major problem with the force resulting from this model—it contains no component tangential to the path.

The lack of a tangential component of the internal force means that, in general, there is no feasible steady state configuration for the elastic. Any component of the external force that acts along the elastic will not be opposed by an internal force. The external force can continuously push particles of the elastic band along the path, resulting in an elastic that becomes progressively *thinner* in some regions.

To see that $\mathbf{f}_{int}$ contains no component along the elastic, consider the dot product of $\mathbf{f}_{int}$ and $\mathbf{c}'$.

$$\mathbf{f}_{int} \cdot \mathbf{c}' = \frac{k}{\|\mathbf{c}'\|} \left(\mathbf{c}'' \cdot \mathbf{c}' - \frac{\mathbf{c}' \cdot \mathbf{c}''}{\|\mathbf{c}'\|^2} \mathbf{c}' \cdot \mathbf{c}' \right) = 0.$$

The fact that the elastic may not have a static configuration may seem surprising. After all, the forces on the elastic are derived from non-negative energy functions and it might be expected that gradient descent of the sum of the energy functions should lead to a minimum. Such a property holds for energy functions that are defined over a finite set of variables such as in the potential field method. For energy functionals, functions of functions, it is possible to move down the gradient without converging to a minimum. In fact, no minimum may exist!

For our model of an elastic band, the internal energy remains constant as particles are moved along the path. Consider the expression for the total internal energy,

$$\begin{aligned} V_{int}[\mathbf{c}] &= \int_0^1 v_{int}\, ds \\ &= k \int_0^1 \|\mathbf{c}'\|\, ds. \end{aligned}$$

This expression can be seen to be k times the length of the elastic, which is obviously invariant under a re-parameterization of the path.

By moving particles along the elastic, the total energy of the elastic can be reduced. As the internal energy will remain constant, such a reduction can be achieved by simply moving particles along the path from configuration in which the external energy v_{ext} is high to regions in which it is low. This process can continue indefinitely, progressively making the elastic thinner and thiner in the higher energy regions.

It is worth noting that such a problem does not occur with the original internal energy density specified by (3.4). If the particles are moved along the path, the internal energy will change. As the elastic becomes thinner in some regions, the local tension and hence the internal energy increases.

3.9 Adding a Constraint Force

One solution to the above problem is to constrain the motion of particles of the elastic. Consider adding a constraint force that eliminates the motion of particles along the elastic. For a given particle, this constraint force will be equal and opposite to the component of the external force along the elastic band. As the elastic band deforms, the direction of the constraint will vary accordingly. More formally, we can add a constraint force $\mathbf{f}_{const}$, given by the equation,

$$\mathbf{f}_{const} = -\frac{\mathbf{f}_{ext} \cdot \mathbf{c}'}{\|\mathbf{c}'\|^2}\, \mathbf{c}'. \tag{3.12}$$

The constraint force given by (3.12) can be seen to be non-holonomic; it is similar to the constraint force that stops a car from sliding sideways. As $\mathbf{f}_{const}$ is non-holonomic, it cannot be expressed as the gradient of an energy function. However, at any given instant, the force $\mathbf{f}_{const}$ for a given particle is along a direction in which the particle does not move; this is the *raison d'être* of the force. Thus the constraint force does no work and can be ignored when considering the energy of an elastic.

We conjecture that the addition of the force $\mathbf{f}_{const}$ will ensure that a static configuration for the elastic always exists. This statement has not been proved, but is supported by our experience with implementations based on this approach.

3.10 Changing the length of the elastic

We now return to the issue of decoupling the internal force from the stretch in the elastic band. First, note that the addition of the constraint force partially solves the problem. As particles in the elastic can no longer move along the elastic, local changes in the stretch will not propagate to other regions and thus the elastic will be reasonably decoupled. This property does not mean that local changes will have no global effect—the particles can still move perpendicular to the elastic causing a change in the local shape, which can have a global effect.

A problem still remains. If a local region of the elastic is deformed, the local stretch will change, resulting in a variation of the amount of external force needed to modify the band. This variation will be a function of both the shape of the elastic and the stretch, while we would prefer it to be a function only of the shape.

To overcome this problem, our current implementation uses a rather ad hoc solution that works well in practice. We postpone a complete description of the solution to Chapter 5, but the basic idea is to vary the length of the elastic band.

Suppose some local region of an elastic band is being deformed in such a way that the amount of stretch is increasing. At some point, we decide to add additional "material" to this section of the elastic and thus decease the amount of stretch. Vice versa, if the amount of stretch decreases to a certain point, some of that section of the elastic is removed. In this way, a reasonably consistent amount of stretch can be maintained along the elastic.

Adding such an ad hoc meta-behavior to the elastic band raises many complex issues. Even though the underlying force-based behavior of the elastic can be shown to be well behaved, it is unclear whether the addition and removal of parts of the elastic may interact with the forces in some undesirable fashion. Although we have little theoretical justification for this approach, we have found in practice that it provides the desired behavior.

Intuitively, it is easy to see that certain problems will not occur. As particles cannot move along the elastic, we do not expect a situation in which elastic material is created in one region, propagates to another region, and is then removed.

A problem that did occur was that the addition of material caused the shape to change slightly such that material is then removed. This problem can be avoided by adding a hysteresis-like property to the process.

3.11 A Summary of the Model

In summary, we have developed the following model for elastic bands.

An elastic band is a one-dimensional continuous solid described by a parameterized function $\mathbf{c}(s)$ where $s \in [0, 1]$. The force applied to each particle s of the elastic is given by the function

$$\mathbf{f}(s) = \mathbf{f}_{int}(s) + \mathbf{f}_{ext}(s) + \mathbf{f}_{const}(s),$$

where $\mathbf{f}_{int}$, $\mathbf{f}_{ext}$, and $\mathbf{f}_{const}$ are the internal, external and constraint force functions. These functions are defined by the following equations

$$\begin{aligned} \mathbf{f}_{int}(s) &= k \, \|\mathbf{c}'\| \, \boldsymbol{\kappa}(s), \\ \mathbf{f}_{ext}(s) &= -\nabla v_{ext}(\mathbf{c}), \\ \mathbf{f}_{const}(s) &= -\frac{\mathbf{f}_{ext} \cdot \mathbf{c}'}{\|\mathbf{c}'\|^2} \mathbf{c}', \end{aligned}$$

where k is the constant of elasticity, $\boldsymbol{\kappa}$(s) is the curvature vector of the path at s, and v_{ext} is a user specified potential function over the configuration space of the robot.

The internal and external forces are defined in terms potential functionals of the shape of the elastic $\mathbf{c}$ respectively given by

$$\begin{aligned} V_{int}[\mathbf{c}] &= \int_0^1 k\|\mathbf{c}'\| \, ds, \\ V_{ext}[\mathbf{c}] &= \int_0^1 v_{ext} \, ds. \end{aligned}$$

The force $\mathbf{f}_{const}$ imposes the constraint that particles do not move along the elastic band. The constraint force is not derived from a potential functional, but by its nature, does no work and thus does not influence the energy of the elastic.

The resultant force on each particle causes a pseudo-static motion given by the relation

$$\frac{\partial \mathbf{c}}{\partial t} \propto \mathbf{f}.$$

Finally, to overcome the coupling between the amount of stretch in the elastic band and the magnitude of the internal force, we apply a procedure, described in Chapter 5, that adds and removes material from the elastic.

The processes of developing the elastic band model was iterative and essentially followed the description in this chapter. The original idea was to bring collision-free paths to "life" by

treating them as deformable physical objects. Our first implementation of this idea closely modeled the true physical behavior of such objects, but as problems arose, we modified our model in more and more non-physical ways. None the less, we still tried to maintain the spirit of modeling a physical object as we feel this provides a powerful metaphor for the design of computer algorithms.

3.12 Modifying a Path using Optimization Theory

To round out this chapter, we next briefly describe an approach to modifying a path based on optimization theory. In some respects, such an approach has a more solid mathematical foundation than the elastic band model described above. It is interesting that the resulting model is similar, but not identical, to the elastic band model.

The classical form for an optimization problem [25] may be expressed as:

$$\begin{array}{lll} \text{minimize} & F(x) & x \in \mathbf{R}^n \\ \text{subject to} & f_i(x) = 0, & i = 1, 2, \ldots, m; \\ & f_i'(x) \geq 0, & i = 1, 2, \ldots, m'. \end{array}$$

The function F is called the objective function, the functions f_i are equality constraints, and the functions f_i' are inequality constraints. Most optimization algorithms do not find the global minimum of F but find some local minimum.

Optimizing a path cannot be directly expressed in the above form as an arbitrary path cannot be described by a point in some finite-dimensional space $\mathbf{R}^n$. In practice, some finite-dimensional representation would be used that can describe a large subset of the possible paths. In this section, we will not consider the particular approximations that could be used to implement the optimization of a path; rather, we consider optimizing the path directly. In particular, the optimization problem we are interested in solving can be stated as:

$$\begin{array}{lll} \text{minimize} & F[\mathbf{c}] & \mathbf{c} : [0,1] \rightarrow C \\ \text{subject to} & \mathbf{c}(s) \in C_{free}, & s \in [0,1] \\ & \mathbf{c}(0) = \mathbf{q}_{start},\ \mathbf{c}(1) = \mathbf{q}_{goal}, & \\ & \mathbf{c} \text{ is a smooth function.} & \end{array}$$

Note that F is a function of the function $\mathbf{c}$, i.e., a functional.

Given that a path is a geometric object, it seems natural that optimizing a path refers to optimizing its shape. Reparameterizing a path does not change its shape and hence we require that the objective functional F be independent of the particular parameterization of the path. Mathematically, given a monotonic function $g : [0,1] \rightarrow [0,1]$ with $g(0) = 0$ and $g(1) = 1$, we can define a reparameterization of $\mathbf{c}$ as the path $\mathbf{c}^*$ where

$$\mathbf{c}^*(s) = \mathbf{c}(g(s)).$$

For all such reparameterizations of $\mathbf{c}$, the functional F should have the property that

$$F[\mathbf{c}^*] = F[\mathbf{c}].$$

For example, consider defining F as the length of a path, i.e.,

$$F[\mathbf{c}] = \int_0^1 \|\frac{d\mathbf{c}(s)}{ds}\| \, ds.$$

For a reparameterization $\mathbf{c}^*(s) = \mathbf{c}(g(s))$, the functional F can be written

$$\begin{aligned} F[\mathbf{c}^*] &= \int_0^1 \|\frac{d\mathbf{c}^*(s)}{ds}\| \, ds \\ &= \int_0^1 \|\frac{d\mathbf{c}(g(s))}{ds}\| \, ds \\ &= \int_0^1 \|\frac{d\mathbf{c}(g)}{dg}\| \, \|\frac{dg}{ds}\| \, ds. \end{aligned}$$

As g is monotonic increasing

$$dg/ds \geq 0,$$

hence

$$\begin{aligned} F[\mathbf{c}^*] &= \int_0^1 \|\frac{d\mathbf{c}(g)}{dg}\| \, \frac{dg}{ds} \, ds. \\ &= \int_0^1 \|\frac{d\mathbf{c}(g)}{dg}\| \, dg \\ &= F[\mathbf{c}]. \end{aligned}$$

What objective functional should we use to implement the path modification described in Chapter 1? Many possibilities exist, but one that seems intuitive is an extension of the notion of path length mentioned above. As the path is constrained to be collision-free, the minimum length path will consist of straight line segments separated by segments that lie on the boundary of the free space. To improve the shape of the minimum length path, suppose a positive cost is associated with each configuration in the free space and the objective function F is defined as a path length that is weighted by this cost. More formally, given a scalar function $v : C_{free} \rightarrow \mathbf{R}$, the objective functional $F[\mathbf{c}]$ is defined as

$$\begin{aligned} F[\mathbf{c}] &= \int_0^1 v(\mathbf{c}(s)) \|\frac{d\mathbf{c}(s)}{ds}\| \, ds, \\ &= \int_0^1 v(\mathbf{c}) \|\mathbf{c}'\| \, ds. \end{aligned} \tag{3.13}$$

The idea behind the objective functional (3.13) is that the cost function v can be used in a manner similar to the potential function v_{ext} in the elastic band model. For example,

increasing the cost near the boundary of C_{free} will move the minimum cost weighted path away from the obstacles.

Using the functional gradient operator $\overline{\nabla}$ described earlier in this chapter, we can examine the properties of paths that minimize the objective functional given by (3.13). Suppose the path $\mathbf{c}$ is well inside the free space; we can ignore the constraint that $\mathbf{c}$ be collision-free. As shown in Appendix A, a necessary condition for a minimum of a functional of the form

$$F = \int_0^1 f(s)\, ds,$$

is

$$\overline{\nabla} f(s) = 0,\ \forall s \in [0,1].$$

For the functional defined by (3.13)

$$\begin{aligned}
\overline{\nabla} f &= \left(\frac{\partial f}{\partial \mathbf{c}} - \frac{d}{ds}\frac{\partial f}{\partial \mathbf{c}'}\right)[v(\mathbf{c})\|\mathbf{c}'\|] \\
&= \|\mathbf{c}'\|\frac{\partial v}{\partial \mathbf{c}} - \frac{d}{ds}v(\mathbf{c})\frac{\mathbf{c}'}{\|\mathbf{c}'\|}, \\
&= \|\mathbf{c}'\|\frac{\partial v}{\partial \mathbf{c}} - \frac{v}{\|\mathbf{c}'\|}\left(\mathbf{c}'' - \frac{\mathbf{c}''\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}'\right) - \frac{\partial v}{\partial \mathbf{c}}\cdot\mathbf{c}'\frac{\mathbf{c}'}{\|\mathbf{c}'\|}, \\
&= \|\mathbf{c}'\|(\nabla v - v\boldsymbol{\kappa} - \frac{\nabla v\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}'),
\end{aligned}$$

where $\boldsymbol{\kappa}$ is the curvature vector of $\mathbf{c}$ (see Appendix B). For $\overline{\nabla} f = 0$ we have

$$v\boldsymbol{\kappa} - \nabla v + \frac{\nabla v\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\mathbf{c}' = 0. \tag{3.14}$$

What is interesting about equation (3.14) is its similarity with the equilibrium condition of the elastic band model. A particle of an elastic band is in equilibrium when the resultant force acting on the particle is zero. Using the model described in the last section, this condition occurs when

$$\mathbf{f}_{int} + \mathbf{f}_{ext} + \mathbf{f}_{const} = k\ \|\mathbf{c}'\|\ \boldsymbol{\kappa} - \nabla v_{ext} + \frac{\nabla v_{ext}\cdot\mathbf{c}'}{\|\mathbf{c}'\|^2}\,\mathbf{c}' = 0. \tag{3.15}$$

By inspection, it can be seen that the only difference between (3.14) and (3.15) is the scalar multiple of the curvature vector $\boldsymbol{\kappa}$. For the elastic band model, $\boldsymbol{\kappa}$ is scaled by a user specified constant k and the local stretch in the elastic $\|\mathbf{c}'\|$. The optimization approach scales $\boldsymbol{\kappa}$ by the value of the cost function v at $\mathbf{c}(s)$.

It is unclear, at least to this author, what the effect of the difference between the elastic band model and the optimization based approach will have on on the modification of collision-free paths. We have not implemented the optimization based approach, but we suspect the difference will be minimal.

Chapter 4

Bubbles of Free Space

4.1 Introduction

In this chapter we describe the *bubble* concept. A bubble represents a local subset of the free-space around a given configuration of the robot. Bubbles are generated efficiently from distance information obtained by examining a model of the robot and the objects in the environment. Bubbles enable applications to utilize information about the free-space in complex and changing environments without resorting to computationally infeasible models of the free-space in its entirety. In Chapter 5, the bubble concept is used to implement the elastic band model efficiently.

The motivation for the bubble concept is the difficulty of generating or representing the entire free-space for a robot. As seen in Chapter 2, it is possible to build models of the free-space, however, these methods grow exponentially in both time and space with the dimension of the configuration space. For configuration spaces with dimension higher than three or four, the cost associated with building and storing a model of the free-space is prohibitive. For example, a uniform discrete representation of a six-dimensional configuration space with one hundred divisions of each axis requires 10^{12} bits, which is approximately 100 gigabytes of memory. Assuming the time to determine if a single configuration is in the free-space is one nanosecond, it will still require 15 minutes to generate the entire free-space.

Even for low-dimensional configuration spaces, the cost of generating a representation of C_{free} may be large enough that it is difficult to maintain, in real-time, for a changing environment.

Instead of representing the entire free-space, we propose to compute, on the fly, local subsets of C_{free} around specified configurations of the robot. These local subsets are called *bubbles*. We use the notation $\beta_{\mathbf{q}}$ to indicate the bubble around the configuration $\mathbf{q}$.

The bubble at configuration $\mathbf{q}$ is generated from information about the distance between the robot when in configuration $\mathbf{q}$ and the current state of the obstacles in the environment.

If the configuration of the robot is in C_{free}, the distance from the obstacles should be greater than zero. In addition, the magnitude of the distance enables us to infer that certain neighboring position in C_{space} are also in C_{free}. A bubble is a set of neighboring configurations that are inferred to be in C_{free}.

Consider the two-link revolute manipulator shown in Figure 4.1a. At the given configuration, the minimum distance between the robot and the environment is shown by the line segment $\mathbf{AB}$. The diamond-shaped region in Figure 4.1b represents a bubble of free-space generated from the length of $\mathbf{AB}$. For reference, the shaded region in Figure 4.1b represents the configuration space obstacles in joint space. These obstacles were generated by sampling ten thousand different configurations of the manipulator. Note, the bubble of free-space was constructed solely on the distance information and did not use the configuration space obstacles shown in the figure.

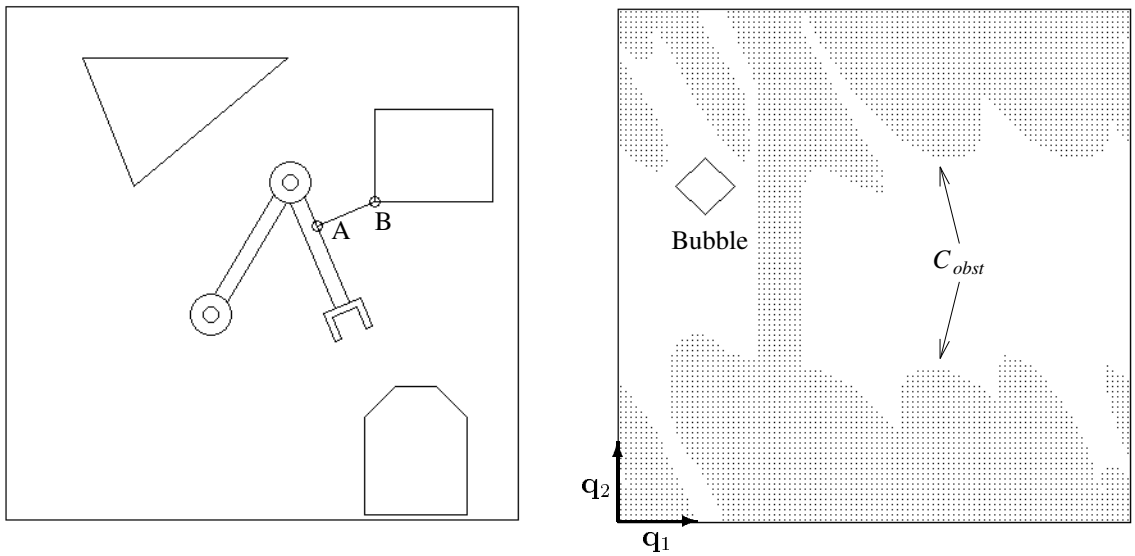


Figure 4.1: **a)** A manipulator with two degrees of freedom in an environment with obstacles. The line segment $\mathbf{AB}$ represents the minimum distance between the robot and the obstacles. **b)** The joint configurations space for the robot. The shaded area represents the configuration space obstacles generated by sampling. The diamond represents the bubble generated from the length of $\mathbf{AB}$.

A bubble represents a local collision-free region around a configuration of the robot. An application can generate bubbles around various configurations of interest and hence determine useful information about the free-space. The advantage of the bubble concept is that information is generated only about areas of interest rather than trying to model the entire free-space. By examining only a small area of the free-space, real-time performance

can be achieved. For example, to deform a collision-free path using the elastic band model, we are interested in the free-space around the current path. In a high-dimensional configuration space, the volume of the free-space around a path is a miniscule fraction of the total volume of the free-space.

For a specific robot, the shape of a bubble of free-space is restricted to a simple geometric object such as a hypersphere. This restriction enables a bubble to be described by a small number of parameters and simplifies tasks such as determining whether two bubbles intersect. The restriction on the shape of a bubble may limit our ability to directly describe the extent of the free-space around the robot, however, this does not pose a problem; if an application requires more information about the free-space, additional bubbles can be created around other configurations of the robot.

It is also useful to require that bubbles are convex. This property facilitates the construction of collision-free paths within the bubble. The straight line motion between two configurations in a convex bubble will be collision free. In addition, many representations of curves have the property that they remain within the convex hull of their defining control points. For example, Figure 4.2 illustrates a Bézier curve defined by four control points. The dotted line represents the convex hull of these control points and it can be seen that the curve remains within this region. If all the control points are within a bubble, the entire curve is collision free.

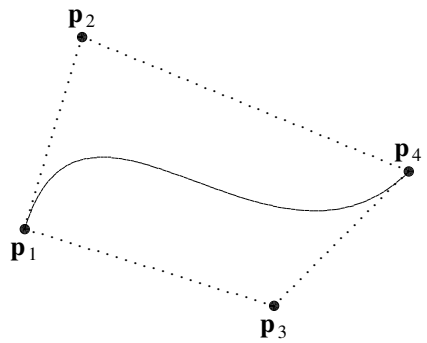


Figure 4.2: A Bézier curve defined by four control points. The dotted line is the convex hull of these four points.

4.2 Bubbles and Cell Decompositions

In many ways, bubbles of free-space are similar to the cells in a decomposition of C_{free} (see Chapter 2). Like a bubble, a cell is a region of C_{free} represented by some simple shape with properties such as convexity.

In contrast to bubbles, with a cell decomposition, the geometric shape and placement of cells is designed to facilitate complete coverage of the entire free-space. For some

applications, notably path planners, the complete coverage of the free-space is an important criterion. If this global property is not needed, however, the somewhat arbitrary nature of the cell boundaries can be rather inconvenient. Consider using a cell decomposition to determine the free-space around a collision-free path. The set of cells that covers the path may contain boundaries such that the free-space around the path is marginal. These boundaries do not indicate that the path comes close to collision; rather, they are artifacts of the precomputed cell decomposition.

An analogy can be drawn between a cell decomposition and an atlas of maps. When we are interested in some area of the world, we can turn to the appropriate page in the atlas for information. Since the division of the world into pages has been determined to facilitate the complete coverage of the various land masses, the location of interest may be inconveniently near the edge of a page.

The equivalent analogy for the bubble approach would be a system that generates a custom map with the location of interest at the center. Such custom maps would avoid the problem of being near the artificial boundary of the page.

Bubbles and cell decomposition methods also differ on the basic computation that is performed. In the bubble approach, a region of free-space is calculated from distance information. The most expensive part of the bubble approach is the generation of distance information. For typical cell decomposition algorithms, the fundamental procedure is determining whether a given cell is contained entirely, partially, or not at all within the free-space. This operation requires the intersection of the cell and the boundary of the free-space and can be quite complex [38].

4.3 Applications of Bubbles

We feel the bubble concept has many potential applications in various areas of robotics. A bubble represents local information about C_{free} and this information can be obtained efficiently, even for robots with high-dimensional configuration spaces or in environments that are changing. We developed the bubble concept to implement the modification of a collision-free path; however, it is not difficult to envision using bubbles in other applications such as path planning, off-line programming, or robot control. In this section, some of these other applications of bubbles are briefly outlined.

Path planning

As mentioned in Chapter 2, one of the trends in path planning for robots with many degrees of freedom is to abandon systematic search of C_{free} in favor of randomized search. One common step in a randomized search is to move the robot from one collision-free configuration to another neighboring configuration along a simple path such as a straight

line. How do we determine that such a motion does not move the robot through an obstacle? A common solution is to restrict the distance between the two configurations to be relatively small. Since the two configurations are in C_{free}, it is probable that the motion between the configurations will not cause a collision. However, setting the step size is an awkward problem. A small step size will reduce the likelihood of moving through an object, but setting the step size too small greatly decreases the efficiency of the search.

By using bubbles, the step-size for the search routine can be determined adaptively and the motion between successive configurations will be collision free. For a configuration of the robot, the associated bubble represents a local region of C_{free}. To step in a desired direction, the robot is moved up to the edge of this bubble with full confidence that no collision has occurred. When the robot is far from the obstacles in the environment, we expect the bubbles of free-space to be relatively large and hence the robot can take large steps; conversely, in a tight situations, small steps will be taken.

The cost of generating bubbles is dominated by the cost of obtaining distance information. Chapter 6 presents an algorithm that can obtain this distance information with only marginally more effort than determining solely whether a configuration is collision free. With such an algorithm, the use of bubbles in randomized path planners seems truly advantageous.

Path smoothing

Under the reasonable assumptions that a path $\mathbf{c}$ is compact, C_{free} is open, and the bubble at any configuration in C_{free} is a non-empty open set with a minimum size that is greater than some constant factor of the distance between the robot and the obstacles, it can be shown that $\mathbf{c}$ can be covered with a finite number of bubbles. Informally, this may be seen by the following procedure. Determine the bubble at the start of the path, i.e., find $\beta_{\mathbf{c}(0)}$. Move along the path until the first value of s is found such that

$$\mathbf{c}(s) \notin \beta_{\mathbf{c}(0)}.$$

Since bubbles are open, either such a value of s will exist or the bubble contains the entire path.

Next, generate another bubble at $\mathbf{c}(s)$ and repeat the above process until the end of the path is covered as illustrated in Figure 4.3. Since the path is collision free and C_{free} is open, there exists some minimum clearance between the path and the obstacles. From this property and the restriction on the minimum size of a bubble, it can be seen that there is a minimum sized bubble along the path. Thus, each iteration of the above step will move at least some constant distance along the path.

At the end of this process we will have a series of values, $s_0, s_1, \ldots, s_m$ such that

$$\beta_{\mathbf{c}(s_i)} \cap \beta_{\mathbf{c}(s_{i+1})} \neq \emptyset$$

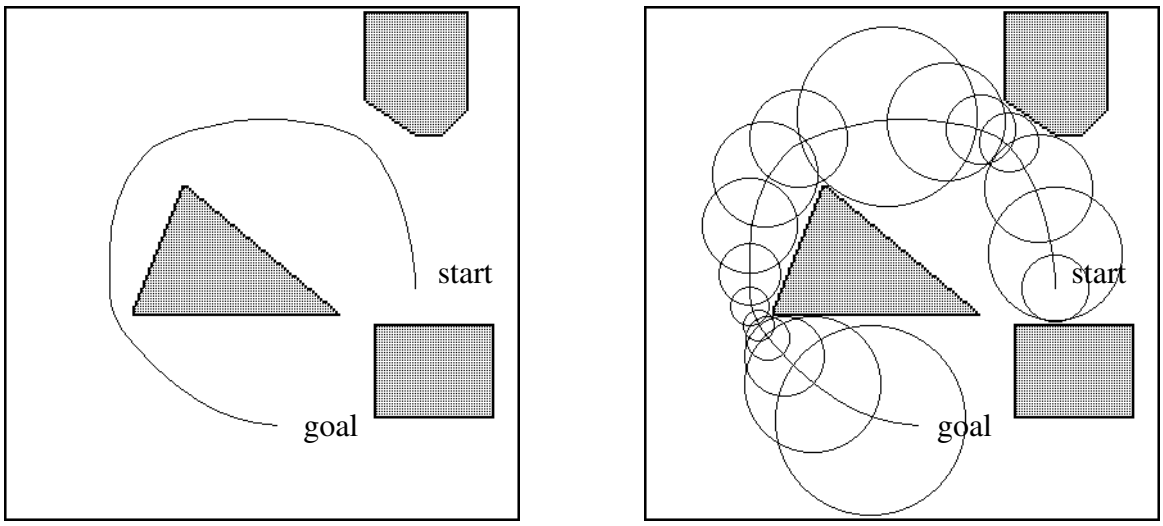


Figure 4.3: **a)** A path in a two-dimensional configuration space. **b)** The path covered with bubbles. Note that the bubbles are generated without computing the configuration space obstacles and hence can be done in high-dimensional configuration spaces.

for all $i < m$. Also, for each $s \in [0, 1]$ there exists some i such that

$$\mathbf{c}(s) \in \beta_{\mathbf{c}(s_i)}. \tag{4.1}$$

One may question the practicality of finding the first value of s such that $\mathbf{c}(s) \notin \beta_{\mathbf{c}(s_i)}$. For many discrete representations of $\mathbf{c}$, this value can be determined efficiently. The details are tedious and depend on the particular representation that is used, but the general idea is as follows. Using a standard root finding procedure, a value of s can be found such that (4.1) holds for s and not for $s - \epsilon$. Next, it is determined if the path between s_i and s is contained in the bubble. For representations such as B-splines, Bézier curves, etc, a set of points for any segment of the curve can be found such that the segment lies within the convex hull of these points. Furthermore, as the length of the segment is reduced, the set of points converges towards the curve. Using this property, we can construct an algorithm that determines if a path segment lies in a bubble by examining the corresponding control points to see if they lie in the bubble. If the control points are not contained in the bubble, the segment is divided into two parts and the procedure is recursively called for each part. This process will continue until it is determined that the segment is in the bubble or a new point is found that is not contained in the bubble.

After covering the path with bubbles, we have a representation for a connected region of C_{free} that includes both the start and the goal position. One can view this region as describing an infinite bundle of homotopic paths from the start to the goal. Any one of these possible paths can be selected as a replacement for the original. One practical application

of this capability is the conversion of continuous piecewise differential paths to a C^1 or C^2 path. In particular, a path from a planner that is constructed of line segments can be converted to a path suitable for a control system.

Path collision checking

A common problem in off-line programming systems is determining whether a user-specified path is collision free. The typical approach to this problem is to check for collision at incremental steps along the path. An issue that arises with this approach is the size of the steps. Smaller steps increase the confidence that the path is collision free, but also increase the time required to check the path.

By a simple modification of the algorithm for covering a collision-free path with bubbles, bubbles can be used to determine efficiently and rigorously if a path is collision free. Consider attempting to cover a given path with bubbles. If this process succeeds, the path is obviously collision free. By setting a minimum size for a bubble, it can be seen that attempting to cover a path that is not collision free will fail in a finite number of steps.

Tracking tolerances

By covering a path with bubbles, useful information can be obtained about the tolerance with which the control system must track the path if collisions are to be avoided.

Given the nature of control systems, when moving a robot along a path there is invariably some degree of tracking error. This error is a result of unmodeled disturbance forces that the robot experiences during motion and is often correlated to the speed with which the robot moves. Since the size of a bubble is related to the distance from the obstacles, it seems reasonable to limit the speed of the robot at a given point on the path based on the size of the corresponding bubble.

4.4 Computing Bubbles

In the remainder of this chapter we examine various specific types of robots and describe viable schemes for computing bubbles of free-space.

For a particular robot there are many possible approaches that can be taken to deriving some local region of the configuration space that is contained in C_{free}. These different approaches often represent tradeoffs between the size of the bubble and the computation cost for its generation. For our application, implementing elastic bands, the effect of having smaller bubbles is to increase the number of bubbles required to represent a collision free path; the tradeoff is between the cost per bubble and the number of bubbles needed. It is unclear what the net effect on the total computation cost will be. In the following

examples we have tended towards finding simple bubbles as these are easier to describe and implement. Overall, we feel that such an approach provides good performance although there is certainly opportunity for experimentation with different balances of computational cost and size.

Computing the distance information needed to generate a bubble requires a model of the robot and the current state of the environment. Such models are at most three-dimensional, and compared to models of the free-space, are simple to generate and maintain. Although there are many interesting issues involved in generating and maintaining models of the robot and environment, we assume below that these models are available.

In general we expect that a robot operates in a changing environment. In fact, one of the main goals of elastic bands is to implement reactive behaviors which implies that some information about the environment in not available *a priori*. One issue that is avoided in this thesis relates to situations in which the changes in the environment can be predicted to some degree, for example, an object that is moving at a constant velocity. For such situations it would be feasible to specify the bubble at a given configuration and a given time. This extension is quite difficult when applied to implement elastic bands and throughout the thesis we consider only the current state of the environment. Such an assumption appears reasonable when the environment changes relatively slowly or unpredictably.

The general strategy for computing a bubble at a configuration $\mathbf{q}$ is as follows. First, for each point on the robot a lower bound on its distance from the obstacles is determined. This distance information may be generated by computing the minimum distance between the robot and the obstacles resulting in the same lower bound for each point on the robot. Alternatively, the lower bound for various components of the robot may be increased by separately computing the minimum distance between these components and the obstacles. Next configurations of the robot are found such that the motion from the initial configuration to these configurations moves each point on the robot a distance less than the corresponding lower bound to the obstacles. Such configurations are in the free-space.

Underestimating the free-space around a configuration can be beneficial as it is often possible to calculate more efficiently a conservative bubble from distance information. For a specific robot with an n-dimensional configuration space, we select some simple n-dimensional bubble shape defined by a small number of parameters. The possible instances of the shape form the basis from which bubbles will be determined. At a given configuration, the system computes information about the distance between the robot and the environment. From this information a set of values of the parameters are determined such that the corresponding instance of the shape is contained in C_{free}; this is the bubble at the configuration.

For the following examples it is assumed that there exists an algorithm for determining the minimum distance between two objects in either two or three dimensions. Given two objects A and B, the minimum distance between the sets is defined by the function

$$d(A, B) = \inf\{\|\mathbf{x} - \mathbf{y}\| : \mathbf{x} \in A, \mathbf{y} \in b\}$$

An efficient algorithm for computing $d(A, B)$ is given in Chapter 6.

Another useful capability is the ability to determine the maximum distance between a point and a set of objects. More formally, given the object A and a point $\mathbf{x}$, we wish to find the minimum value for r such that the hypersphere that is centered at $\mathbf{x}$ with radius r completely contains A. The distance r will be used to determine an upper bound on the motion of all points in A resulting from a rotation about an axis that passes through the point $\mathbf{x}$.

4.5 A Non-Rotating Free-Flying Robot in the Plane

The first robot we will examine is a non-rotating free-flying robot in a planar environment. For such a robot, the natural generalized coordinates are the Cartesian coordinates (x, y) of some fixed point on the robot, which describes the position with respect to some fixed reference frame.

Consider moving the robot from configuration $\mathbf{q}$ to configuration $\mathbf{p}$ along a straight line. Given that the robot is rigid and does not rotate, we know that every point on the robot moves a distance $\|\mathbf{q} - \mathbf{p}\|$. Now suppose the minimum distance d is calculated between the robot at configuration $\mathbf{q}$ and the obstacles in the environment. If the robot is in the free-space then $d > 0$. Also, if the robot moves a distance less than d, no collision can occur.

The above property suggests we define the bubble at $\mathbf{q}$ as the open circle of radius d centered at configuration $\mathbf{q}$. More formally,

$$\beta_{\mathbf{q}} = \{\mathbf{p} : \|\mathbf{q} - \mathbf{p}\| < d\}. \tag{4.2}$$

Figure 4.4 depicts an example bubble for such a robot.

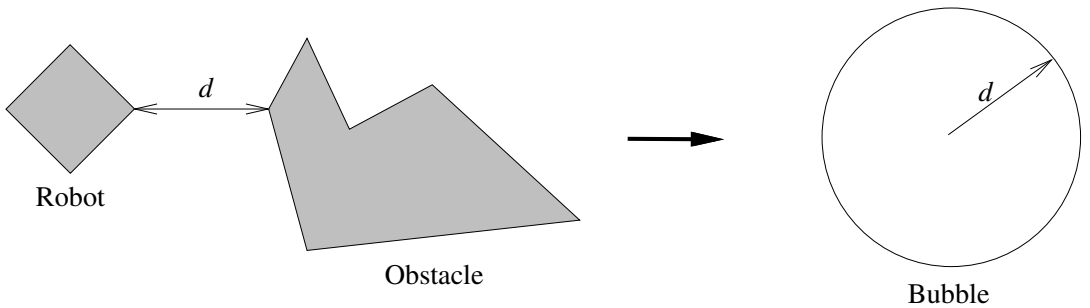


Figure 4.4: A bubble for a non-rotating robot that is a distance d from the obstacles.

The bubble defined by (4.2) follows rather naturally from the given distance information d. In some sense, it fully utilizes the distance information as any motion larger than d

may cause collision. We now consider an alternative bubble definition for this robot. This alternative offers few advantages over (4.2) and is not particularly useful, but we include it as an example of the different possible bubbles that can be computed for a given robot.

Consider again the motion of the robot from $\mathbf{q}$ to $\mathbf{p}$. The shortest path for this robot is the line segment between $\mathbf{q}$ and $\mathbf{p}$. An alternative path is to successively move along the x and then y axes. The distance traveled by every point on the robot along such a path equals the Manhattan distance (1-norm metric) between $\mathbf{q}$ and $\mathbf{p}$, i.e.,

$$|\mathbf{q} - \mathbf{p}| = |q_x - p_x| + |q_y - p_y|.$$

The Manhattan distance is always greater or equal to the usual Euclidean distance and for this particular type of robot it seems unattractive. For more complex robots, the utility of decomposing a motion into separate steps along subspaces of the configuration space will become apparent.

Given that the Manhattan distance bounds the distance traveled by any point on the robot when moving from $\mathbf{q}$ to $\mathbf{p}$, we can define a bubble as follows:

$$\beta^*_{\mathbf{q}} = \{\mathbf{p} : |\mathbf{q} - \mathbf{p}| < d\}.$$

Such bubbles will have the shape of a diamond as shown in Figure 4.5. For a given configuration, the bubble $\beta^*_{\mathbf{q}}$ will be a proper subset of $\beta_{\mathbf{q}}$ and hence the bubble defined by (4.2) is preferable.

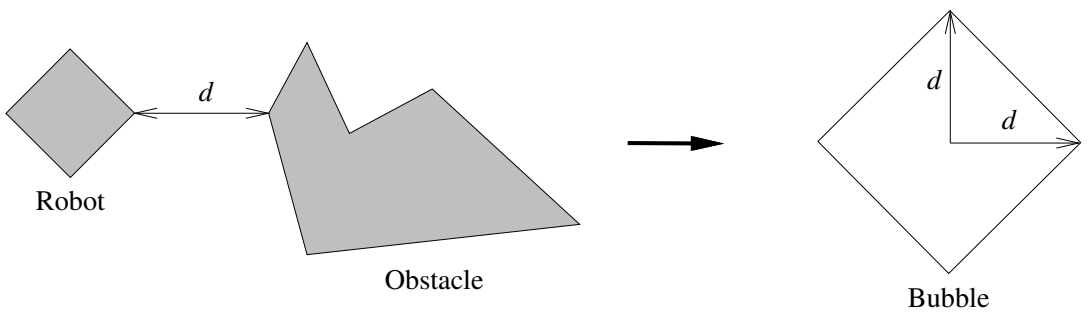


Figure 4.5: An alternative bubble for a non-rotating robot that is a distance d from the obstacles.

Another, perhaps more useful, alternative to defining bubbles for this robot is to use more information than provided by the minimum distance between the robot and the obstacles. One common suggestion is to somehow determine the minimum distance in various directions. These multiple distances could enable us to derive bubbles that are larger than those defined above. We will not explore this possibility, but it illustrates the following general property of bubbles: *The region of the configuration space that can be*

deduced as lying in C_{free} depends on the form of distance information that is provided. This property opens up many schemes for computing bubbles, but in this thesis, we limit ourselves to rather simple approaches.

4.6 Planar Robots with Rotation

Let us now consider the case in which the planar robot can rotate. In this situation, the natural parameterization of the configuration space is the Cartesian position and angle of rotation of a fixed frame on the robot with respect to some global frame. A configuration is represented by the standard three coordinates x, y, and θ.

Let d be the minimum distance between the robot at a configuration $\mathbf{q}$ and the obstacles. Suppose, in addition, a constant r_{max} has been determined that represents the maximum distance from the origin of the robot to any other point on the robot.

The general scheme to define a bubble for this robot will be the same as the previous case. We will consider the motion of the robot from one configuration to another and derive a bound for the distance traveled by any point on the robot. The bubble will consist of configurations which the distance traveled is less than d.

Consider moving the robot from configuration $\mathbf{q}$ to configuration $\mathbf{p}$. The straight line motion between $\mathbf{q}$ and $\mathbf{p}$ will, in general, cause the robot to rotate and translate at the same time. For such motion, the path traveled by any particular point on the robot is quite complex. To avoid this problem, let us examine the simpler, albeit longer, motion consisting of a translation followed by a rotation. During the translation part of the motion, each point on the robot moves a distance equal to the Euclidean distance between the points (q_x, q_y) and (p_x, p_y), i.e.,

$$d_{trans}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}.$$

For the rotation part of the motion, the distance traveled by each point on the robot varies depending on its distance from the origin of the robot. The largest distance traveled will be for the point farthest from the origin. Given the constant r_{max} and using some simple geometry, the maximum distance can be shown to be

$$d_{rot}(\mathbf{q}, \mathbf{p}) = 2r_{max} \sin \frac{1}{2}|q_\theta - p_\theta|. \tag{4.3}$$

The sum of the translational and rotational distances is an upper bound on the distance traveled by any point on the robot. From the this bound, we can define the bubble $\beta_{\mathbf{q}}$ as,

$$\beta_{\mathbf{q}} = \{\mathbf{p} : d_{trans} + d_{rot} < d\}. \tag{4.4}$$

Recall that for this robot, the configuration space is three-dimensional. The bubble specified by (4.4) consists of a region of this space as illustrated in Figure 4.6.

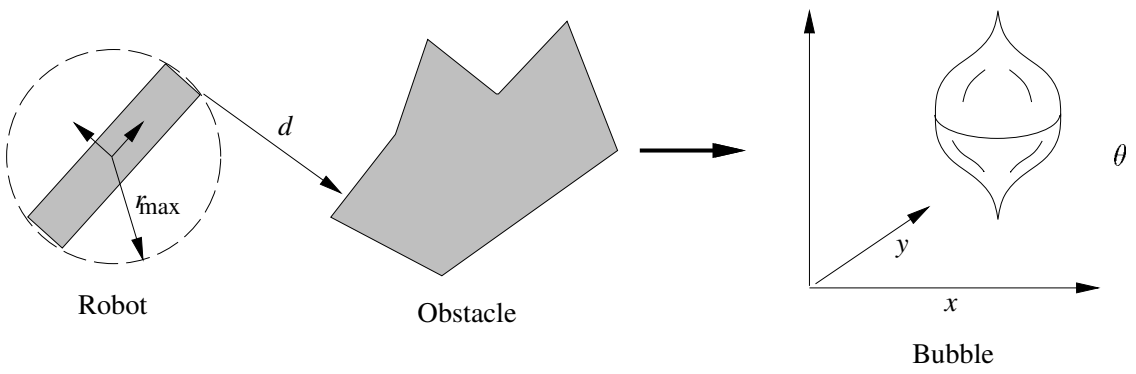


Figure 4.6: A non-convex bubble for a rotating robot.

As can be seen in Figure 4.6, the bubble defined by (4.4) is non-convex. Non-convex bubbles are undesirable as it is difficult to design smooth paths that remain within the bubble. Simple inspection shows that the reason the bubble is non-convex is a result of the sine function in the definition of d_{rot} in equation (4.3). One simple solution to this problem is the use a different bound for d_{rot}. Given the inequality $\sin\theta < \theta$, we can define the alternative bound

$$d'_{rot}(\mathbf{q}, \mathbf{p}) = r_{max}|q_\theta - p_\theta|.$$

Combining this alternative bound on the rotational distance with the translation distance, we arrive at a bubble with a conic shape as shown in Figure 4.7. Another way to view this alternative rotational distance is the arc distance traveled by a point on a circle of radius r_{max}. Such a distance is linear with the angle of rotation.

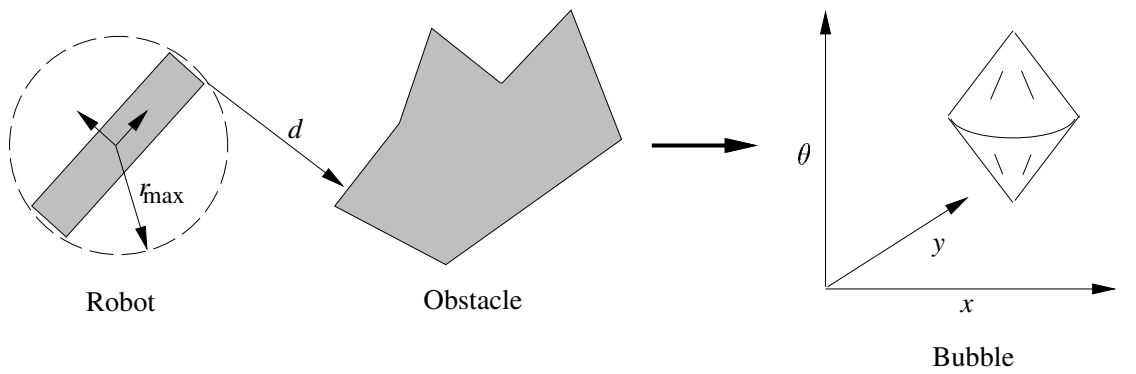


Figure 4.7: A convex bubble for a rotating robot.

4.7 Open Chain Manipulators

Let us now consider the case of open chain manipulators, the type of robot that motivated much of the work in this thesis. These robots are common in industrial situations, yet algorithms in robotics often have difficulty handling their many degrees of freedom and complex motions. As shown below, it is relatively easy to generate bubbles for manipulators.

For manipulators, the most natural configuration space is the displacement in each joint, often referred to as *joint space*. For an n degree-of-freedom manipulator, a configuration is described using the n joint variables q_i. In this section, we consider only manipulators with revolute joints; the extension to prismatic or screw joints is straightforward. For revolute joints, $q_i \in [0, 2\pi)$.

Let d be the minimum distance between the manipulator and the obstacles in the environment. Also, suppose that for each joint of the manipulator, a radius r_i is determined such that the cylinder centered along the axis of the joint contains all the subsequent links of the manipulator. For example, consider the two-degree-of-freedom manipulator shown in Figure 4.8. Since the manipulator is planar, the bounding cylinder becomes a circle. The circle of radius r_1 contains the entire manipulator, while the circle of radius r_2 contains link 2.

In the same fashion as for the previous types of robots, a set of configurations around $\mathbf{q}$ can be determined such that, along a specified path, no point on the manipulator moves a distance greater or equal to d. Consider moving the manipulator to a configuration $\mathbf{p}$ by first rotating joint 1 to position p_1. From the bounding cylinder at joint 1, we know that no part of the manipulator will move a distance greater than $r_1|p_1 - q_1|$. Next, rotate joint 2 to p_2—link 1 does not move, and the other links move a distance no greater than $r_2|p_2 - q_2|$. Each joint of the manipulator is moved in turn, and the total distance traveled by these motions is bounded by

$$\sum_{i=1}^{n} r_i|p_i - q_i|. \tag{4.5}$$

This expression represents a type of weighted 1-norm distance between $\mathbf{p}$ and $\mathbf{q}$ where the weightings are given by the different values of r_i.

Using expression (4.5) and the distance d, we can specify a joint space bubble $\beta_{\mathbf{q}}$ by

$$\beta_{\mathbf{q}} = \{\mathbf{p} : \sum_{i=1}^{n} r_i|p_i - q_i| < d\}. \tag{4.6}$$

The resulting bubble will be diamond shaped. For a two-degree-of-freedom manipulator, the diamond will have a ratio of width to height equal to the ratio of $1/r_1$ to $1/r_2$. We have already seen an example of such a bubble in the beginning of the Chapter (see Figure 4.1). Figure 4.9 illustrates two more examples. For both examples, the configuration space obstacles are shown for reference. Note that the configuration space obstacles are not needed to calculate the bubbles.

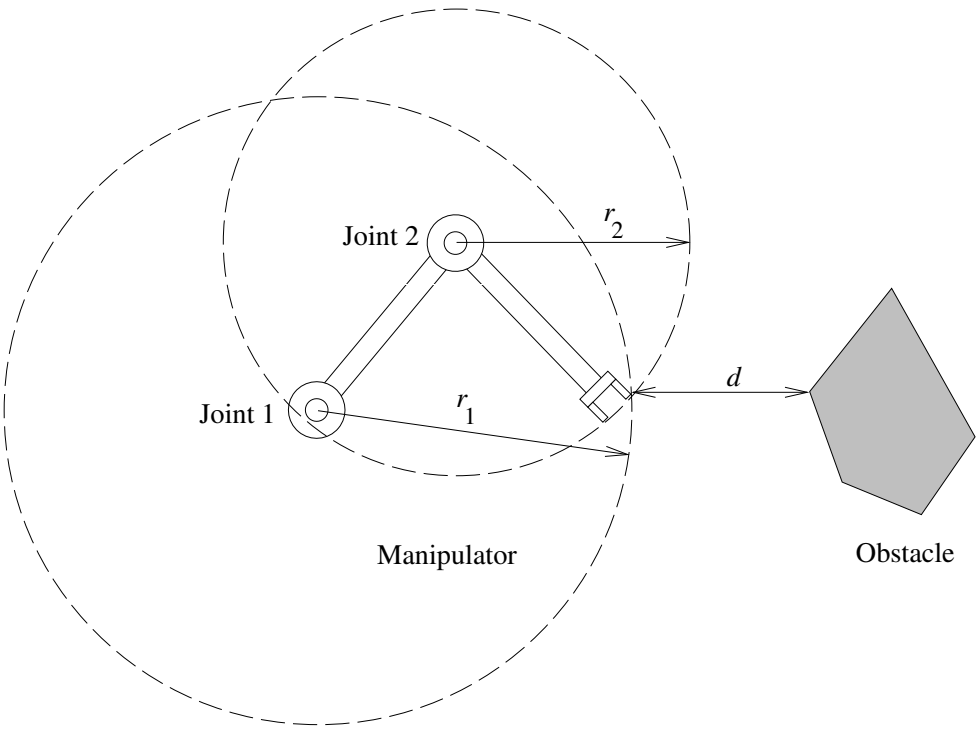


Figure 4.8: A two-degree-of-freedom manipulator. The minimum distance between the manipulator and the obstacle is given by d. The circle at the axis of joint 1 of radius r_1 contains the entire manipulator. The circle at joint 2 of radius r_2 contains link 2.

The bubble shown in Figure 4.9b extends almost to the boundary of C_{obst}. This indicates that a motion of the type used to specify the bubble will bring the manipulator almost into contact with the obstacles. As can be seen, this motion corresponds to a clockwise rotation of joint 2. Such a bubble is desirable as it indicates that the distance information is being well utilized.

The bubble shown in Figure 4.9d does not have the above property. The bubble is small, yet it is far from the configuration space obstacles. Examining the relationship between the manipulator and the obstacles in Figure 4.9c illuminates the problem. The manipulator is indeed close to the obstacle in the bottom right hand corner but kinematic constraints prohibit any collision. The motion of link 1 is such that this link cannot come in contact with the obstacle in question. Link 2 can collide at the given point, but by only a relatively large motion.

The discrepancy between the size of the bubble in Figure 4.9d and the distance to collision reflects the fact that equation (4.6) considers the manipulator to be surrounded by obstacles at a distance d. To generate less conservative bubbles would require a different type of distance information.

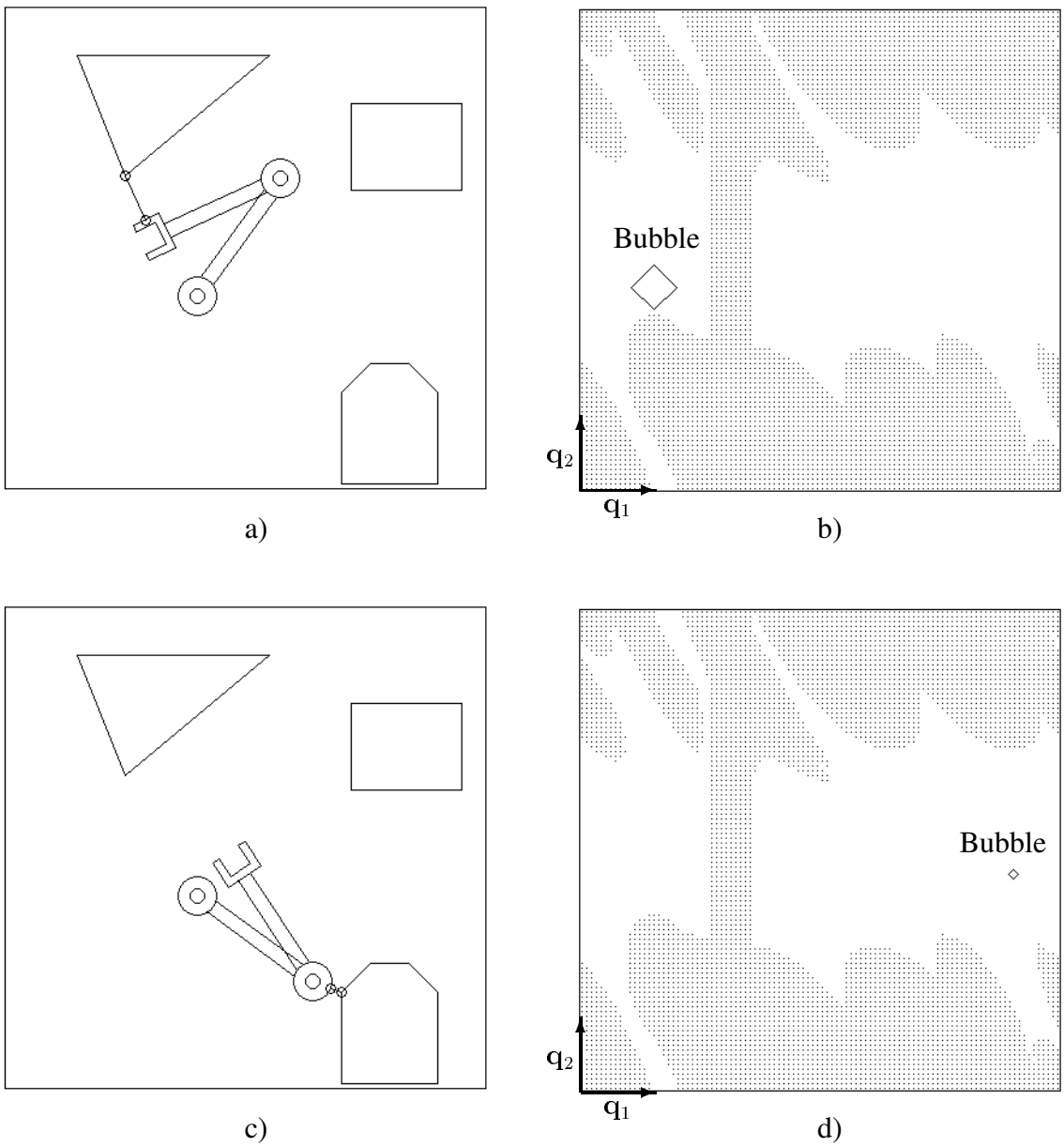


Figure 4.9: Two example bubbles for a manipulator

As an example of where different distance information is needed, consider the situation where the manipulator is attached to a base. We could argue that the distance between the manipulator and the environment is always zero and the above bubbles would be empty. Such a situation is obviously rather common. To avoid this problem we could set d in equation (4.6) to the lesser of the distance between link 1 and all the obstacles in the environment other than the base, and the distance between the other links and everything in the environment.

Joint limits

The bubble derived in equation (4.6) ignores the effect of joint limits, which are a characteristic of most manipulators. Assuming the joint limits for each link are independent, we can place the joint constraints in the expression used to define the bubble. If the minimum and maximum position of each joint is given in the vectors $\mathbf{q}^{min}$ and $\mathbf{q}^{max}$ respectively, a bubble can be specified as

$$\beta_{\mathbf{q}} = \{\mathbf{p} : \sum_i r_i|p_i - q_i| < d \text{ and } q_i^{min} < p_i < q_i^{max}\}. \tag{4.7}$$

The problem with this selection is that the shape of the resulting bubbles becomes more complex as shown in Figure 4.10; such shapes would require, in general, many parameters for their explicit representation.

A better solution to the joint limit problem is based on the observation that the bubble in Figure 4.10 is still convex. If the maximum and minimum position of each joint is calculated separately, then as the constraints are linear, any convex combination of these extreme joint positions will reside in C_{free}. Mathematically, the extreme joint positions $\mathbf{p}^{max}$ and $\mathbf{p}^{min}$ are determined by

$$\begin{array}{rcl} p_i^{max} & = & \min(q_i + d/r_i,\, q_i^{max}), \\ p_i^{min} & = & \max(q_i - d/r_i,\, q_i^{min}). \end{array} \tag{4.8}$$

A bubble can then be specified by

$$\beta_{\mathbf{q}} = \{\mathbf{p} : \mathbf{p}_i = q_i + a_i p_i^{min} + b_i p_i^{max} \text{ and } \sum_i |a_i| + |b_i| = 1\}. \tag{4.9}$$

Such a bubble is shown in Figure 4.11.

Self collision

Another desirable constraint on the motion of a manipulator is that its various links do not collide with each other. Collision between successive links can be avoided by the appropriate selection of joint limits. Avoiding collision between links that are not directly

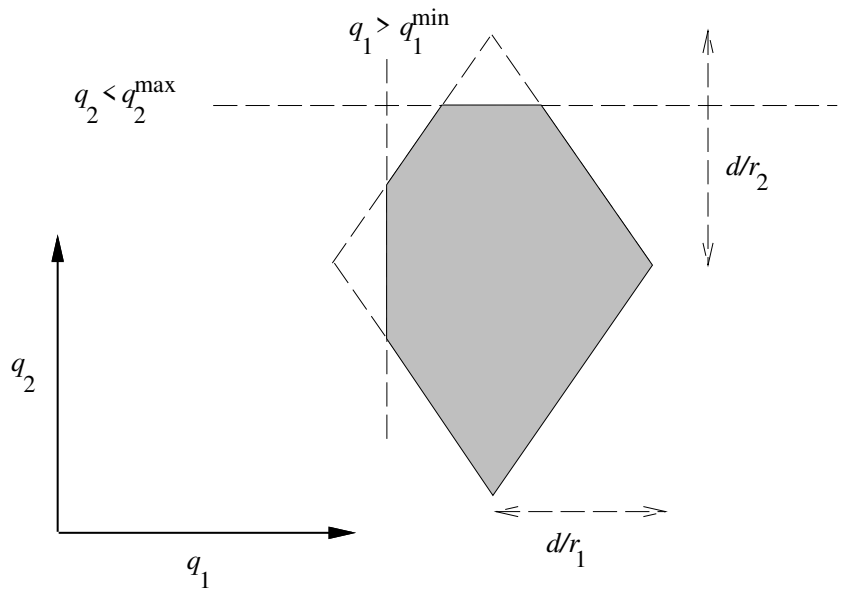


Figure 4.10: A bubble for a two-degree-of-freedom manipulator with joint limits.

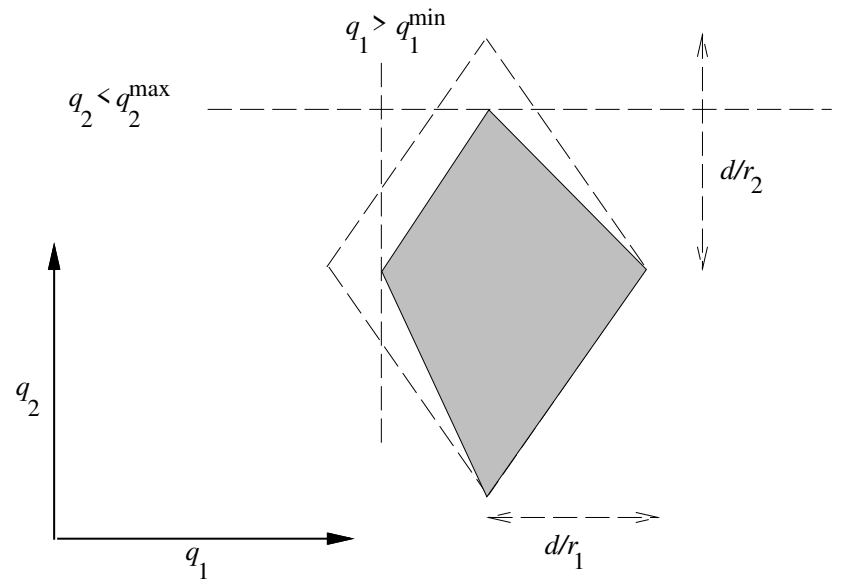


Figure 4.11: A simpler bubble for a two-degree-of-freedom manipulator with joint limits.

adjacent can be achieved by a slight change in the distance information used to construct the bubble.

Let d'_i be the minimum distance between the ith link of a manipulator and the combination of the obstacles in the environment and the other non-adjacent links of the robot. Let d' be the minimum of the various distances d'_i. If d' is used in equation (4.8) instead of d, the resulting bubbles defined by (4.9) will be free of collision between non-adjacent links. To see this property, suppose that non-adjacent links i and j could collide, where $i < j$. First, links i and j are at least a distance d'_i apart by definition, and hence are at least a distance d' apart. Now, because the manipulator is an open chain, the motion of link 1 thru i will not affect the relative distance between links i and j. In addition, the motion of link $i + 1$ thru j does not effect the position of link i. Thus, since the links collide, the motion of links $i + 1$ thru j must move link j a distance of at least d'. This is a contradiction, since the bubble is defined so that no part of the robot moves a distance of d' or greater.

Chapter 5

Implementation of Elastic Bands

This chapter describes how we implement elastic bands. An elastic is represented as a finite series of configurations with the constraint that the bubbles at successive configurations overlap; this constraint ensures that the elastic band represents a collision-free path. To modify the elastic, the configurations are moved utilizing artificial potentials.

5.1 Representing an Elastic Band

In Chapter 3, a theoretical model for an elastic band was developed. This model was based on representing a path as a parameterized function $\mathbf{c}(s)$ where $s \in [0, 1]$. Since it is infeasible to represent an arbitrary real-valued function in a computer, an approximate model that can be implemented must be developed.

In the current implementation, an elastic is represented as the position of a finite sequence of particles corresponding to equally-spaced values of s, i.e., for the selected values of s, $\mathbf{c}(s)$ is specified. The position of the infinite number of other particles is interpolated from the specified particles.

A problem with the above approach is ensuring that the elastic band represents a collision-free path. By selecting a large enough set of particles along the elastic band and checking that the robot is not in collision at the corresponding configurations, we can be fairly certain that the elastic indeed describes a collision free path. But how is the number of particles determined? Using too many will increase the processing time needed to perform the check. Using too few will decrease the likelihood that the elastic band in fact represents a collision-free path as an obstacle could intervene between adjacent particles.

In Chapter 4 we described the bubble concept, a region of free-space around a given configuration of the robot generated from distance information. Using bubbles, we can guarantee the specified particles represent a collision-free elastic.

Suppose an elastic is represented by m particles, and at a given instant, these particles are at configurations $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$ respectively. Also, suppose the bubbles at consecutive

configurations overlap, i.e.,

$$\beta_{\mathbf{q}_i} \cap \beta_{\mathbf{q}_{i+1}} \neq \emptyset, \quad i = 1, 2, \ldots, m-1. \tag{5.1}$$

It is possible to construct a collision-free path from $\mathbf{q}_1$ to $\mathbf{q}_m$. For example, one possible path can be constructed as follows. Between each pair of neighboring particles, a configuration is found that lies in both the corresponding bubbles; such a configuration exists as the bubbles overlap. Consider the line segments from the particles to this shared configuration. Since bubbles are restricted to convex shapes, these line segments will lie in the free-space, and together, they form a collision-free path from one particle to the other as illustrated in Figure 5.1. This process can be repeated down the sequence of particles to construct the desired path.

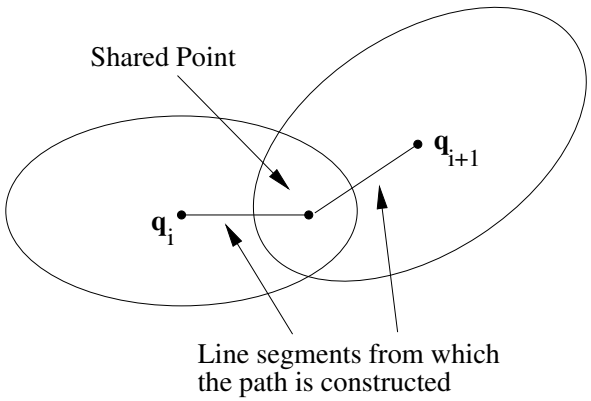


Figure 5.1: Constructing a path through two bubbles.

Figure 5.2 illustrates, for a non-rotating planar robot, the bubbles around a series of configurations of the robot. As can be seen, consecutive bubbles overlap and the interpolation of particles, depicted by the path, remains within these bubbles—such an elastic is collision free. Note that the bubbles and path are in the configuration space, while the robot and the obstacles are in the environment space. These two spaces are superimposed in the figure as they are closely related; they have the same dimension, the same coordinate axes, and the configuration space obstacles are generated from the environment obstacles by a simple expansion based on the shape of the robot [38].

As mentioned above, the interpolated particles can be generated by line segments. In Figure 5.2, however, the depicted path is not constructed in that manner; the details for computing the interpolation shown are given in Chapter 7, but the important point is the interpolated particles remain with the bubbles of the specified particles. The advantage of the smooth path shown in the figure is it removes the discontinuities in direction present in the simple line segment path; such smooth paths are preferred from a control point of view.

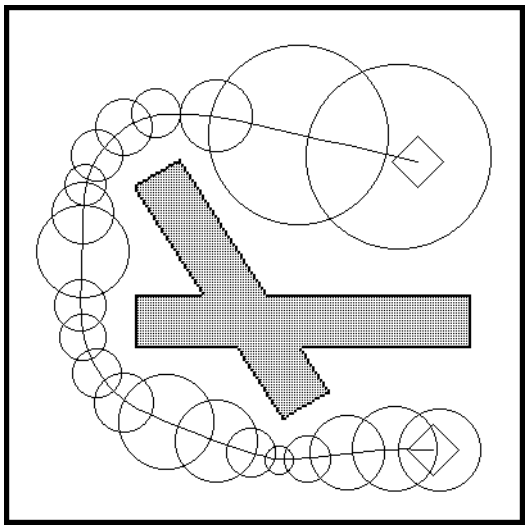


Figure 5.2: A path though a series of bubbles

As shown in Chapter 4, bubbles can be generated efficiently for robots with many degrees of freedom. Figure 5.3 shows the configurations used to represent an elastic band for a planar stick-like robot that can rotate, and Figure 5.4 depicts an elastic for a planar six-degree-of-freedom manipulator. Note, for the first example, the configuration space and bubbles have dimension three. For the second example they have dimension six. In these illustrations, the bubbles associated with the particles of the elastic are not shown due to the difficulties of displaying three- and six- dimensional objects. Although they are not displayed, the bubbles associated with successive particles overlap with each other.

5.2 The Forces on a Bubble

The magnitude and direction of the motion of each particle is determined by calculating a force. The force is composed of three components: internal, external, and constraint. In the current implementation, the internal force acts to remove slack in the elastic band, the external force moves the elastic away from the configuration space obstacles, and the constraint force inhibits motion of particles along the elastic band.

As described in Chapter 3, the internal and external forces for the continuous model of an elastic band are derived from potential functionals. For the implementation, the elastic is represented discretely and the corresponding potentials are functions of a finite number of parameters—the resulting forces can thus be determined by the standard gradient operator ∇. In the following two subsections these potential functions are described and the corresponding forces acting on the particles are derived.

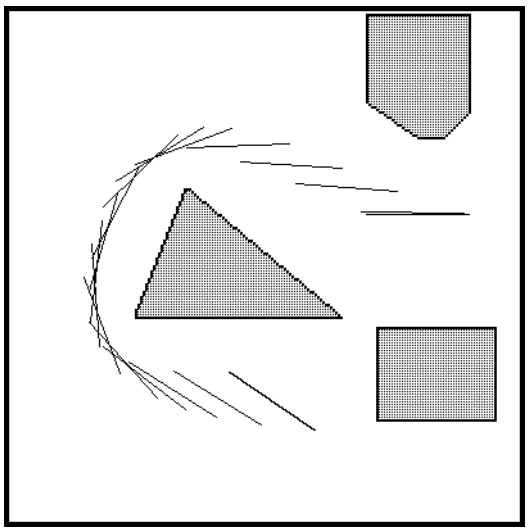


Figure 5.3: The configurations describing an elastic band for a robot.

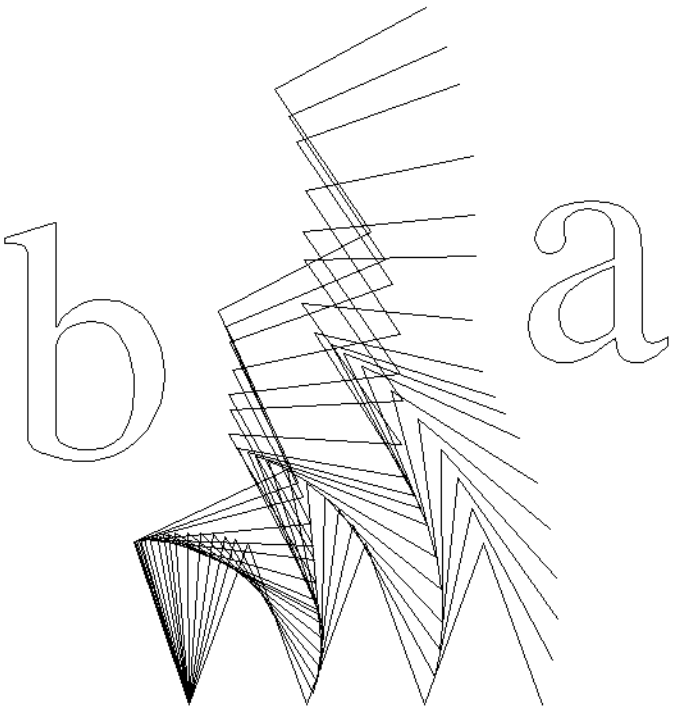


Figure 5.4: The configurations describing an elastic band for a manipulator.

The internal contraction force

For the continuous elastic band model (see Chapter 3), the internal potential functional is specified as

$$V_{int}[\mathbf{c}] = k_c \int_0^1 \|\mathbf{c}'\|\, ds,$$

where k_c is the constant of elasticity.

For the implementation, the sequence of configurations $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$ that describes the elastic band corresponds to values of $\mathbf{c}(s)$ at regularly spaced values of the parameterization variable s. If h is the distance between successive values of s, a discrete equivalent to the continuous potential is

$$\begin{aligned} V_{int}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n) &= k_c \sum_{i=1}^{m-1} h \frac{\|\mathbf{q}_{i+1} - \mathbf{q}_i\|}{h} \\ &= k_c \sum_{i=1}^{m-1} \|\mathbf{q}_{i+1} - \mathbf{q}_i\|. \end{aligned} \tag{5.2}$$

The contraction force on the particle at $\mathbf{q}_i$ may be derived from equation (5.2) by the negative gradient of V_{int} with respect to $\mathbf{q}_i$. Since $\mathbf{q}_1$ and $\mathbf{q}_m$ are the start and goal configurations, these quantities are constant and no force need be calculated. For each of the other configurations, the right of side of (5.2) contains two dependent components. Performing the partial differentiation, the contraction force on particle i where $1 < i < m$ is given by

$$\begin{aligned} \mathbf{f}_{int}(i) &= -\nabla_{\mathbf{q}_i} V_{int} \\ &= -k_c \frac{\partial}{\partial \mathbf{q}_i} \left(\|\mathbf{q}_{i+1} - \mathbf{q}_i\| + \|\mathbf{q}_i - \mathbf{q}_{i-1}\| \right) \\ &= k_c \left(\frac{\mathbf{q}_{i+1} - \mathbf{q}_i}{\|\mathbf{q}_{i+1} - \mathbf{q}_i\|} + \frac{\mathbf{q}_{i-1} - \mathbf{q}_i}{\|\mathbf{q}_{i-1} - \mathbf{q}_i\|} \right) \end{aligned} \tag{5.3}$$

This equation captures the nature of uniform tension. Each particle experiences two forces of magnitude k_c directed towards its two neighbors. Alternatively, each particle pulls its two neighbors towards it with a force k_c.

The external repulsion force

For the continuous model, the external potential functional is specified as

$$V_{ext}[\mathbf{c}] = \int_0^1 v_{ext}(\mathbf{c})\, ds,$$

where v_{ext} is some potential function over the configuration space of the robot. The discrete equivalent to this potential is

$$V_{ext}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n) = \sum_{i=1}^{m} h v_{ext}(\mathbf{q}_i). \tag{5.4}$$

Notice that equation (5.4) is linearly dependent on the value of h, while equation (5.2) is not. This difference raises the question as to a suitable value for h. In Chapter 3 we stated that $s \in [0, 1]$. Spacing the m configurations evenly in s gives h a value of $1/(m-1)$. The selection of the range $[0, 1]$ is completely arbitrary, and for reasons explained below, it is advantageous to set $h = 1$.

The form of equation (5.4) enables considerable latitude in the design of the external force—one can select any potential function over the configuration space. The potential we use is described in the following; however, there are many other possibilities such as the potentials described by Khatib [33].

The current usage of the external force is to repel the elastic from the configuration space obstacles. Such a force will give the path clearance from the obstacles, and by careful design, will interact with the internal contraction force to generate smooth paths.

In Khatib's artificial potential field method, the repulsion force is designed to increase towards infinity as the robot approaches the boundary of C_{free}. Such a force ensures the robot does not collide with the obstacles. In contrast, the repulsion force in the current implementation does not increase in such an unbounded fashion. Instead, the algorithm that modifies an elastic band explicitly takes into account the collision-free constraint when deforming a path. This approach has the advantage of avoiding numerical problems due to the large forces near the boundary of C_{free}.

Assuming $d(\mathbf{q})$ is the minimum distance between the robot at configuration $\mathbf{q}$ and the obstacles in the environment, we propose that the external potential energy $v_e xt$ is given by

$$v_{ext}(\mathbf{q}) = \begin{cases} \frac{1}{2} k_r (d_0 - d(\mathbf{q}))^2 & \text{if } d(\mathbf{q}) < d_0 \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

where k_r is a user specified repulsion gain and d_0 is the distance up to which the force is applied. Equation (5.5) takes the form of a linear spring; the energy is proportional to the displacement squared, where the displacement is the amount the robot has breached the desired clearance d_0. Clearance greater that d_0 is not detrimental and thus the potential is one-sided.

Determining the force associated with equation (5.5) is non-trivial. For many environments there will exist a manifold of points in the configuration space where the force is not defined. Recall that the force associated with a potential function is derived from the gradient operator ∇. Consider a situation in which the robot is equidistant from two obstacles as shown in Figure 5.5, and assume d for this configuration is less than d_0. It

should be clear that d will decrease, and the potential function will increase, if the robot moves either to the left or right. As a consequence, the derivative of v_{ext} with respect to the horizontal direction is not well defined, and the resulting force cannot easily be computed.

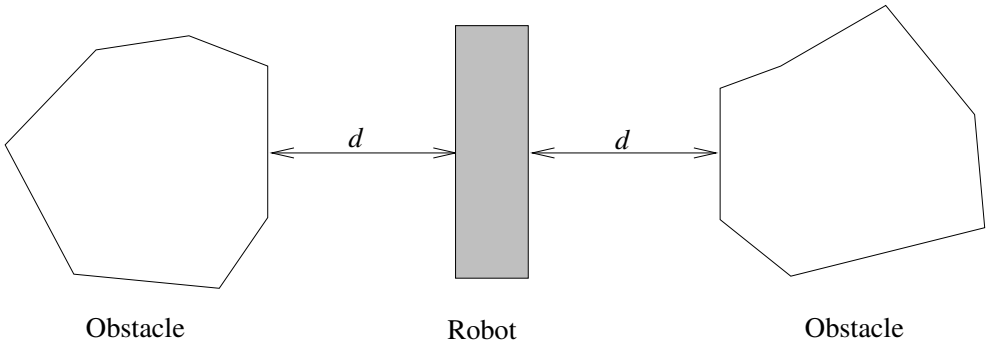


Figure 5.5: When the robot is an equal distance d from two obstacles and $d < d_0$, the potential v_{ext} is not differentiable.

Ignoring such problem configurations for a moment, let us attempt to compute the force associated with equation (5.5) when $d < d_0$.

Suppose, when calculating the minimum distance d, we record the minimum vector $\mathbf{d}$ from the obstacles to the robot. More mathematically, if $R_{\mathbf{q}}$ is the region of the environment occupied by the robot in configuration $\mathbf{q}$ and O is the region occupied by the obstacles, $\mathbf{d}$ is the vector

$$\mathbf{d} = \mathbf{x} - \mathbf{y},\ \mathbf{x} \in R_{\mathbf{q}},\ \mathbf{y} \in O$$

such that

$$\|\mathbf{d}\| = d.$$

The points $\mathbf{x}$ and $\mathbf{y}$ are the closest points between the robot and the environment. Typically, such information is calculated in order to determine d in the first place (see Chapter 6). Note that for some configurations of the robot, such as shown in Figure 5.5, the vector $\mathbf{d}$ is not unique. These configurations are the ones for which the repulsion force is not defined.

After moving the robot some infinitesimal amount, suppose $\mathbf{x}$ and $\mathbf{y}$ are still the closest points between the robot and the environment. Using the Jacobian $\mathbf{J}_{\mathbf{x}}$ associated with the point $\mathbf{x}$ of the robot, the force associated with (5.5) is given by

$$\begin{aligned} f_{ext}(\mathbf{q}) &= -\nabla v_{ext} \\ &= -\frac{\partial v_{ext}}{\partial \mathbf{q}} \\ &= -\frac{1}{2} k_r\, \mathbf{J}_{\mathbf{x}}^T\, \frac{\partial}{\partial \mathbf{x}} (d_0 - \|\mathbf{d}\|)^2 \\ &= k_r\, \mathbf{J}_{\mathbf{x}}^T\, (d_0 - d) \frac{\mathbf{d}}{d} \end{aligned} \tag{5.6}$$

As can be seen, this force is proportional to $(d_0 - d)$ along the unit vector $\mathbf{d}/d$, as we would expect from the spring-like energy function.

The above analysis breaks down when $\mathbf{x}$ and $\mathbf{y}$ do not remain the closest points during the differential motion. In particular, if $\mathbf{x}$ and $\mathbf{y}$ change suddenly, as would occur for small motions of the robot in Figure 5.5, then the gradient of the potential function may be undefined. Even when such a discontinuity does not occur, we expect $\mathbf{x}$ and $\mathbf{y}$ to move continuously along the surface of the robot and the environment. To take in to account this motion is difficult, however, as it requires a parameterization of the objects' and robot's surfaces, and even then, will not work for polygonal models.

Even through equation (5.6) is not perfect, it can still be used to move a particle of the elastic band. The details are given below, but the basic idea is to use (5.6) when determining a direction of motion, and by monitoring the sum of the potential functions, to ensure that the energy of the elastic is decreasing. In addition, if such an approach does not succeed, an approximate gradient is calculated employing finite difference.

The motion constraint

In the continuous model of the elastic band, we added the constraint that particles do not move along the elastic. This constraint can be implemented by calculating a force to counteract any component of the other forces along the elastic, or by directly restricting the motion of the particles.

One issue is the specification of the direction along the elastic. For the continuous case, this is simply $\mathbf{c}'$. We use the symmetrical finite difference approximation to this derivative, i.e., the direction at particle i is

$$\frac{\mathbf{q}_{i+1} - \mathbf{q}_{i-1}}{2h}. \tag{5.7}$$

One result of using equation (5.7) is that, unlike the continuous case, the contraction force (5.3) may contain a component along this direction. An alternative direction that does not have this property is given by

$$\frac{\mathbf{q}_{i+1} - \mathbf{q}_i}{\|\mathbf{q}_{i+1} - \mathbf{q}_i\|} + \frac{\mathbf{q}_i - \mathbf{q}_{i-1}}{\|\mathbf{q}_i - \mathbf{q}_{i-1}\|}. \tag{5.8}$$

We have experimented with both these expressions and observed no discernible distinction.

5.3 Deforming the Elastic

We now describe how an elastic band is deformed. The overall strategy is to scan up and down the sequence of particles that describe the elastic, moving each in turn. To maintain the elastic band as a collision-free path, we must ensure that the bubble at each of these

particles overlaps with its immediate neighbors. This constraint may require particles to be added to the representation of the elastic. In addition, it is desirable to remove excess particles to improve efficiency. Figure 5.6 shows the bubbles around the particles describing an elastic as an obstacle moves in the environment. Figure 5.7 and Figure 5.8 depict the deformation of elastic for a three-degree-of-freedom robot and a six-degree-of-freedom manipulator respectively. For these two figures, the robot is drawn at the configurations of the particles.

The elastic is deformed by moving one particle at a time. Consider a particle at configuration $\mathbf{q}$; how do we select the next configuration $\mathbf{q}'$ for this particle? Given the force on a particle, one simple update procedure is: move the particle along the force, scaled by some constant factor α. Mathematically, we have

$$\mathbf{q}_i{}' = \mathbf{q}_i + \alpha \mathbf{f}_{total}. \tag{5.9}$$

The above update equation will approximately implement the pseudo-static simulation of the elastic described in Chapter 3.

Using equation (5.9) has two problems. First, there is no guarantee that the new configuration of the particle will be collision-free. Second, equation (5.9) implements an explicit Euler method which tends to be unstable [47]; in other words, the elastic may oscillate wildly and never come to rest. In the following, we describe solutions to these two problems.

Maintaining a collision-free elastic

Consider three neighboring particles of an elastic band at configurations $\mathbf{q}_{i-1}$, $\mathbf{q}_i$, and $\mathbf{q}_{i+1}$ respectively. If the elastic band is in a valid state, the pair of bubbles at $\mathbf{q}_{i-1}$ and $\mathbf{q}_i$ and the pair at $\mathbf{q}_i$, and $\mathbf{q}_{i+1}$ overlap as shown in Figure 5.9a.

Suppose the ith particle of the elastic is to be moved. We must ensure that the new configuration of the particle is in the free-space. This is achieved by restricting the motion of the particle such that it remains in the bubble at $\mathbf{q}_i$, i.e., the new position $\mathbf{q}_i' \in \beta_{\mathbf{q}_i}$. To ensure that the bubble at the new configuration has some volume, the motion is restricted to lie slightly inside the boundary of the bubble.

After moving a particle, the bubble at new configuration is determined. It may occur, as shown in Figure 5.9b, that the bubble does not intersect the two neighboring bubbles. Such a situation means that the set of particles does not represent a valid elastic band. Even though all the explicitly represented particles are in the free-space, the interpolated particles may not be.

To restore the validity of the representation of the elastic band, additional particles are added until the bubbles overlap as illustrated in Figure 5.9c. Although the details are tedious, it is not difficult to see that it is always possible to generate a valid representation for an elastic by the addition of particles. Suppose the bubble at $\mathbf{q}_i'$ does not overlap the bubble

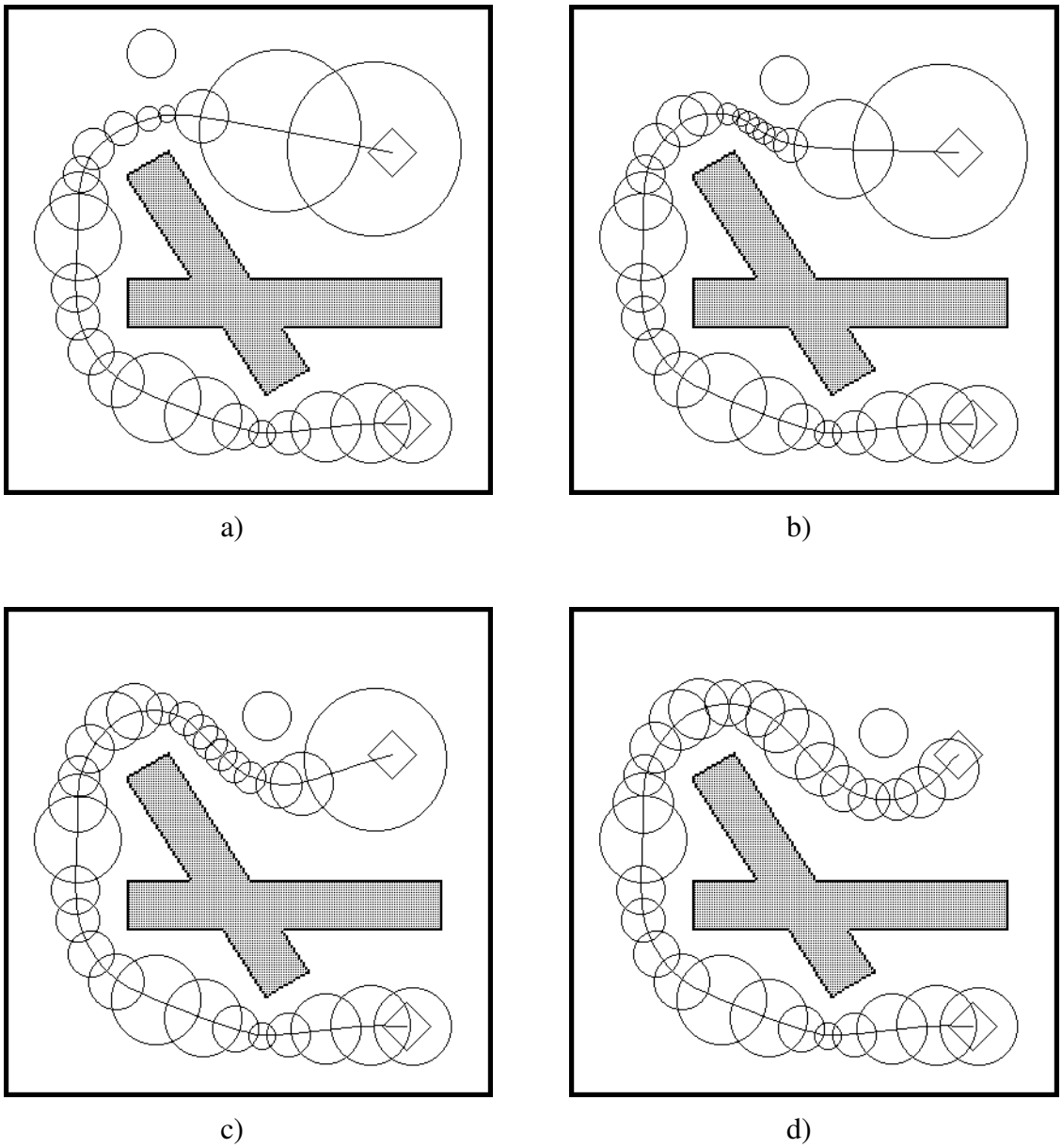


Figure 5.6: As the circular obstacle moves, the elastic deforms to minimize the forces. If needed, extra particles are inserted and deleted to maintain a collision-free path.

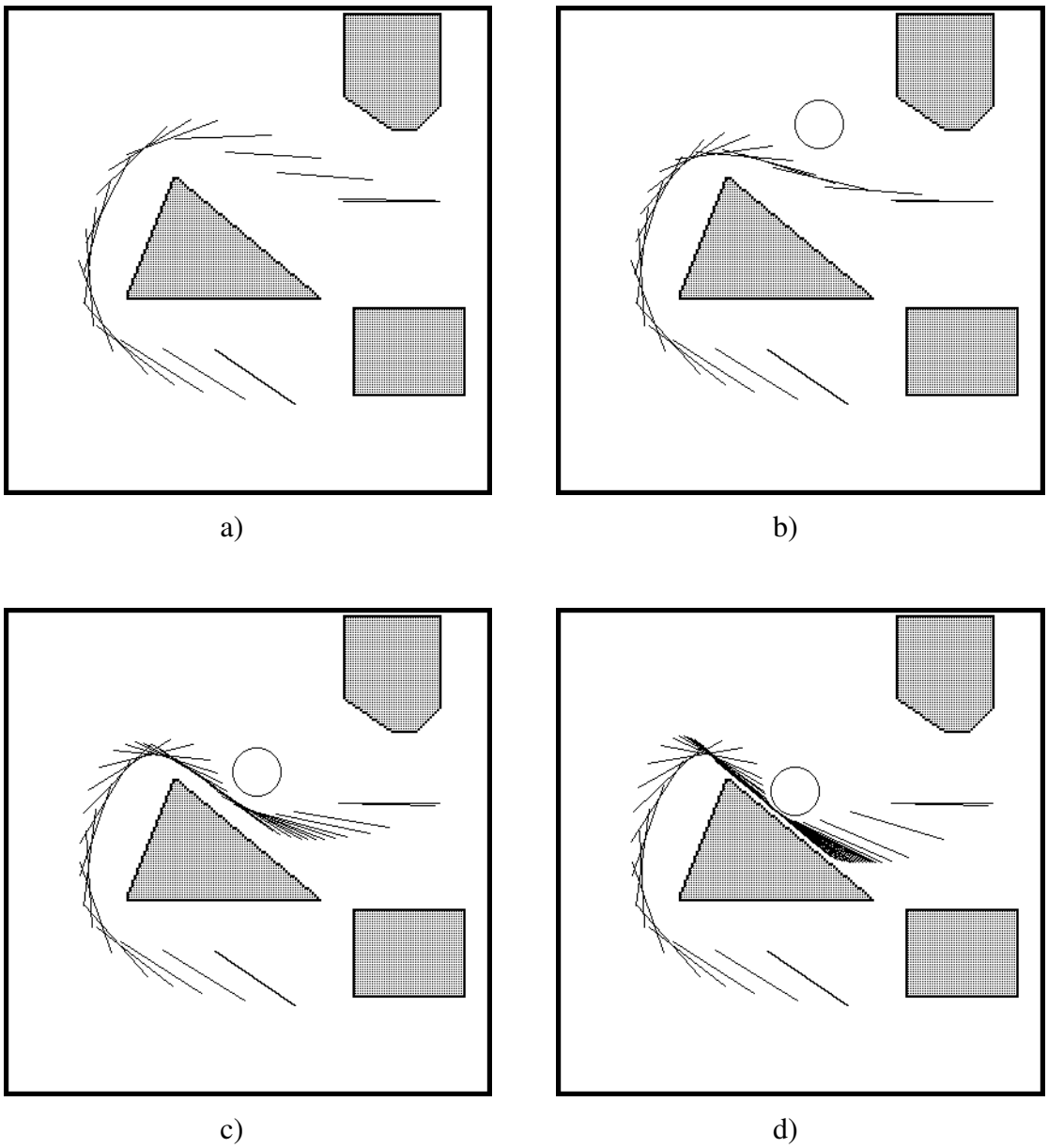


Figure 5.7: An elastic band for a stick robot deforms to avoid an obstacle.

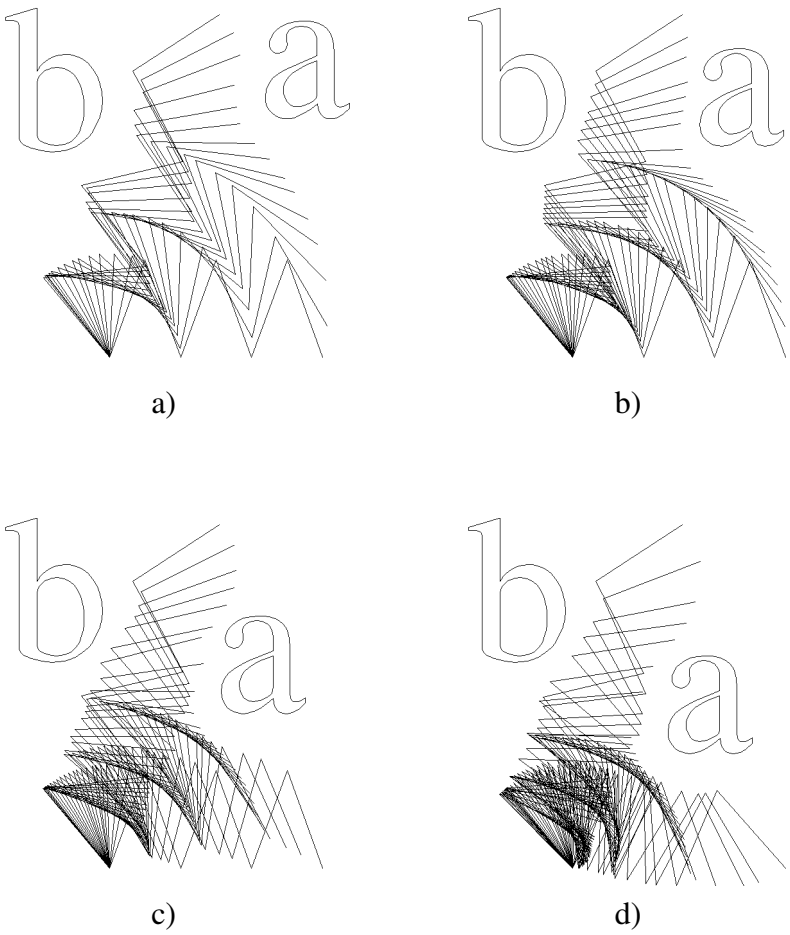


Figure 5.8: An elastic band for a manipulator deforms to avoid an obstacle.

$\mathbf{q}_{i+1}$ as in Figure 5.9b. Assuming the elastic was in a valid state before the particle was moved, the bubble at $\mathbf{q}_i$ does overlap the bubble at $\mathbf{q}_{i+1}$ and there exists some configuration, say $\mathbf{p}$, in the intersection of these two bubbles. The line segment between $\mathbf{q}_i'$ and $\mathbf{p}$ will remain entirely within the bubble at $\mathbf{q}_i$ and thus is collision-free. The elastic band can be reconnected by adding particles along this line segment.

After adding particles to the representation of the elastic band, subsequent particles are renumbered.

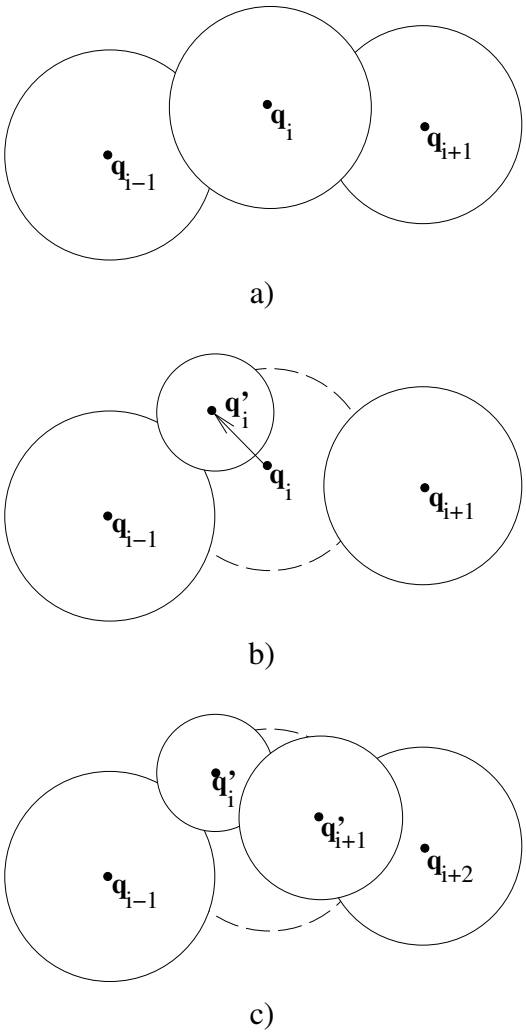


Figure 5.9: Moving a particle. **a)** The initial state of the elastic. **b)** The ith particle is moved from configuration $\mathbf{q}_i$ to $\mathbf{q}'_i$. The new configuration must be within the bubble $\beta_{\mathbf{q}_i}$. **c)** The bubble at configuration $\mathbf{q}'_i$ does not overlap the bubble around the particle at $\mathbf{q}_{i+1}$. To ensure the particles represent a collision-free elastic, a particle at configuration $\mathbf{q}'_{i+1}$ is added. The other particles in the elastic are renumbered.

The effect of adding particles

Adding particles to the elastic can be viewed in two way: representing the elastic at more values of s, or increasing the amount of material that the elastic band is composed of. For the current implementation, the second interpretation is used.

Earlier in this chapter, the discrete representation of the elastic was described as the configuration $\mathbf{c}(s)$ of a set of particles corresponding to equally spaced values of s where h is the spacing between the particles. Suppose a particle is added between the particles s and $s + h$. We could consider this as the addition of the particle $s + h/2$ to the representation. In other words, the values of s for the represented particles are no longer equally spaced in s, but are determined adaptively so that the bubbles around the corresponding particles overlap.

Representing the elastic with particles at unequally spaced values of s changes the development of the external and internal force described earlier in the chapter. To implement these changes, the value of s associated with each particle needs to be maintained.

An alternative interpretation for the addition of particles to the representation is that the amount of material in the elastic is increased. Near the end of Chapter 3, we proposed changing the length of the elastic as a scheme for decoupling the internal force from the amount of stretch in the elastic. The intuitive idea is to add material to a section of the elastic when the local amount of stretch increases past some point. The addition of material will decrease the local stretch. Similarly, if the local stretch decreases too much, material is removed.

In general, the addition of particles, as described in the last subsection, occurs when the elastic is being pushed by some moving obstacle. When the bubbles associated with the represented particles no longer overlap, it seems reasonable to treat this as a situation where the local stretch has increased too far. Of course, the true local stretch is determined by the distance between particles in the configuration space. We combine the addition of particles to re-obtain a valid elastic band representation and the addition of material to reduce the local stretch.

Treating the addition of particles as the addition of elastic material has the consequence that the values of s for the particles can remain equally spaced. When a particle is added, we consider the range for s to be increased by the constant h. Since h is arbitrary, we assign it a value of one, resulting in the particles having integer values of s.

Stable motion of particles

Recall that the internal and external forces are specified as the negative gradient of potential functions and the constraint force does no work. These conditions imply that the motion of a particle should always decrease the total energy of the elastic. If the update procedure ensures the total energy does indeed decrease when a particle is moved, the motion of the elastic will be stable.

Using equation (5.9) does not enable us to make any statement about the change in the energy of the elastic. Although moving in the direction indicated by the force should decrease the energy, a finite step in that direction may not; it depends on the value of α. Since α is an arbitrary constant, it seems reasonable to move in the direction given by the force until the energy stops decreasing or until the edge of the bubble is reached. This procedure corresponds to a bounded one-dimensional minimization of the potential along the direction given by the force at the initial configuration.

Using a minimization procedure has an added advantage when the external force is considered. As mentioned earlier, the gradient of the external potential does not exist in some configurations, and even when it does exist, it is difficult to determine precisely. Using the force suggested in equation (5.6) may not move the elastic in a direction that decreases the potential function. The minimization procedure will not completely solve this problem, but at least the effect of such a force will be that the particle does not move. In other words, an incorrect force will cause the elastic to get stuck, but will not cause instabilities. The details of the minimization procedure are important and are given later in the chapter.

Even though the motion of a particle reduces the total energy of the elastic, two other effects can increase the energy: motion of the obstacles and the addition of elastic material.

If the obstacles in the environment move, the external potential function can change in an arbitrary manner. Once the obstacles stop moving, the energy of the elastic should start decreasing and eventually come to rest.

The addition of particles to the elastic band representation can also increase the energy of the elastic. If the addition of particles was considered as representing the elastic at more values of s (see the last subsection), the changes in the energy would correspond to different sampling of a continuous function and could probably be neglected. In the current implementation, the addition of particles is viewed as the addition of elastic material, which cannot so easily be ignored. In practice, however, we have found the addition of particles does not seem to reduce the stability of the elastic band.

Removing particles

Another modification to the elastic band is to remove redundant particles. The series of particles of the elastic are scanned for situations in which the bubbles overlap for a pair of particles that are not immediate neighbors. If such a pair is found, the particles between the two are removed from the representation. For example, in Figure 5.10, the bubble at $\mathbf{q}_i$ can be removed as the bubbles at $\mathbf{q}_{i-i}$ and $\mathbf{q}_{i+i}$ overlap. For efficiency, only pairs of particles separated by one particle are considered.

Removing particles from the elastic band increases the efficiency with which the elastic can be modified. Also, to be consistent with the addition of particles, the removing of particles is viewed as a reduction in the amount of material in the elastic.

From an energy perspective, the removal of particles always decreases the energy of

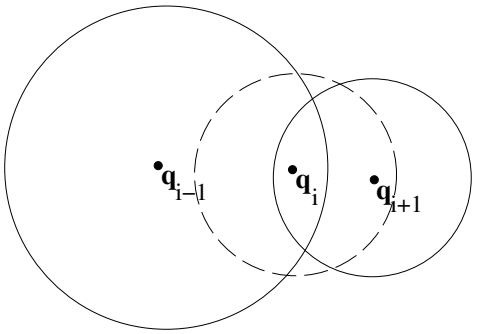


Figure 5.10: A situation in which the particle at $\mathbf{q}_i$ can be removed from the elastic. The particle can be removed because the bubbles at $\mathbf{q}_{i-1}$ and $\mathbf{q}_{i+1}$ overlap.

the elastic. One precaution must be taken. It is possible for a cycle to occur in which a particle is removed, the elastic deforms slightly, a particle is added, the elastic deforms, a particle is removed, etc. To avoid such a situation, a small amount of hysteresis is added. In particular, a particle is removed only if the neighboring two bubbles overlap to a certain degree. In our current implementation, we require 20 percent overlap.

5.4 Minimization Procedure

We desire to move a particle in a given direction until the potential energy of the elastic band stops decreasing or until the edge of a given bubble of free space is reached. Such a procedure corresponds to a bounded one-dimensional minimization of the potential along the given direction. In this section, we develop a slightly non-standard method for performing this minimization.

There exist many efficient and robust methods for bounded one-dimensional minimization [47]. A typical algorithm efficiently decreases a bounded segment of the real line that contains the desired minimum. A good example is the method by Brent [9].

For the implementation of elastic bands, the calculation of the potential energy of the elastic is relatively expensive. In particular, the external component of the elastic's potential energy is defined as the sum of the potential v_{ext}, given by (5.5), at the configuration of each particle of the elastic. To compute v_{ext} at a single configuration of the robot requires the minimum distance between the robot and the obstacles. To compute the total energy of the elastic requires a distance calculation for each particle of the elastic. In the case where one particle is moved, the change in the potential can be determined using only one distance calculation, but even one is expensive.

To increase efficiency, we have implemented a non-standard minimization procedure that attempts to evaluate the potential function v_{ext} as few times as possible. The idea is to repeatedly use a standard minimization algorithm with successive approximations to

v_{ext}. Each use of the standard minimization algorithm results in a configuration of the particle that minimizes the approximation function, but may or may not minimize v_{ext}. The potential function is sampled at this new configuration to determine if the energy of the elastic agrees with the approximation. If it does, the configuration is used as the new position of the particle. Otherwise, the approximation function is improved and the process is repeated.

To approximate the potential function v_{ext}, suppose the minimum distance between the robot and the obstacles has been determined at various configurations. In addition, suppose the points $\mathbf{x}_i$ and $\mathbf{y}_i$ were the closest points between the robot and the environment for the ith distance computation. The minimum distance between the robot and the environment for some given configuration can be approximated (and bounded) by the minimum distance between the pairs of points $\mathbf{x}_i$ on the robot and $\mathbf{y}_i$ in the environment, i.e.,

$$d(\mathbf{q}) \approx \min(\|\mathbf{x}_i(\mathbf{q}) - \mathbf{y}_i\|).$$

Note that the points $\mathbf{x}_i$ move with the robot while the points $\mathbf{y}_i$ are stationary in the environment. This approximate distance can be used to approximate the potential v_{ext}.

Selecting a direction of motion

It is well know in optimization theory [25] that successive one-dimensional minimizations along the instantaneous direction of steepest descent results in rather slow convergence to a stationary point. An intuitive explanation for this property is that the direction of steepest descent is the best direction of motion only at beginning of the step. During the step the best direction will change and thus the total reduction in the potential may be relatively small.

To overcome this problem we use an idea adapted from optimization theory [25]. Consider the first-order Taylor expansion of the force around a particle at configuration $\mathbf{q}$,

$$\mathbf{f}(\mathbf{q} + \delta\mathbf{q}) \sim \mathbf{f}(\mathbf{q}) + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} \delta\mathbf{q}. \tag{5.10}$$

Note that since $\mathbf{f} = -\nabla V$, the expression $\partial\mathbf{f}/\partial\mathbf{q}$ represents the negative Hessian of the potential V. Ideally, we would like to move a particle to a position in which the force is expected to be zero. If the gradient and Hessian are known, the approximation to the force given by (5.10) is zero along the direction $\delta\mathbf{q}$ where

$$\mathbf{f}(\mathbf{q}) + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} \delta\mathbf{q} = 0. \tag{5.11}$$

The constraint that particles do not move along the elastic prevents us from directly using the direction $\delta\mathbf{q}$ suggested by the solution of (5.11). A good direction of motion

should respect the constraint and minimize the expected value of the force. The constraint can be expressed mathematically as

$$\delta\mathbf{q} \cdot \mathbf{c}' = 0$$

where $\mathbf{c}'$ is the direction of the elastic at configuration $\mathbf{q}$. Using the well-known method of Lagrange undetermined multipliers [37], the direction $\delta\mathbf{q}$ that minimizes (5.10) and conforms to the constraint is a solution to the equation

$$\mathbf{f} + \frac{\partial \mathbf{f}}{\partial \mathbf{q}} \delta\mathbf{q} = \lambda \mathbf{c}',$$

where λ is some undetermined quantity. Writing this equation and the constraint in matrix form, we have

$$\begin{bmatrix} \partial\mathbf{f}/\partial\mathbf{q} & \mathbf{c}' \\ \mathbf{c}'^T & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{f} \\ 0 \end{bmatrix}.$$

If the dimension of the configuration space is n, we have $n + 1$ equations and $n + 1$ unknowns, δq and λ, which can be solved using standard linear algebra.

Computing a finite difference gradient

When the minimization procedure fails to find a configuration along the given direction that reduces the potential energy, a second attempt is made along an alternative direction. The alternative direction is selected by a finite difference approximation to the gradient of the potential. The idea is, by sampling the potential around the configuration, a promising direction for minimization can be determined. Since this direction does not use information about the gradient of the potential, it does not suffer from the non-differentiable regions of the external potential.

To avoid the expense of sampling the potential when computing the finite difference gradient, an approximation to the potential is used. The approximation is the same as the one used during the original minimization procedure as described earlier in the section.

5.5 Moving Multiple Particles Simultaneously

In the present implementation, the elastic is deformed by moving one particle at a time. By scanning up and down the sequence of particles, the effect of the motion of a particle will propagate along the elastic and potentially cause a global change. The number of scans of the elastic needed for the global changes to propagate, however, can be quite large. For example, if all the obstacles in the environment are removed, the elastic will deform to a straight line in the configuration space. The rate of convergence towards this final state will be approximately linear.

To increase the rate of convergence, one could envision moving multiple particles simultaneously; by extending the method described in Section 5.5, quadratic convergence should be possible. The difficulty with moving multiple particles, however, is maintaining a valid representation of the elastic. In particular, a method is needed for moving the particles such that they remain in the free-space, and perhaps after adding new particles, the bubbles along the elastic overlap. We have not yet developed such a procedure, but it appears an interesting area for future research.

5.6 Concluding Remarks

In this chapter, the current implementation of elastic bands has been described in some detail. The implementation seems rather far removed from the original idea of simulating the motion of an elastic material in the presence of artificial forces. We still feel, however, that the original physical model provided much inspiration and motivation for the development of the above ideas.

Chapter 6

Distance Computation

6.1 Introduction

Computing the distance between objects is a common problem in robotics. Using a mathematical model of two objects, a point is found on each object such that the distance between the points is minimized. If one object is a robot and the other object is the union of all the obstacles in the environment, such information describes how close the robot is to collision. Distance computation has been used for real-time collision avoidance [33], real-time path modification [49], and optimal path planning [6].

Previous work on the distance computation problem has focused on convex objects. Lumelsky [42] describes an efficient algorithm for pairs of line segments. Gilbert et al. [24], Bobrow [7], and Lin and Canny [39] present algorithms that find the distance between two convex polyhedra. Each of these three algorithms iteratively finds pairs of points, one on each object, such that the distance between the points monotonically converges to the minimum. These algorithms rely heavily on the properties of convex objects and it appears difficult to extend them directly to the non-convex case.

To compute the distance between non-convex objects, we can break each object into convex components and then use one of the above algorithms to determine the distance between components. The distance between the two objects is then the smallest distance between any pair of convex components. However, a naive implementation that examines all such pairs has complexity $O(nm)$, where n and m are the number of components of each of the objects.

Collision detection is related to distance computation; two objects are in collision if and only if the distance between the objects is zero. Collision detection has been used extensively in robotics for applications such as path planning [38], and in computer graphics for physical based modeling [2]. For such applications, collision detection often consumes a significant percentage of the execution time and much research has been directed at finding efficient algorithms.

Two related approaches for efficient collision detection are hierarchical models and bounding representations. A hierarchical model, such as proposed by Faverjon [21], describes an object at various levels of detail. The collision detection algorithm uses the different levels of detail to reduce the number of components that are examined. Similarly, bounding representations approximately model an object with simple primitives such as rectangles. Efficient algorithms, such as described by Baraff [3], determine if collision has occurred between the bounding representation and only then are components of the original model examined.

In this chapter, we present an efficient algorithm for distance computation between non-convex obstacles. The approach builds on the research on collision detection and convex distance computation. Objects are described as a set of convex components and we refer to this description as the underlying model. From this model we build a hierarchical bounding representation, based on spheres, that approximates the object. A search routine examines the hierarchical bounding representation of each object and determines pairs of components to compare with a convex distance algorithm. Experimental results show that only a small fraction of the possible pairs are compared, thus avoiding the $O(nm)$ complexity of the naive implementation.

To increase the efficiency of our algorithm, we propose computing the distance between objects with a *relative error*. The idea is that, for some applications, it is acceptable to partially underestimate the distance between objects. The error between the reported distance and the exact distance is limited to be a user-specified fraction of the exact distance; the magnitude of the error is reduced as the two objects approach each other and goes to zero as they touch.

By using a relative error, exact distance computation and collision detection become two extremes of the same problem. When the user specifies zero relative error, the exact distance is computed. Conversely, when the acceptable relative error approaches one hundred percent, the value returned by the distance computation becomes meaningless, but a value of zero is returned if and only if the objects intersect. Providing a continuum between these two extremes enables applications to gain the benefits of some distance information with the efficiency of collision detection.

The remainder of the chapter is divided into five sections. First, we describe the bounding representation and how it is built from the underlying model. Next, we examine the time to build the bounding representation and discuss how it can be precomputed. We then describe the search routine used to determine the distance between objects. To examine the efficiency of the algorithm, we present the results of several empirical trials. We conclude with a summary of the two main ideas of this chapter and their consequence.

6.2 The Hierarchical Bounding Representation

Before computing the distance between objects, the underlying model of the objects is used to build a hierarchical bounding representation.

The version of our algorithm described in this chapter assumes the underlying model is a surface representation consisting of a set of convex polygons. This assumption is not fundamental and extending the approach to other representations would not be difficult. Using a surface representation implies that collision will not be detected when one object completely contains another object, but such situations are avoided in many applications.

The bounding representation is based on spheres. The sphere is the simplest geometric solid; it can be specified with a position vector and a radius. To calculate the distance between two spheres requires only seven additions, three multiplications, and one square root. Other primitives, such as rectanguloids or ellipsoids, may better approximate components of the underlying model, however, we feel the simplicity of the sphere makes it the preferred bounding shape. A collision detection algorithm by del Pobil et al [18] also uses spheres to build a bounding representation.

The bounding representation consists of an approximately balanced binary tree. Each node of the tree contains a single sphere and the tree has the following two properties: the union of all the leaf spheres completely contains the surface of the object and the sphere at each node completely contains the spheres of its descendant leaf nodes.

The idea behind the bounding representation is as follows. The leaf spheres closely approximate the surface of the object. Interior nodes of the tree represent approximations of descendant leaf spheres. One can use the sphere at an interior node to determine a lower bound for the distance to any of the descendant leaf nodes, and hence to the object's surface. Nodes that are close to the root of the tree represent many leaf nodes, although to a coarse resolution. Conversely, nodes near the bottom of the tree closely approximate the shape of the few leaf spheres below them. The tree represents a hierarchical description of the object.

The first step to building the tree is to cover the object's surface with small spheres. These spheres form the leaf nodes of the tree. The underlying model of the object is a set of convex polygons; the surface is covered by covering each polygon. The covering of a convex polygon is done in a process similar to scan conversion in computer graphics [54]

A regular grid of equal sized spheres covers the polygon with the center of each sphere lying in the plane of the polygon. In addition, each leaf sphere is labeled with the polygon for which it was created; such labeling enables the search routine to determine which convex components to compare.

After covering the object with small spheres, a divide and conquer strategy is used to build the interior nodes of the tree. The set of leaf nodes is divided into two approximately equal subsets. A tree is built for each of the subsets and these are combined into a single tree by creating a new node with each of the subtrees as children. The subtrees are built by

recursively calling the same algorithm until the set is reduced to a single leaf node.

Each node has a sphere that contains all the spheres of the descendent leaf nodes and represents an approximation to these leaves. The two children of the node are intended to represent a slightly more accurate approximation of the same leaves. To maximize the improvement of the approximation, we desire to divide a set of leaf nodes into two subsets so that the bounding sphere for each subset will be small. Although we have not found an optimal method of dividing a set of leaf nodes, the process used is simple, efficient, and effective.

To divide a set of leaf node into two subsets, a bounding rectanguloid box is calculated that is aligned with the object's coordinate frame and contains the centers of all of the leaf spheres. Such a bounding box can be found by determining the minimum and maximum value for each of the three coordinates for the leaves' position vectors. Next, the algorithm selects the axes along which the bounding box is longest, and divides the leaf nodes using the average value along this axis as the discriminant. Each of the resulting two subsets should be rather compact and contain approximately equal numbers of elements.

After dividing the set into two subsets, trees for each subset are built by recursively invoking the algorithm. The two trees are combined by creating a new node with the two subtrees as children. All that remains is to determine a bounding sphere that contains all the descendant leaf spheres.

There is no obvious way to compute the smallest sphere that contains a set of spheres, so we use two heuristic methods and select the smaller of the two resulting spheres. The first method finds a bounding sphere that contains the spheres of the two children nodes and hence, by induction, all the descendant leaf nodes. The position and size of such a sphere can be determined optimally and uniquely as we are only bounding two spheres; the details are trivial and not included here. The second method directly considers the leaf spheres. A center for the bounding sphere is selected and each of the descendant leaf spheres is examined to determine the minimum radius required. The selection of the bounding sphere center is done by using the average position of the centers of the leaf spheres, which has already been calculated in the process of dividing the leaf spheres. The first method works well near the leaves of the tree, while the second methods produces better results closer to the root.

Figure 6.1 illustrates a bounding tree generated by the above algorithm. Due to the difficulty of visualizing three-dimensional objects, we show the analogous two-dimensional version. The object is a polygon approximation to the letter "g" as shown in Figure 6.1a. Figure 6.1b depicts the one hundred and thirty leaf spheres used to cover the outline of the object. The resulting bounding tree has eight levels. Figure 6.1c shows level five and Figure 6.1d shows the root of the tree.

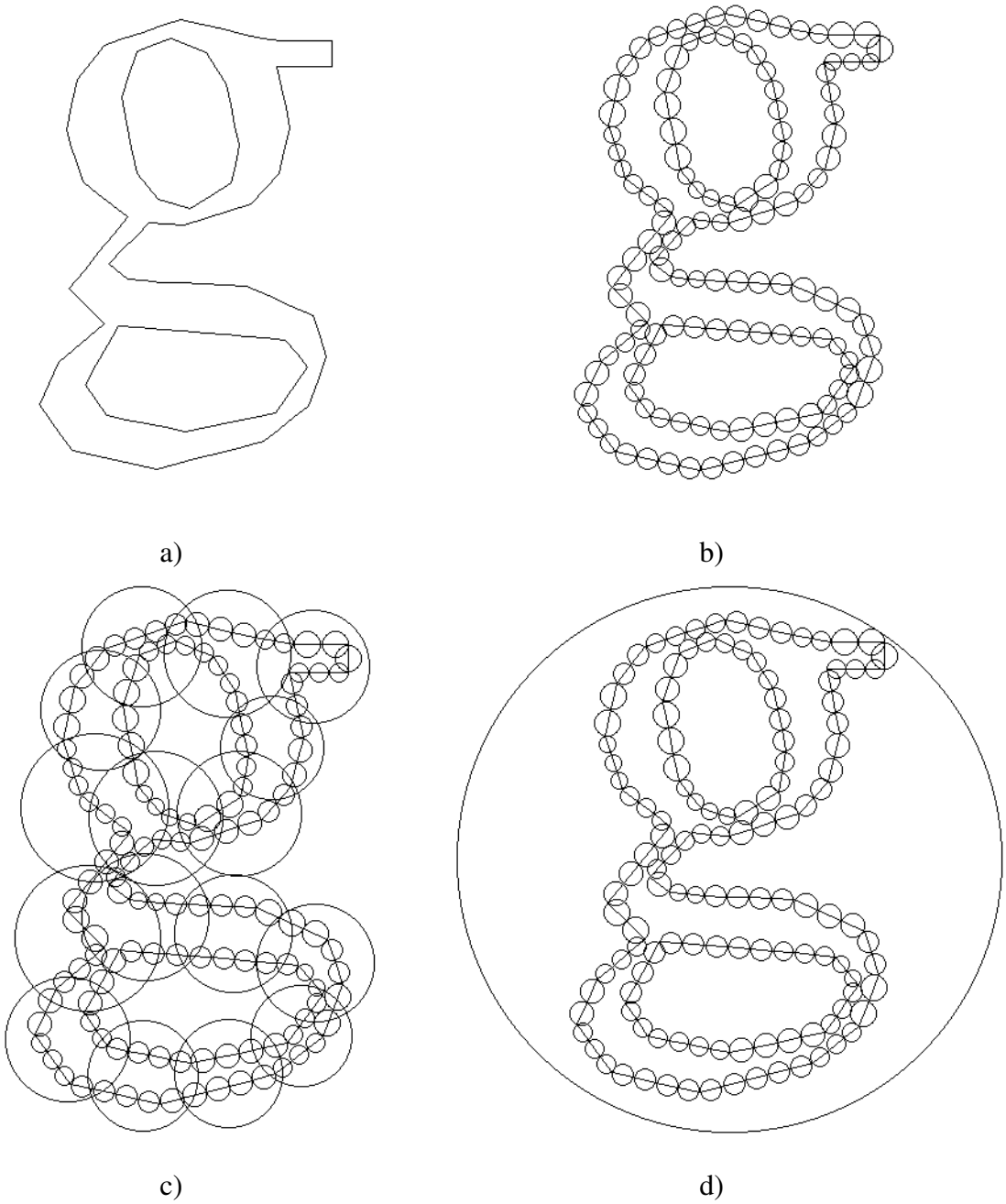


Figure 6.1: The bounding tree for an object.

6.3 Execution Time

The bounding tree is expected to be approximately balanced. If there are n leaf nodes, we expect there to be about n interior nodes and the depth of the tree to be about $\log n$. To divide a set of nodes is into two and form the bounding sphere taken time of order $O(n_i)$ complexity, where n_i is the number of leaf nodes descending from the ith node. A close look at the execution time of the above algorithm reveals an expected execution time of $O(n \log n)$, but a worst case of $O(n^2)$.

The worst case execution time can be reduced to $O(n \log n)$ if we use the median rather than the average to partition a set. The median can be found in $O(n)$ via the classic algorithm [1], or more simply, by performing an initial sort of the leaf nodes along each of the three axes and carrying out some additional bookkeeping [46]. The disadvantage of using the median is the higher constant factor in the expected execution time. A close analogy can be drawn with the relative benefits of Quicksort, which has worst case $O(n^2)$ performance, versus merge sort, which has worst case $O(n \log n)$; Quicksort is preferred for its faster expected execution time [1].

For many applications, building the bounding representation can be performed as a precomputation step. If the two objects are rigid bodies, we compute the bounding tree for each object in it local coordinate frame. Before computing the distance between the two objects at specific positions and orientations, each tree is augmented with the corresponding transformation matrix describing the positions of the object with respect to some global coordinate frame. Before a node is used in the search routine, the node's sphere is mapped through the tree's transformation matrix. The mapping is done only when a node is used because the search routine examines only a small fraction of the total nodes of a tree; mapping all the nodes before the search would result in lower efficiency.

A similar scheme can be devised if an object consists of several rigid bodies. A tree is built for each rigid body as a precomputation step. For a given configuration of the rigid bodies, we build a meta-tree with one leaf node per rigid body. The algorithm for building the meta-tree is identical to the algorithm for the individual rigid body trees. Each leaf node corresponds to the root of the tree for one rigid body and contains the transformation matrix needed to map the body from its local coordinate frame to the global frame. The bounding sphere for a leaf node is the bounding sphere of the root of the rigid body tree mapped into the global frame. In this fashion, only the construction of a relatively small tree need be done before each distance computation. The search routine traverses the tree in such a manner that any node can be mapped into the global frame when needed.

6.4 Computing the Distance between Two Objects

In this section we describe the algorithm to compute the distance between objects. As mentioned in the introduction, the goal of our algorithm is to compute the distance with

a user specified relative error; to aid the presentation we first describe a version of the algorithm that has no error.

To compute the exact distance between two objects we need to find a pair of points, one on each object, such that the distance between the points is less than or equal to the distance between any other pair. In our implementation, the object's surfaces are described as a set of convex polygons and we assume one object does not completely contain the other object; the distance between the objects is calculated by finding a pair of polygons such that the distance between the polygons is less than or equal to the distance between any other pair. The distance between polygons is computed using a convex distance algorithm.

An overview of our algorithm to compute the distance, d, between objects is as follows. We initially set d to infinity. A search routine attempts to show the objects are at least a distance d apart. Suppose the search finds two polygons from the underlying model that are less than d apart; for the initial value of d this is not difficult. If the polygons intersect, we know that the distance between the two objects is zero and we are done. Otherwise, we set d to the distance between the two polygons and continue the search with the new value of d. Eventually, the search shows that either the objects are a distance d apart or the objects intersect.

The key to the algorithm is be able to show the two objects are a distance d apart without examining all possible pairs of polygons. Since each polygon is covered by a set of leaf nodes in the bounding tree, we need only examine pairs of polygons for which a corresponding pair of leaf spheres are less than a distance d apart. Of course, if we had to examine all possible pairs of leaf spheres, we would have gained nothing, but the hierarchical structure of the bounding tree enables us to avoid this situation.

The search routine finds pairs of leaf nodes that are less than a distance d apart. The search examines pairs of nodes in a depth-first manner starting with the root nodes of the two trees. If the distance between the nodes' spheres is greater or equal to the current value of d, then from the structure of the bounding trees, we know the distance between the two sets of descendant leaf spheres is greater or equal to d and can thus be ignored. If the two nodes are less than d apart, we must further examine the children of the nodes. There are three cases to consider.

1. If both the nodes are from the interior of the tree, we decompose one of the nodes into its two children then recursively search the two pairs consisting of a child and the other node. Deciding which node to decompose is based on the heuristic of selecting the node with the larger associated sphere. If all leaf spheres are assumed to be roughly the same size, we would expect the node with the smaller sphere to more closely approximate the shape of the underlying surface; more information about the surface will be obtained by decomposing the larger sphere. Of the two subsequent recursive searches, we first examine the pair of nodes with spheres that are closer together. This heuristic aids the search in quickly lowering the value of d, reducing the number of nodes that are examined.

2. In the case of one interior node and one leaf node, the interior node is decomposed. The order of the two subsequent searches is the same as above.

3. In the case where two leaf spheres are less than the distance d apart, the underlying model must be examined. Each leaf sphere is labeled with the polygon that it covers. The distance between two polygons can be computed using one of the many available distance algorithms for convex objects. In the current implementation, we used the algorithm developed by Gilbert et al. [24]. If the distance between the two polygons is less than d, we have found a new minimum. If the distance is zero, i.e., the polygons intersect, we know the distance between the objects is zero and the search need not be continued. Otherwise, we set d to the new distance and continue the search.

Each polygon may be covered by many leaf spheres, thus it is possible that the search routine may compute the distance between the same pair of polygons multiple times. Since such repeated computations are redundant, we record which pairs of polygons have been examined, and before computing the distance between two polygons, we check whether the computation has already been performed. Due to the large number of possible pairs of polygons, we record this information using a hash table.

The search routine can be modified to include our notion of relative error. In the modified algorithm, the user specifies a relative error α. We calculate a distance d' such that $d' \leq d$ and $d - d' \leq \alpha d$. Note that d' can equal zero only if d equals zero; we will not incorrectly report collision.

To implement the modified algorithm, the search routine must guarantee that the objects are a distance d', rather than d, apart. The initial value of d' is set to infinity as in the exact case. However, when we find two polygons that are closer than d' apart, we set d' to be a fraction $1 - \alpha$ of the distance between them. After completing the search, we know the objects are at least d' apart. In addition, the true distance d between the objects is obviously less than or equal to the distance between the two polygons that were used to set d'. Hence, it can be shown that the error between d and d' meets the relative error specification.

6.5 Empirical Trials

To illustrate the performance of the distance computation algorithm, we present the results of several empirical trials. The nature of the algorithm is such that its performance depends on many factors: the shape of the objects, the underlying representation, the specified relative error, the distance between the objects, the accuracy of the bounding tree, etc. Since these factors depend on the application for which the distance algorithm is used, it is difficult to make general statements about its performance. Instead, we have chosen one scenario and examined the performance with respect to a few factors. Hopefully, these results give a reasonable impression of the behavior of the algorithm. Reported execution times are from an implementation on a DECstation 5000/240.

The scenario for which the experiments are performed is based on determining the distance between chess pieces. Six chess pieces, a king, queen, rook, bishop, knight and pawn, are randomly placed in three-dimensional space. For each chess piece, we determine its distance to the union of the other five pieces. Figure 6.2 represents a typical configuration of the chess pieces.

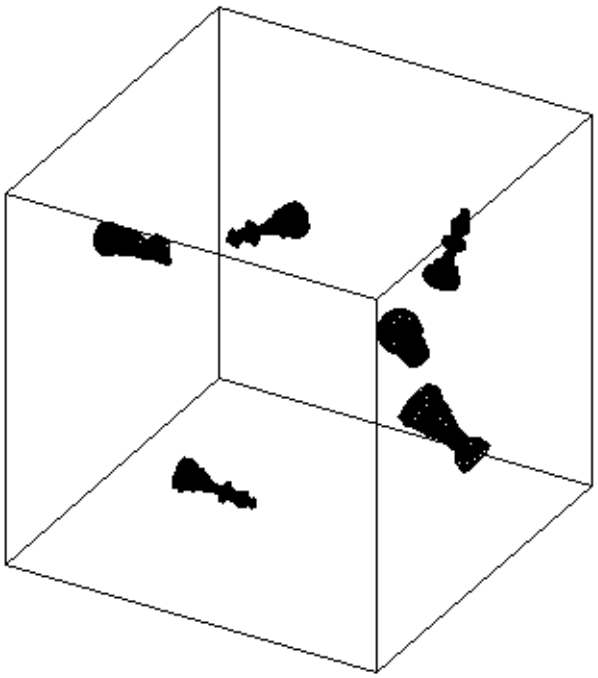


Figure 6.2: A typical configuration of the chess pieces.

The model of the pieces was designed by Randy Brown and is publicly available by ftp from wuarchive.wustl.edu. Each piece is described by a boundary representation consisting of roughly 2,000 triangles. The pieces are non-convex, have rather detailed features, and are roughly 100 units high. It is worth noting that the naive implementation for computing the distance from one chess piece to the other five pieces would examine all of the possible $2,000 \times 10,000$ pairs of triangles.

As a precomputation step, the bounding tree for each of the chess pieces was built. For the first two experiments, the radius of the leaf spheres was set to 2 units, resulting in a total of 34461 leaf nodes. A smaller leaf size would increase the number of nodes and the accuracy of the bounding tree. The result would be an increase in the search time and a decrease in the number of polygon comparisons. The net effect of such a change depends

on the situation; in this case 2 units was found to be a reasonable size. The bounding trees for all the pieces was built in 6.4 seconds.

The objects were placed with random position and orientation. The orientation was selected from a uniform distribution of all possible orientations. The position vector was selected from a uniform distribution within a cube measuring 500 units along the sides. Each point in the following graphs represent the average for one hundred random positions.

Before performing each distance computation, we built a meta-tree for the five chess pieces to be compared to the current piece. Since the tree contains only five leaf nodes, the time to build the tree was negligible.

Figure 6.3 illustrates the effect of varying the relative error. The smaller relative errors effectively correspond to computing the exact distance between objects. Even in these case, the number comparisons of both nodes and triangles is far less than the possible $20,000,000$. As the relative error is increased, the number of comparisons drops dramatically. There is approximately two orders of magnitude improvement if a 20% relative error is specified.

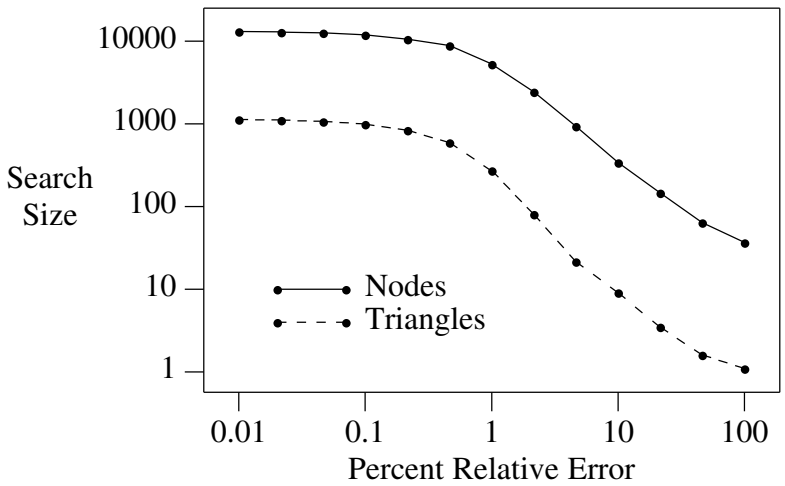


Figure 6.3: Search Size vs Relative Error. Note that both axes are log scale.

The current implementation can examine 80,000 pairs of nodes a second and compute the distance between 11,000 triangles a second. For a 20% relative error, the average execution time is 2.0 milliseconds. We speculate that such execution time would be comparable to the performance of a good collision detection algorithm, with the advantage that we obtain considerable distance information.

As two objects get closer, one would expect the search routine to examine more nodes and triangles. Figure 6.4 depicts this relationship for a relative error of 20%. As can be seen, the algorithm runs a lot slower when the objects are close. The effect on the average time to perform a distance computation depends on the distribution of distances for a given

application. Note that the far left points correspond to configurations where a chess piece intersected with one of the other five pieces.

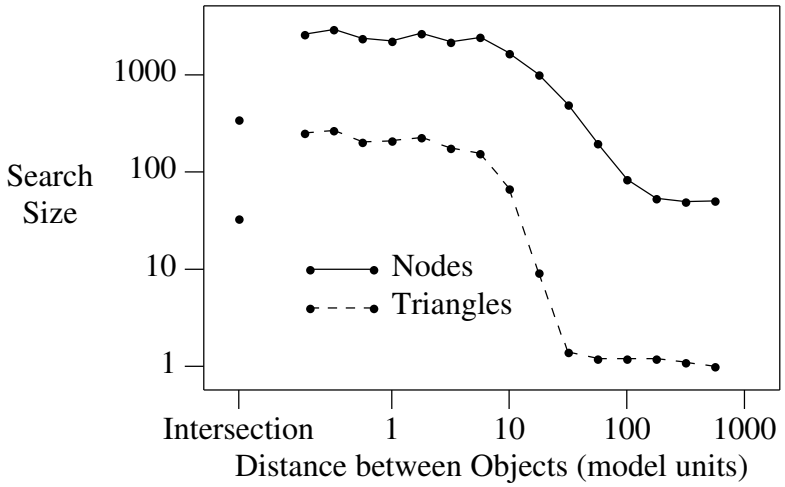


Figure 6.4: Search Size vs. Distance between Objects. Note that both axes are log scale.

One feature of the second graph is the drastic decrease in the number of triangle distance computations as the distance increases past about 20 units. Since the leaf spheres have a radius of 2 units, the error between two bounding trees is about 4 units. With a relative error of 20%, one would expect the bounding representation would enable the search routine to eliminate almost all pairs of triangles when the objects are more than 20 units apart.

Figure 6.5 examines the effect of increasing the amount of detail used to describe each of the chess pieces. Since we have only one polygonal model of the pieces, we perform this experiment by assuming the leaf spheres of the bounding representation exactly describe the object. We use the underlying model only to build the bounding trees; it is not used during the search routine. By varying the radius of the leaf spheres, we can vary the detail of the description. As can be seen, the number of nodes searched appears to be proportional to the log of the number of leaf spheres. This is very encouraging; a more detailed model of the object can be used with very little affect on the execution time. These results were obtained with a relative error of 20%. Unfortunately, the same relationship does not hold for very small relative errors.

6.6 Conclusion

An efficient distance algorithm for non-convex objects can be built by combining a hierarchical bounding representation, a simple search routine.

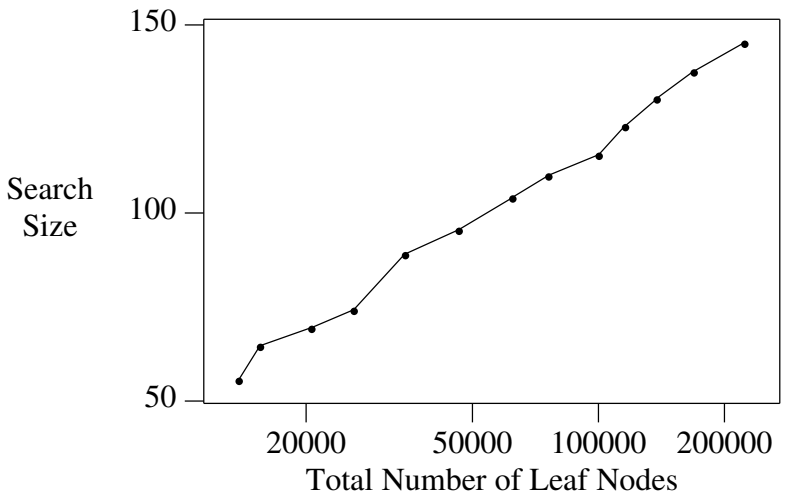


Figure 6.5: Search Size vs. Number of Leaf Nodes. Note that the horizontal-axis is log scale.

The notion of relative error enables exact distance computation and collision detection to be unified as two extremes of a single problem. By accepting a reasonable relative error, the performance of a collision detection algorithm can be achieved while still obtaining useful distance information.

Efficient distance computation opens the possibility of modifying applications that currently employ collision detection. Since distance computation provides information about how close objects are to collision, it is possible to reason rigorously about the collision free motion of objects. In contrast, reasoning about collision-free motion is difficult if we consider only the question of whether or not a given configuration of the objects is not in collision, since any motion of objects may bring them into contact.

Chapter 7

System Integration

So far, we have described how one may efficiently modify a collision-free path for a robot. In this Chapter, we turn our attention to integrating this capability into a complete robotic system. Several issues need to be addressed:

- In Chapter 5, a scheme for constructing a collision-free path from the representation of an elastic band was described. The resulting path consisted of a sequence of line segments through the bubbles associated with the representations. Unfortunately, such a path is not ideal from a control perspective; the transition from one line segment to the next involves a discontinuous change in direction. An alternative scheme for constructing a path is described that avoids this problem. The scheme is based on B-splines, and results in a path with C^2 continuity.

- A path must be converted to a trajectory before a control system can move a robot. A path specifies only where a robot should move, while a trajectory specifies a time history for the motion. Using a trajectory, a feedback controller can determine where the robot should be at any given instant, and then apply actuator effort to achieve this goal. Many schemes have been used for generating a trajectory from a path; Craig [17] gives a good introduction. Below, we present a novel real-time algorithm that can construct trajectories that minimize the time to reach the goal.

- Separating the tasks of modifying a path and constructing a trajectory can cause problems. If the robot is moving along a path that is then modified, it may not be possible to construct a trajectory that the robot can follow. This problem occurs because the robot has constraints on its acceleration. We present a few ideas for overcoming this problem.

Finally, the chapter concludes with a description of a robotic system we have built that incorporates the ideas presented in this thesis. The system consists of three Puma 560 robots operating in a shared environment. Paths may be specified for each arm, and these paths are modified to account for the position of the other arms and changes in the environment.

7.1 Constructing a Smooth Path

In Chapter 5, an implementation of elastic bands was described in which an elastic is represented by a finite sequence of particles. To ensure that a collision-free path can be constructed between successive particles, the bubbles associated with the current configuration of neighboring particles are required to overlap. As illustrated in that chapter, a path can be constructed through the bubbles using line segments and configurations that are shared between adjacent bubbles.

The construction of a collision-free path from the representation of the elastic band need not be limited to a sequence of line segments. As long as the path remains inside the bubbles, it will be collision-free. The disadvantage of line segments is that the robot will have difficultly moving along the path when a corner is reached as the direction of the path changes discontinuously. By selecting more complex curves, these sharp corners can be eliminated, thus improving the path from a control point of view. In this section we describe the construction of such a smooth path.

The difficulty with constructing a smooth path from an elastic band is ensuring that the path is completely collision-free. To achieve this goal, we specify the path using a B-spline [22, 5, 20], and utilize the convex hull property of this representation. The convex hull property enables us to bound the region in which the curve will lie. If this region is contained in the bubbles associated with the elastic band, the path is collision-free. This approach is similar to the work of Kant and Zucker [29] where B-splines are used to construct smooth paths from a cell decomposition of the free-space.

Before describing the details of the path construction, it is desirable to change slightly the constraint imposed on the bubbles associated with the representation of an elastic band. Previously, we have stated that the two bubbles associated with consecutive particles must overlap. To facilitate the construction of smooth paths, we strengthen this constraint and require that the bubbles overlap along the line segment joining the location of the two particles. Figure 7.1 illustrates examples where this condition is and is not met. Note, for some shapes of the bubbles, such as hyperspheres, the new constraint is equivalent to requiring the bubbles simply overlap.

A cubic B-spline approximates a series of $m+1$ control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_m$ by a curve composed of $m-2$ cubic polynomial segments $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{m-2}$. The ith segment is specified by the control points $\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}, \mathbf{p}_{i+2}$, and is given by the parametric equation

$$\mathbf{c}(s) = \frac{1}{6}[(1-s)^3\mathbf{p}_{i-1} + (3s^3 - 6s^2 + 4)\mathbf{p}_i + (-3s^3 + 3s^2 + 3s + 1)\mathbf{p}_{i+1} + s^3\mathbf{p}_{i+2}],$$

where $s \in [0, 1]$. The end of each segment corresponds to the start of the succeeding segment, and the place where this transition occurs is referred to as a *knot* point.

Some of the important properties of B-splines include:

- Each polynomial segment is contained in the convex hull of the associated four control points.

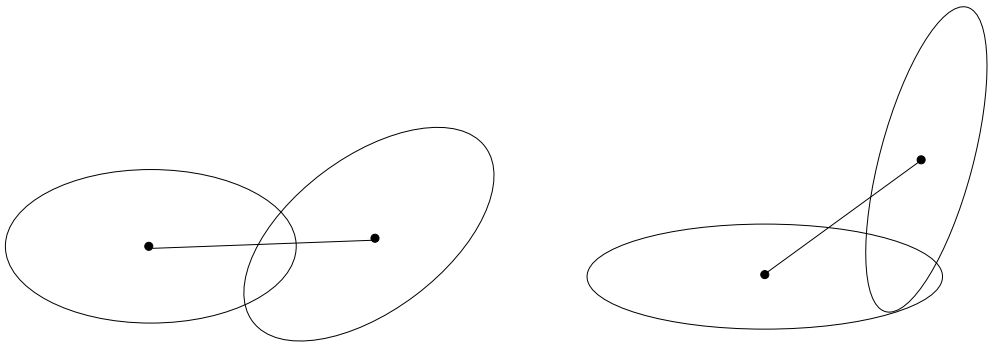


Figure 7.1: An illustration of the new constraint on the bubbles of an elastic band. **a)** The new constraint is met. **b)** A counter-example. The bubbles do not overlap along the line segment joining the two configurations from which the bubbles were created.

- The first and second derivatives of a cubic B-spline are continuous, i.e., the curve has C^2 continuity.
- The curve does not, in general, interpolate any of the control points. However, a suitable selection of control points can be made to cause the curve to interpolate specific points.

Figure 7.2 shows a B-spline curve defined by seven control points. Note, the curve interpolates control points $\mathbf{p}_1$ and $\mathbf{p}_5$ due to the special selection of the pair $\mathbf{p}_0$ and $\mathbf{p}_2$, and the pair $\mathbf{p}_4$ and $\mathbf{p}_6$ respectively.

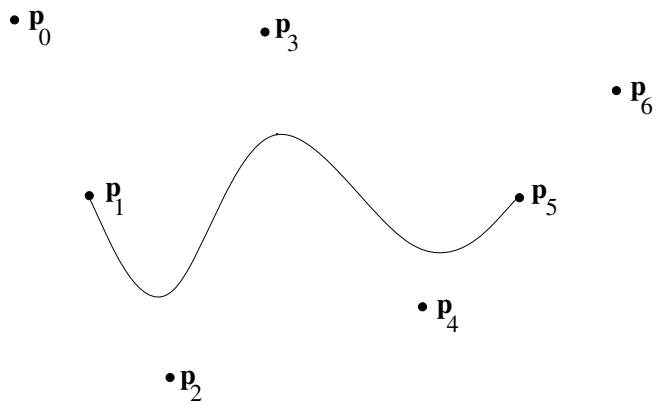


Figure 7.2: An example B-spline defined by seven control points.

We now describe the procedure for constructing a smooth collision-free path from the representation of an elastic band. The description is in the form of a worked example for

an elastic band represented by four particles; the extension to more than four particles is not difficult.

Consider the elastic band shown in Figure 7.3a. The elastic band is composed of four particles at configurations $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, and $\mathbf{q}_4$. Let us assume the bubbles associated with the robot are circles and the bubbles around the given configurations are as shown.

As required, the bubbles overlap along the line segments connecting adjacent particles. Along the line segments $\mathbf{q}_1\mathbf{q}_2$, the configuration $\mathbf{m}_1$ is selected so that it lies in the intersection of $\beta_{\mathbf{q}_1}$ and $\beta_{\mathbf{q}_2}$ as shown in Figure 7.3b. Similarly, configurations $\mathbf{m}_2$ and $\mathbf{m}_3$ are selected along the line segments $\mathbf{q}_2\mathbf{q}_3$ and $\mathbf{q}_3\mathbf{q}_4$ respectively. These three configurations act as the transition points from one bubble to the next, and will be included as control points in the B-spline path. The exact equation used to select these points depends on the shape of the bubbles for a given robot.

Next, for each bubble $\beta_{\mathbf{q}_i}$, a set of configurations $\mathbf{p}_{i.j}$ are selected using the equations given below (see Fig. 7.3c). These configurations along with the transition points $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$ form the set of the control points for the B-spline path. There are many possible choices for these configurations; the following selection was found to give satisfactory results in terms of smoothness.

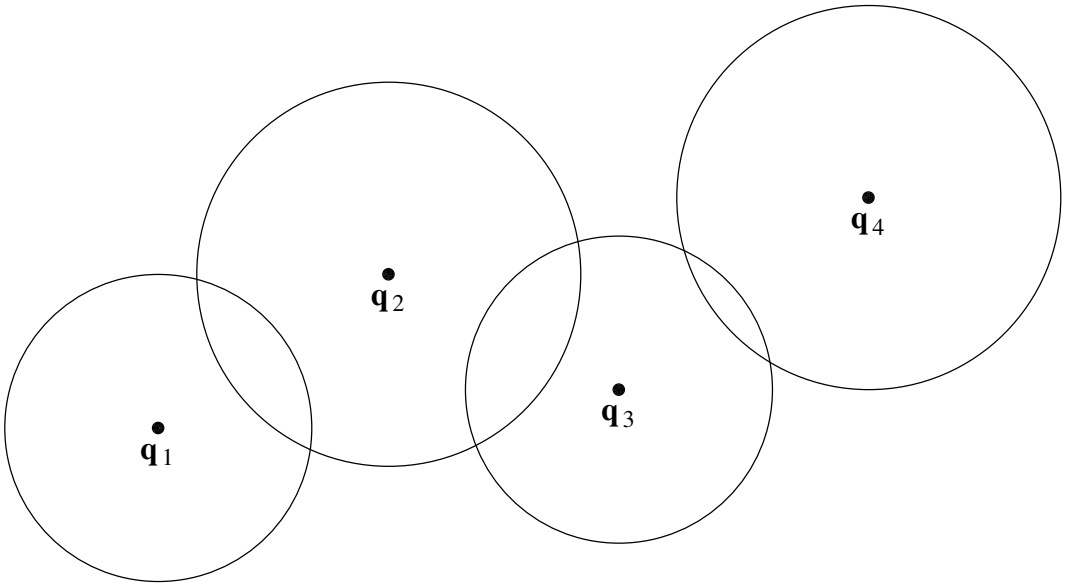
For the first bubble, the points $\mathbf{p}_{1.j}$ are selected so that the B-spline will start at $\mathbf{q}_1$ and progress in the direction towards $\mathbf{m}_1$. The points are defined by

$$\begin{aligned}
\mathbf{p}_{1.1} &= \mathbf{q}_1 - \frac{1}{3}(\mathbf{m}_1 - \mathbf{q}_1), \\
\mathbf{p}_{1.2} &= \mathbf{q}_1, \\
\mathbf{p}_{1.3} &= \mathbf{q}_1 + \frac{1}{3}(\mathbf{m}_1 - \mathbf{q}_1), \\
\mathbf{p}_{1.4} &= \mathbf{q}_1 + \frac{2}{3}(\mathbf{m}_1 - \mathbf{q}_1).
\end{aligned}$$

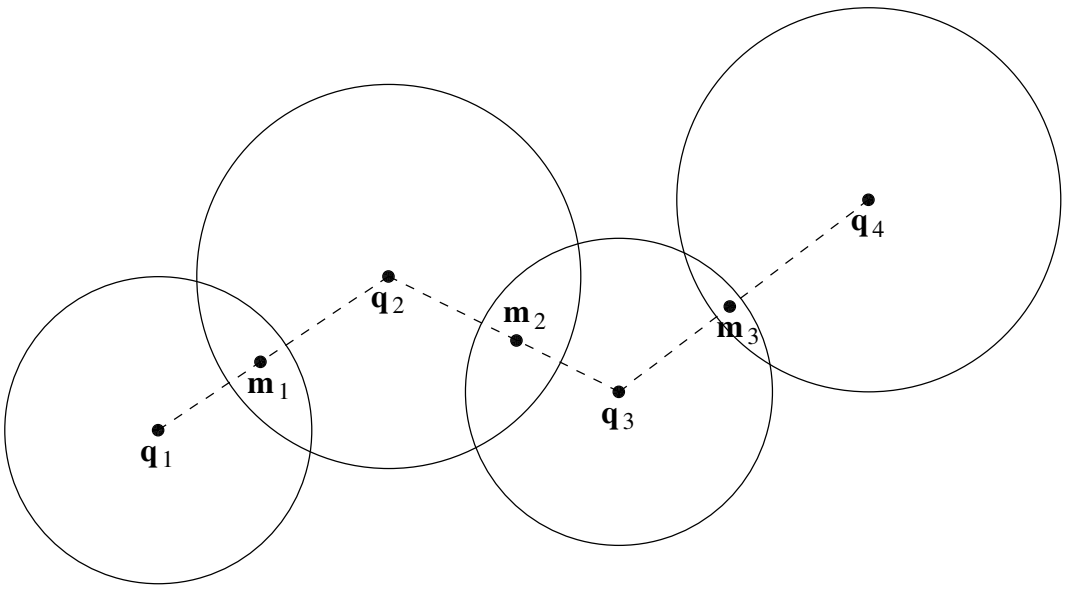
For the second and third bubble, the points $\mathbf{p}_{i.j}$ are defined by

$$\begin{aligned}
\mathbf{p}_{i.1} &= \mathbf{q}_i + \frac{2}{3}(\mathbf{m}_{i-1} - \mathbf{q}_i), \\
\mathbf{p}_{i.2} &= \mathbf{q}_i + \frac{1}{3}(\mathbf{m}_{i-1} - \mathbf{q}_i), \\
\mathbf{p}_{i.3} &= \mathbf{q}_i + \frac{1}{4}(\mathbf{m}_{i-1} - \mathbf{q}_i) + \frac{1}{4}(\mathbf{m}_i - \mathbf{q}_i), \\
\mathbf{p}_{i.4} &= \mathbf{q}_i + \frac{1}{3}(\mathbf{m}_i - \mathbf{q}_i), \\
\mathbf{p}_{i.5} &= \mathbf{q}_i + \frac{2}{3}(\mathbf{m}_i - \mathbf{q}_i).
\end{aligned}$$

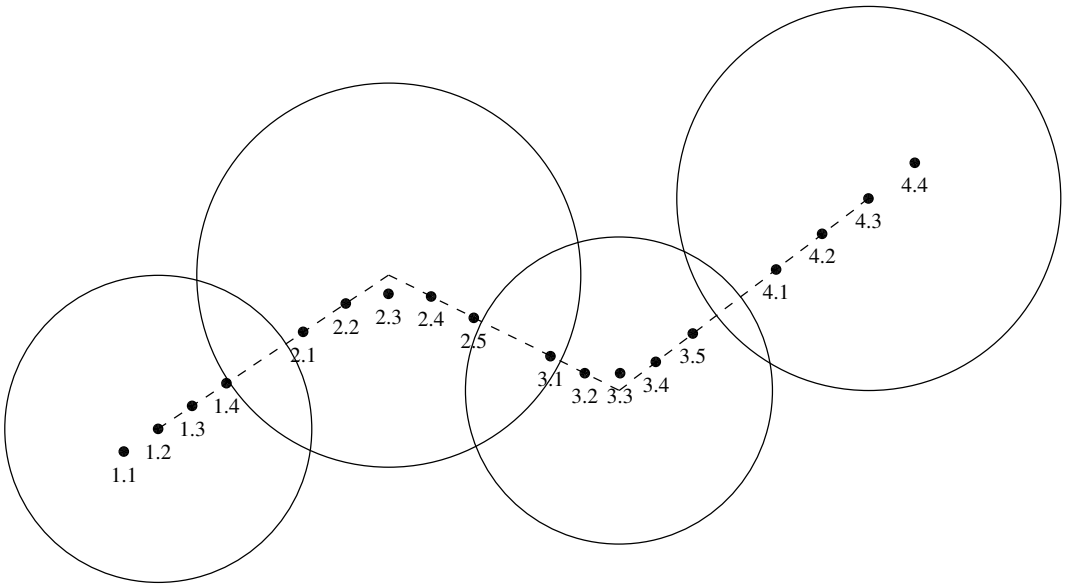
These points approximate a reasonably smooth path through the bubbles. If there were more than four particles in the elastic, additional sets of the above points would be selected for each of the extra intermediate bubbles.



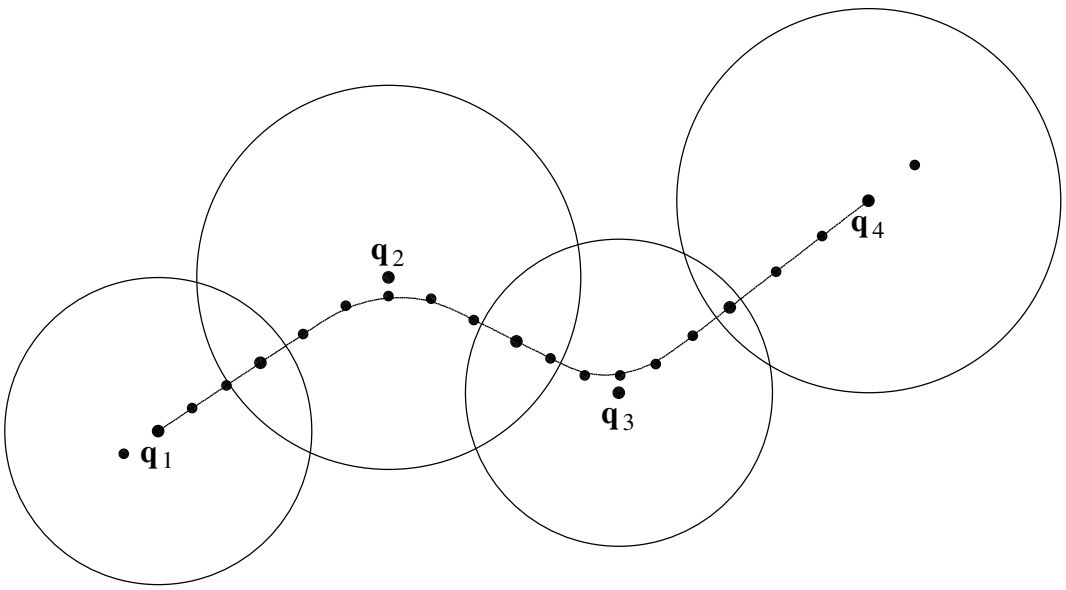
a)



b)



c)



d)

Figure 7.3: The construction of a B-spline path for an elastic band represented by four particles. **a)** The four particles of the elastic and the associated bubbles. **b)** Selecting the transition points between successive bubbles. **c)** The additional configuration used to define the B-spline. **d)** The final curve.

For the last bubble, the points $\mathbf{p}_{4.j}$ are selected so that the B-spline will end at $\mathbf{q}_4$. The points are defined by

$$\begin{aligned}
\mathbf{p}_{4.1} &= \mathbf{q}_4 + \frac{2}{3}(\mathbf{m}_3 - \mathbf{q}_1), \\
\mathbf{p}_{4.2} &= \mathbf{q}_4 + \frac{1}{3}(\mathbf{m}_3 - \mathbf{q}_1), \\
\mathbf{p}_{4.3} &= \mathbf{q}_4, \\
\mathbf{p}_{4.4} &= \mathbf{q}_4 - \frac{1}{3}(\mathbf{m}_3 - \mathbf{q}_1).
\end{aligned}$$

The B-spline path is specified by the control points

$$\mathbf{p}_{1.1}, \ldots, \mathbf{p}_{1.4}, \mathbf{m}_1, \mathbf{p}_{2.1}, \ldots, \mathbf{p}_{2.5}, \mathbf{m}_2, \mathbf{p}_{3.1}, \ldots, \mathbf{p}_{3.5}, \mathbf{m}_3, \mathbf{p}_{4.1}, \ldots, \mathbf{p}_{4.4}.$$

The resulting spline is shown in Figure 7.3d. This spline can be shown to lie within the bubbles associated with the elastic. To see this, first notice that the control points $\mathbf{m}_{i-1}, \mathbf{p}_{i.j}, \mathbf{m}_i$ lie within the ith bubbles. From the convex hull property, any of the segments of the spline generated by points within a single bubble will be collision-free. For segments that pass from one bubble to another, notice that the associated control points fall on the line segments between the particles. By the convex hull property, the B-spline segments will also fall on these line segments, and thus are collision-free.

7.2 Time Parameterization

To move the robot along a path, a time parameterization is needed to convert the path into a trajectory. A time-parameterization corresponds to associating a desired time with each point on the path. An equivalent alternative to a time-parameterization is to specify a desired speed for each point along the path.

A time-parameterization must respect the capabilities of the given robot. In particular, all robots have constraints on the maximum rate of acceleration that they can achieve. Such acceleration constraints may vary with the configuration of the robot and may be velocity dependent. A robot may have other constraints, such as a maximum velocity, but these can typically be expressed in terms of acceleration constraints. For example, a velocity constraint can be represented by specifying a zero maximum acceleration when the robot is moving at the maximum velocity.

The acceleration constraints provide an upper bound on the possible speed at each point along a path. In addition, the constraint at one point on the path may propagate; for example, a robot must begin to slow down before entering a sharp bend in a path, otherwise the speed may be too large to remain on the path.

Apart from satisfying the acceleration constraints, we may desire other criteria. A common criterion is that the robot minimizes the time required to move along the path; this is the time-optimal time parameterization problem.

Finding time-optimal parameterizations has been well studied; two early analyses of the problem are given by Bobrow et al. [8] and Shin and McKay [56]. This section describes an original algorithm, based on their work, that enables a discrete approximation to the optimal-time parameterization to be determined incrementally and in real time.

Overview of the problem

The time parameterization problem can be specified more formally as follows. Assume we have a C^2 continuous parameterization of the the path for the robot given by $\mathbf{c}(s)$. We must compute s as a function of time, that is find a function $s(t)$. From this we can determine the velocity and acceleration of the robot by

$$\dot{\mathbf{c}} = \frac{d\mathbf{c}}{dt} = \frac{d\mathbf{c}}{ds}\dot{s}$$

$$\ddot{\mathbf{c}} = \frac{d^2\mathbf{c}}{dt^2} = \frac{d\mathbf{c}}{ds}\ddot{s} + \frac{d^2\mathbf{c}}{ds^2}\dot{s}^2$$

Suppose the robot has acceleration constraints of the form

$$a(s, \dot{s}) \leq \ddot{\mathbf{c}} \leq b(s, \dot{s}),$$

where a and b represent functions that return the maximum possible deceleration and acceleration respectively. These functions are dependent on the configuration of the robot and its velocity, both of which can be determined for a given path by knowing s and $\dot{s}$. It is possible to rewrite these conditions on the acceleration in terms of conditions on $\ddot{s}$, i.e., we determine functions $f(s, \dot{s})$ and $g(s, \dot{s})$ such that

$$f(s, \dot{s}) \leq \ddot{s} \leq g(s, \dot{s}). \tag{7.1}$$

Bobrow et al. give details of this transformation.

The set of values of s and $\dot{s}$ for which $f(s, \dot{s}) \leq g(s, \dot{s})$ is called the *admissible region*. Values of s and $\dot{s}$ in this region correspond to states for which it is possible for the robot to track the path. If the robot ever enters a state which is not in this region, there is no valid acceleration of the robot that will maintain it on the path; leaving the admissible region must be avoided.

The basic idea of the time-optimal solution is to choose the acceleration $\ddot{s}$ to make the velocity $\dot{s}$ as large as possible, while maintaining the constraint given by equation (7.1). Bobrow et al. [8] have shown that the minimum time trajectory has the property that for each point on the trajectory either $\ddot{s} = g(s, \dot{s})$ or $\ddot{s} = f(s, \dot{s})$. These ordinary differential equations (ODEs) correspond to the robot accelerating and decelerating along the path as quickly as possible. Finding the optimal parameterization corresponds to finding the

switching points, the positions along the path at which $\ddot{s}$ switches between its maximum and minimum value.

Consider the B-spline path shown in Figure 7.4 for joints two and three for a Puma 560 manipulator. The path is defined by the seven control points $\mathbf{p}_0, \ldots, \mathbf{p}_6$ and the resulting B-spline is defined for values of $s \in [0, 4]$.

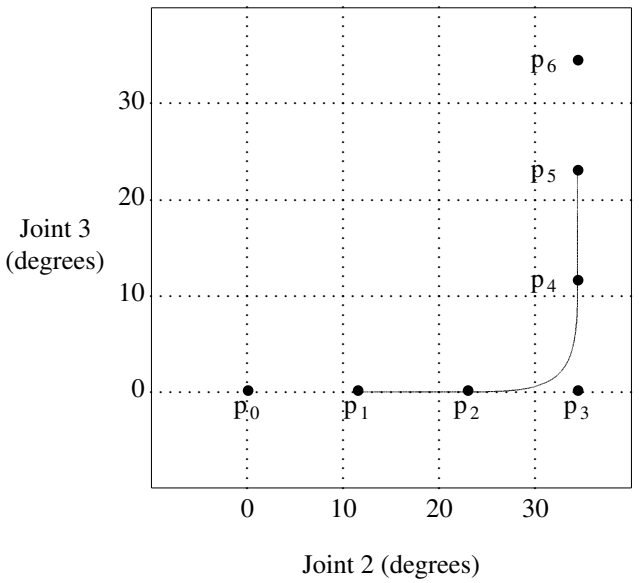


Figure 7.4: A path for two links of a Puma 560.

The complete state of the robot is given by s and $\dot{s}$, the position and speed along the path respectively. As a consequence, a useful way to visualize the time-optimal solution is to examine the s–$\dot{s}$ phase-plane.

Using the phase plane, Figure 7.5 depicts the time-optimal solution for the path in Figure 7.4, calculated from a conservative model of the Puma 560's actuator capabilities. The trajectory is given by the lower line in the figure, while the the upper line indicates the boundary of the admissible region. As expected, the trajectory starts by accelerating along the initial straight section of the path. The acceleration is at the maximum rate possible given by $\ddot{s} = g(s, \dot{s})$. The first switching point occurs when $s \approx 1$; the robot decelerates at the maximum rate of $\ddot{s} = f(s, \dot{s})$ as it moves into the corner. The next switching point is reached when $s = 2$ as the robot accelerates out of the corner. Notice that this switching point is on the boundary of the admissible region, a characteristic of time-optimal parameterizations [56]. Finally, a third switch occurs when $s \approx 3$, enabling the robot to decelerate to rest at the end of the path.

Early algorithms for computing the time-optimal trajectory along a path were off-line in nature [8, 56]; these algorithm required extensive numerical computation and emphasis

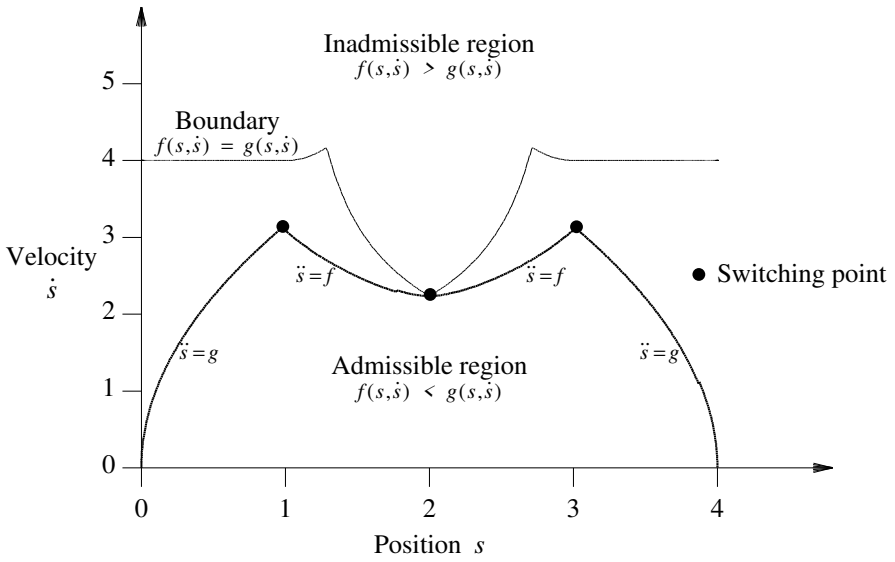


Figure 7.5: The time-optimal parameterization of the path in Figure 7.4 using a conservative model of the Puma 560's actuator capacity. The trajectory has three switching points.

was placed on accurately finding the true optimal trajectory. More recent algorithms [45] sacrifice accuracy to decrease the total time needed to find a solution. These algorithms attempt to reduce the cost of finding near-optimal solutions to the point where it is feasible to use such algorithms in on-line situations; in this way, time-optimal trajectories can replace the less sophisticated trapezoid trajectories that are commonly used in industry [17].

A real-time algorithm

The algorithm we have developed finds an approximation to the time-optimal trajectory. The major advantage of our algorithm is that it can be performed incrementally along the path; the time parameterization for the first Δt seconds can be computed without determining the time parameterization along the remainder of the path. By ensuring that each iteration of the algorithm takes, at most, a constant amount of time, the time parameterization can be overlapped with the motion of the robot. Such a scheme effectively reduces the time-cost of the of the time-parameterization to zero. In addition, for elastic bands, the path the robot is following may change; an incremental algorithm enables us to avoid wasting effort computing the time parameterization of an entire path when only a small section may be used.

The basic idea of the algorithm is to restrict the possible switching points to moments in time that are integer multiples of some sampling interval Δt. Between each of these possible switching points, the robot is either accelerating or decelerating at the maximum

possible rate. Figure 7.6 illustrates this idea for the path in Figure 7.4. The sample time Δt was set to 0.1 seconds. Each of the possible switching points is shown by a bullet. As can be seen, the resulting trajectory is a reasonable approximation to the optimal trajectory, which is also indicated in figure. Note that during final segment of the trajectory, the robot is not decelerating at the maximum possible rate. As explained in a following section, this special segment is needed to ensure the robot comes to rest at the end of the path.

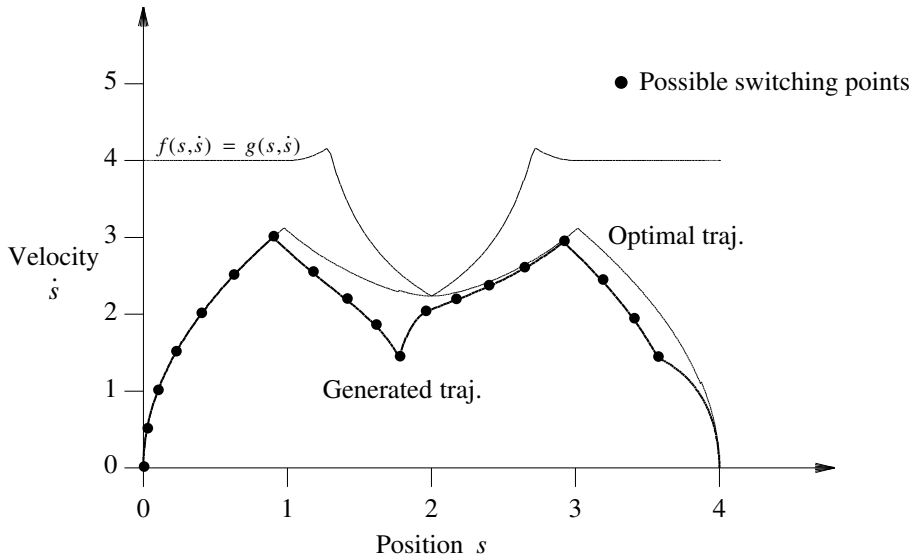


Figure 7.6: The time-parameterization generated when the possible switching points are restricted to occur at multiples of Δt seconds. In this figure $\Delta t = 0.1$ seconds. For reference, the time-optimal trajectory is also shown.

Each iteration of the algorithm generates a time-parameterization for the next Δt seconds along the path. Given the restriction on the possible switching points, generating the time-parameterization corresponds to determining if the robot should accelerate or decelerate from the current state. Accelerating will reduce the time to reach the goal configuration, but may not be feasible. If accelerating is not feasible, the robot should decelerate.

A robot may not be able to accelerate for Δt along a path, given its current position and velocity, for two reasons:

1. The acceleration moves the robot out of the admissible region.

2. The new state of the robot, after accelerating, is such that no feasible trajectory exits for the remainder of the path.

The first condition can be checked by integrating the ODE $\ddot{s} = g(s, \dot{s})$ from the current state for Δt seconds. If the integration causes the state of the robot to leave the admissible region,

accelerating the robot is not feasible. The second condition is a little more difficult. Under suitable assumptions, however, the new state of the robot can be checked by determining if it is possible to decelerate to rest along the path.

Let us assume that the boundary of the admissible region is a single valued function in the s–$\dot{s}$ plane. In other words, for each point on the path, there exists a speed such that any lower speed is valid and any greater speed is invalid. This assumption seems reasonable, but Shin and McKay [56] have shown that this property does not hold in the case where a friction model is included in the acceleration constraints. Such situations are difficult to handle correctly, but we may reduce the admissible region so the above property does hold at the cost that suboptimal trajectories will be generated; see the mentioned paper for more details.

Suppose the final velocity of the robot at the end of the path is zero; this is the case for many applications. We claim, without proof, that a feasible trajectory exits if and only if the robot can come to rest along the path by decelerating at the maximum rate $f(s, \dot{s})$. The intuition for this claim is as follows. If the robot can come to rest, it can continue to the end of the path at some infinitesimally slow speed. If it cannot come to rest, the deceleration trajectory must cross into the inadmissible region. Any other trajectory will be above the deceleration trajectory and hence will also cross into the inadmissible region.

We now describe an iteration of our time-parameterization algorithm. The ODE $\ddot{s} = g(s, \dot{s})$ is integrated from the current state for Δt seconds, recording the trajectory in phase space. If the integration succeeds, i.e., does not leave the admissible region, the ODE $\ddot{s} = f(s, \dot{s})$ is integrated from the new state of the robot. If this second integration reaches a state where $\dot{s} = 0$ without leaving the admissible region, the robot can safely come to rest along the path from the new state. The acceleration is deemed valid, and the recorded trajectory is returned as the time-parameterization for the next Δt. If either of the above integrations fails, the robot decelerates; the ODE $\ddot{s} = f(s, \dot{s})$ is integrated from the current state of the robot for Δt seconds and the resulting trajectory is returned. Figure 7.7 attempts to illustrate this process for the trajectory generated in 7.6 by showing all the integrations that were performed.

One can view the algorithm as "filling-in" the region enclosed by the time-optimal trajectory. This filling-in process proceeds from left to right in discrete segments. The advantage of the algorithm is that only one segment needs to be generated in order to determine the time-parameterization for a Δt time step.

Real-time performance

In order to overlap the generation of the trajectory with the motion of the robot, the execution time for one iteration of the algorithm must be less than Δt; it must operate in real time. Our current implementation, which has not been optimized, can achieve this goal for $\Delta t = 0.1$ seconds on a 10 MIPs processor.

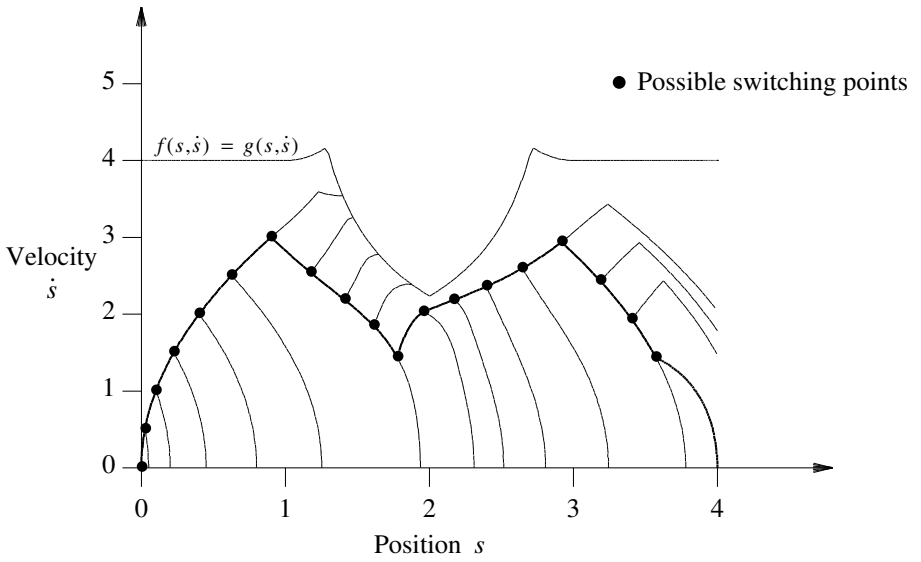


Figure 7.7: ODE integrations used to generate the trajectory

One problem with achieving real-time performance is that checking that the robot can come to rest from a given state can take an arbitrarily amount of time. For example, for a robot with constant acceleration on a straight path, one expects the robot to accelerate until the middle of the path, then decelerate the remainder of the way. When the robot reaches the middle, the ODE integration that checks the robot can stop will cover half the length of the path.

To bound the execution time of the ODE integrations, we place a limit on the number of iterations performed by the numerical integrator. This bound has the effect of limiting how far forward the deceleration check will examine, thus potentially producing sub-optimal trajectories. In practice, it is rare that a robot cannot stop in a second or two along any given path; a reasonable integration bound will have negligible effect on the trajectory.

End conditions

As described so far, the time-parameterization algorithm will not, in general, be able to generate a trajectory that brings the robot to rest at the end of the path. Consider a situation in which a robot is near the end of the path. The velocity of the robot may be such that if it decelerates at the maximum rate it will come to rest before reaching the path's end. Alternatively, if the robot accelerates for Δt seconds, the velocity may be too great to stop before overshooting the end. The time-optimal solution to this problem is to accelerate for some small amount of time, then decelerate at the maximum rate; this implies, however, that the final switching point is not a integer multiple of Δt. Another solution, although not

optimal, is to decelerate at a rate that is not maximal; this is the solution we use.

During each iteration of the time-parameterization algorithm, several trajectories are considered that take the robot from the current state to the final rest state at the end of the path. These trajectories are constructed by computing a cubic Bèzier curve between these two states. The difference between the trajectories is the time-period that the curve is defined over; in the current implementation we consider trajectories with duration $\Delta t, 2\Delta t, 3\Delta t, 4\Delta t$. When the robot is far from the end of the path, each of these trajectories would require very large accelerations from the robot. As the robot approaches the end configuration, the magnitude of these accelerations will decrease, until a valid trajectory is found. This trajectory is then used to end the motion of the robot.

Improving the trajectories

The trajectory shown in Figure 7.6 only roughly approximates the true time-optimal parameterization. The approximation can be improved in two ways: increasing the number of possible switching points, and adding additional acceleration options for each interval.

The number of possible switching points can be increased by reducing Δt. In this way, the algorithm can find switching points that are closer to those of the time-optimal trajectory. Figure 7.8 illustrates the effect of reducing Δt to 0.05 seconds. As described above, the computational cost of each iteration of the algorithm can be made constant, however, decreasing Δt increases the number of iterations; halving Δt will double the total execution time. In situations where the generation of the trajectory is overlapped with the motion of the robot, Δt must be greater than the worst case time for one iteration.

Although Bobrow et al. [8] have shown that the time-optimal trajectory consists of only periods of maximal acceleration and deceleration, additional acceleration options can improve our approximation. For example, Figure 7.9 illustrates a trajectory in which the option to maintain a constant speed has been added. If it is not feasible for the robot to accelerate at the maximum rate, the algorithm tries to maintain the robot's current speed; if this option is also not feasible, the robot decelerates. The feasibility of maintaining the current speed can be checked by a procedure that is similar to the one used for the accelerating case. Comparing this trajectory with the one shown in Figure 7.8, we see that at $s \approx 1$, $s \approx 3$, and $s \approx 3.6$, the algorithm selects this new option instead of decelerating. These changes result in a better approximation to time-optimal trajectory.

7.3 Combining Path Modification and Trajectory Generation

In this section, some ideas are presented for combining path modification and trajectory generation. The simplest method is to allow a path to deform until a static configuration is

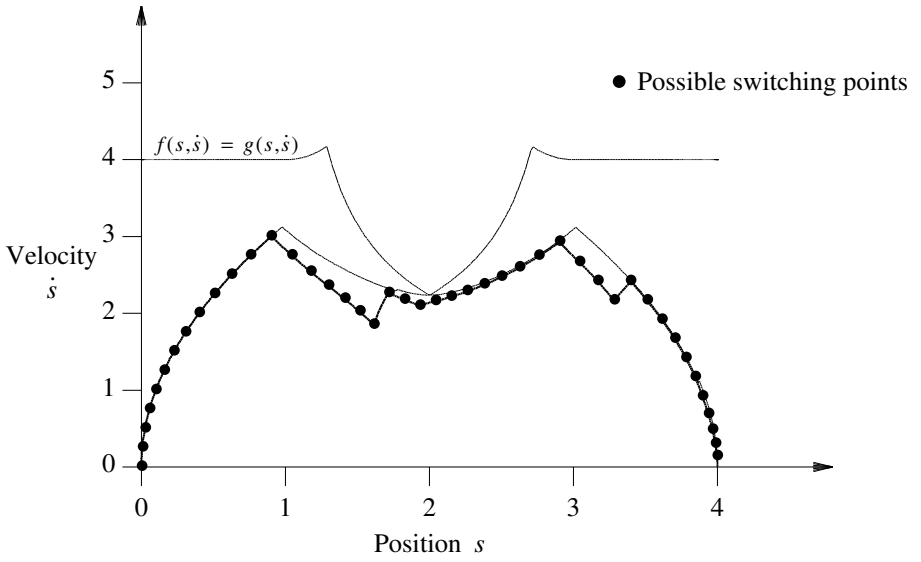


Figure 7.8: The time-parameterization generated when $\Delta t = 0.05$sec.

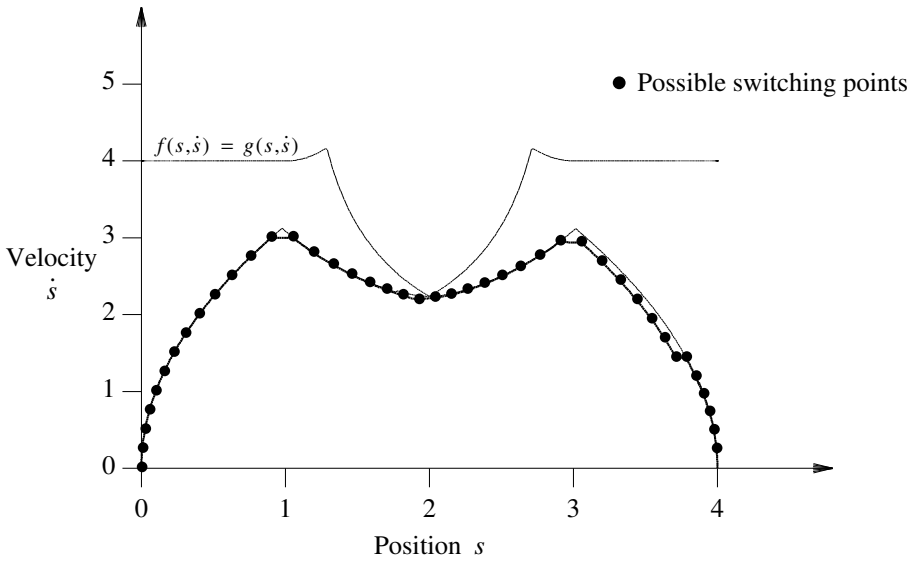


Figure 7.9: Adding the option of maintaining a constant speed. $\Delta t = 0.05$.

reached, convert this path to a trajectory, and move the robot along the trajectory using a control system. Such a scheme is useful as a pre-motion optimization of a path, however, sensor data acquired during the motion of the robot would not influence its behavior.

Allowing the path to deform during the motion of the robot enables sensor data to be incorporated, providing a form of reactive behavior. If the path is modified, a new trajectory must be computed, however, this may not always be possible. The problem arises because the robot has a non-zero velocity. This initial velocity may make it infeasible for the robot to follow the modified path, especially when the trajectory along the original path was time-optimal.

One solution to this problem is to freeze an initial section of the path and modify only the subsequent portion. To see how this can work, consider a robot moving along a path using a valid time parameterization. If the robot was to decelerate at the maximum possible rate, it can come to rest at some point along the path. If modifications are made only to the path after this point, a valid trajectory can always be generated for the new path.

Although the above scheme will work, it may not be satisfactory. Consider the following situation: a robot is moving along a straight line path with a time-optimal trajectory. For a simple acceleration bound, the trajectory will consist of accelerating at the maximum rate until the middle of the path is reached, then decelerating at the maximum rate to come to rest at the goal position. Obviously, after passing the middle point, the robot cannot come to rest before the end of the path and hence the previously described algorithm would freeze the entire remaining path and would not allow any changes. The problem is that the robot is working at its limits of performance.

One intuitive, although less rigorous, scheme is to use some percentage of the robot's full actuator capacity when computing a time parameterization. A small segment of the path is frozen and the remainder is allowed to change according to sensor data. The reserve actuator capacity will enable valid time parameterizations to be generated in response to changes in the path. Such an approach is more ad hoc but works quite well in practice, however, situations can still occur in which no valid trajectory is possible even if all actuator effort is used. We currently handle such rare cases by reverting to the original path and associated trajectory.

Maintaining reserve actuator effort is also useful for another reason. The time-optimal trajectory along a path provides no margin for error during the motion of the robot. In practice, a robot follows a trajectory by using some type of feedback loop that compares the current state of the robot with the desired state specified by the trajectory. This comparison is used by the robot to adjust the actuator effort in order to correctly follow the trajectory. A time-optimal trajectory, however, uses one hundred percent of the available effort of some actuator; there is no room for adjustment.

In summary, it is possible to decompose the problem of modifying a trajectory of a moving robot into the two steps of modifying the path, and generating a time parameterization. Precautions must be taken, however, to prevent situations where no time parameterization

is feasible given the current velocity of the robot.

7.4 A Robotic System

We have constructed a prototype robotic system to validate the ideas presented in this thesis. The system was designed for the following scenario: three Puma 560 manipulator arms, operating in a shared environment, are required to perform user specified motions while avoiding collision with each other and the environment.

In the current system, each of the Puma manipulators operates independently. There is a separate elastic band for each manipulator, and the shape of an elastic is not effected by the planned motions of the other manipulators. The elastic is influenced, however, by the current configuration of the other arms; they can be considered as obstacles that can move.

Through a command language and a graphical interface resembling a traditional teach pendant, a user can move the manipulators. Motions of an arm are restricted so that collision is avoided with the other arms and the environment. For each arm, there is an associated elastic band. One end of the elastic band is connected to the current configuration of the robot. As an arm is moved, the elastic band records the motions; in this way, the user can specify the initial shape of the elastic.

The elastic band associated with each arm continuously deforms in response to the applied artificial forces. Such deformation attempts to improve the shape of the path for the arm. If the user stops moving the manipulators and the environment is static, the elastic band eventually reaches a state of equilibrium.

The user can interactively modify the path of a manipulator by varying several parameters of the potential functions associated with the elastic. The parameters include: the contraction gain, the repulsion gain, and the distance that the repulsion force extends for the obstacles. Varying these parameters produces global changes to the elastic.

A desirable addition to the system would be the ability to make local changes to the elastic. One idea, not yet implemented, is to provide a mechanism to specify configurations of attraction and repulsion. In this way, the user could "push" and "pull" the elastic into a desired shape. The specified configurations would influence the elastic by adding additional terms to the external potential defined over the configuration space of the robot. At the same time, the original potentials would maintain smoothness and obstacle avoidance for the elastic.

When requested, the manipulator moves along the elastic band. The speed along the elastic is determined by the algorithm, described earlier in the chapter, for generating an approximation to the time-optimal trajectory. The trajectory is generated incrementally, and is overlapped with the motion of the manipulator.

The experimental environment

The hardware for the robotic system consists of three Puma 560 manipulators, a real-time computer system, and several graphical workstations. The software consists of a set of communicating processes distributed across several processors. The software runs on top of a combination of a custom real-time operating system, called Oz, and UNIX.

The three Puma 560 manipulators are mounted on a table in such a way that they share a common workspace. Figure 7.10 depicts the manipulators when all joint angles are equal to zero. The Pumas are connected to modified Unimate controllers. These controllers have had most of the supplied electronics replaced by a single interface boards that enables high speed reading of the manipulators' joint encoders, and writing of the motor torques.

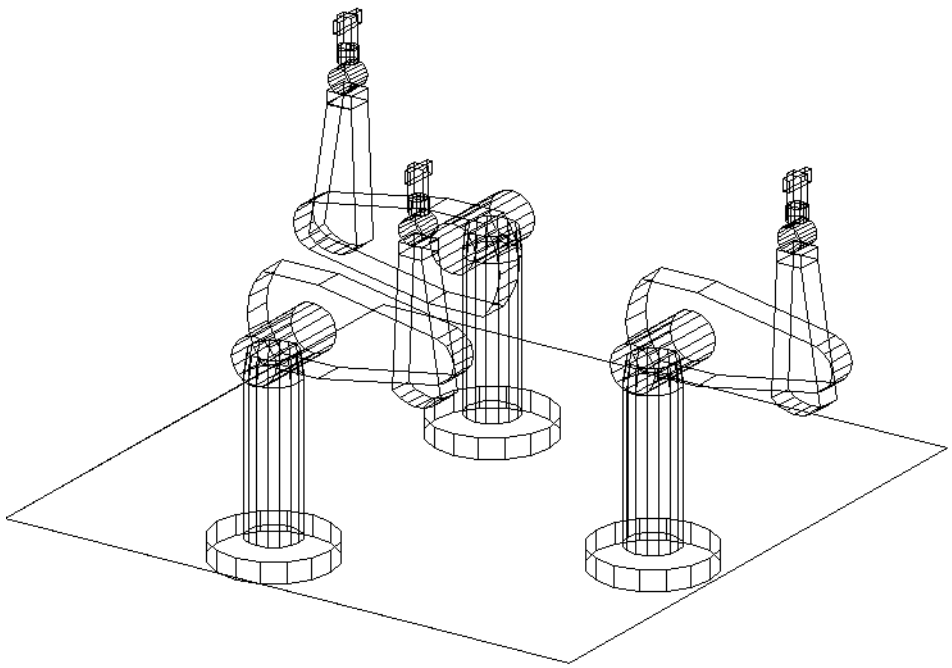


Figure 7.10: Three Puma manipulators mounted on a table.

The real-time computer system consists of a number of boards installed in two connected VME backplanes. These boards include: several CPUs, an interface to each of the PUMA controllers, a safety watchdog for the manipulators, video capture and processing, A/D and D/A boards, a laser light strip system, Ethernet, and SCSI. The system is used for several projects within the Stanford Robotics Laboratory.

Real-time computation is performed on two boards that are based on the Motorola 88K RISC CPU. The first board contains a single CPU, while the second contains four integrated CPUs. Each CPU operates at 20MHz, and is benchmarked at about 10 VAX MIPs.

Another CPU board, based on the Motorola 68030, runs a version of the UNIX operating system. Through an Ethernet connection onto the Stanford-wide network, this board acts as the bridge between the other UNIX machines in the laboratory and the real-time system. It also provides access to local magnetic storage.

The graphical workstations, running various versions of the UNIX operating system, provide the development environment and user interfaces for the robotic system.

The software architecture for the system is based on a small operating system, which we developed, called Oz [48]. With Oz, processes can communicate only by sending messages. Such an architecture enables transparent distribution across multiple processors, and can be implemented on top of standard operating systems such as UNIX.

Oz has been implemented for the five 88K CPUs in our system, and on top of UNIX. The 88K implementation provides real-time performance—scheduling is based on multi-level round-robin, context switch times are about 14 microseconds, and message passing is implemented by shared memory, even between processes on different processors. The UNIX version is not real-time, but has two distinct uses. First, it enables the robotic system to be simulated using the same code as the real-time implementation. This feature is beneficial in our environment, where the real-time hardware is concurrently used for several projects. Second, it provides the interface between the real-time version of Oz and UNIX. When Oz is started on the 88K CPUs, a UNIX version is also executed on the Motorola 68030 processor in the VME cage. Messages can be sent between processes on any of the CPUs. In particular, a process on one of the 88K CPUs, can send a message to a process on the UNIX machine. A process on the UNIX machine can execute both Oz and UNIX system calls, and can respond to a message by, for example, writing to a UNIX file system, communicating over the network, displaying graphical output, etc.

The world model

To implement elastic bands, a model of the manipulators and obstacles in the environment is needed. Such a model changes as the manipulators move, and as unexpected obstacles and changes in the environment are detected.

Each of the Puma manipulators is modeled as six movable links and a fixed base, all of which are assumed to be rigid. The links and the base are described by a boundary representation consisting of convex polygons. These polygons were determined by careful measurement of a Puma manipulator, and are accurate to approximately 2mm.

The position and orientation of the links for a manipulator are determined using forward kinematics and the displacement of each joint. The joint angles are measured by digital encoders incorporated in the manipulator. Errors in the calibration of the manipulator,

backlash, and flexibilities in the actuator transmissions result in an accuracy of about 1-2mm for each of the links.

The various cables and tubes that pass between links on the outside of the manipulator are not modeled. It seems unrealistic to ignore these elements; they can protrude a considerable distance from the manipulator, and their shape is particularly prone to snare objects in the environment. The flexible nature of cables and tubes, however, makes prediction of their shape difficult. The only solution that seems feasible is a conservative model that encloses the entire region where the cable and tubes may reside. We have yet to develop such a model.

In the current implementation, the model of the obstacles in the environment is not created automatically from sensor data. Instead, the user describes the environment as a collection of polygonal objects using a text-based modeling language. The model of the environment can be changed by specifying a new file to read.

We would much prefer to build the model of the environment from sensory input. Ideally, the model should be updated automatically and promptly as changes in the environment are detected. Towards this goal, we designed and constructed a structured light sensor, based on the conventional design of a plane of laser light with a camera mounted at an offset [23]. The sensor uses triangulation to determine the distance between the laser and the objects intersected by the plane of light. The accuracy of the range data is approximately 1mm.

The original plan was to mount the sensor on one of the Puma manipulators, enabling the sensor to scan the environment. Although some preliminary trials were conducted with this setup, it has yet to be integrated into the robotic system. The major issue that need to be addressed is the representation of the model constructed by the sensor. The representation should be able to describe complex three-dimensional environments and be easily updated using the range data provided by the sensor.

An overview of the implementation

The robot system is implemented as a collection of communicating processes. Figure 7.11 depicts the various processes and their relationship to each other. The arrows indicate the direction in which communication between processes is initiated. The system is comprised of the following processes:

Joint Controller: For each arm there is a joint control process that implements a computed torque feedback loop. The joint controller accepts trajectories from the elastic band process, and moves the robot along these trajectories. During the motion of the robot, the joint controller will accept updated trajectories that are consistent with the robot's current motion. The process also acts as a server for information about the state of the corresponding manipulators; other processes can query the controller to determine where the arm is. The process has the highest priority, and operates with a 200Hz servo loop.

Elastic Band: At the heart of this robotic system are the elastic band processes, one for each arm. Each process is responsible for the elastic band associated with its arm. The process modifies the elastic, and when requested, incrementally time-parameterizes the path and passes the resulting trajectory fragments on to the associated joint controller. The process also accepts commands to move the arm in specified directions, and serves information about the current state of the elastic. At regular intervals, the control processes are queried to determine the current state of all the arms. The environment model is obtained by reading a user specified file. Future implementations will compute an environment model for sensor data acquired by a light strip sensor.

Command Interpreter: The command interpreter is responsible for parsing user commands and interacting with the other processes in the system. Commands are provided to examine the current state of the system, move the robot arms, modify various parameters, etc.

Command Window: A standard text window is provided on a graphical workstation for the input commands and display of textual information.

Control Panel: A graphical control panel enables the user to move the arms in manner similar to a traditional teach pendant. A window on a workstation depicts various buttons, which can be pressed to move the manipulators in joint space, task space, and global coordinates.

3D Displays: Several windows on a workstation provide three-dimensional graphical information about the state of the system. One window depicts the state of the robots and the environment. For each arm, a separate window displays the current state of the associated elastic band by depicting a stylized version of the robot at each configuration in the discrete representation of the elastic.

Data Gathering: To analyze the behavior of the control process, information about the state of the robot can be recorded and stored as a MATLAB file. The data gathering process collects the data, and performs the file I/O.

Light Stripe Sensor: The light strip sensor has yet to be incorporated into the robot system. The plan is for this process to collect the data from the sensor and build a world model.

The implementation of the elastic band process deserves a more detailed description. As described in the Chapter 5, the elastic band is represented as a sequence of particles, with the constraint that the bubbles associated with consecutive particles overlap. Although there is no upper bound, a typical elastic for a Puma has twenty to forty particles.

The Puma 560 is an open chain manipulator, thus the bubble associated with a configuration can be generated as described in Chapter 4. The bubbles are computed from distance information obtained using the algorithm described in Chapter 6. We use a relative error of

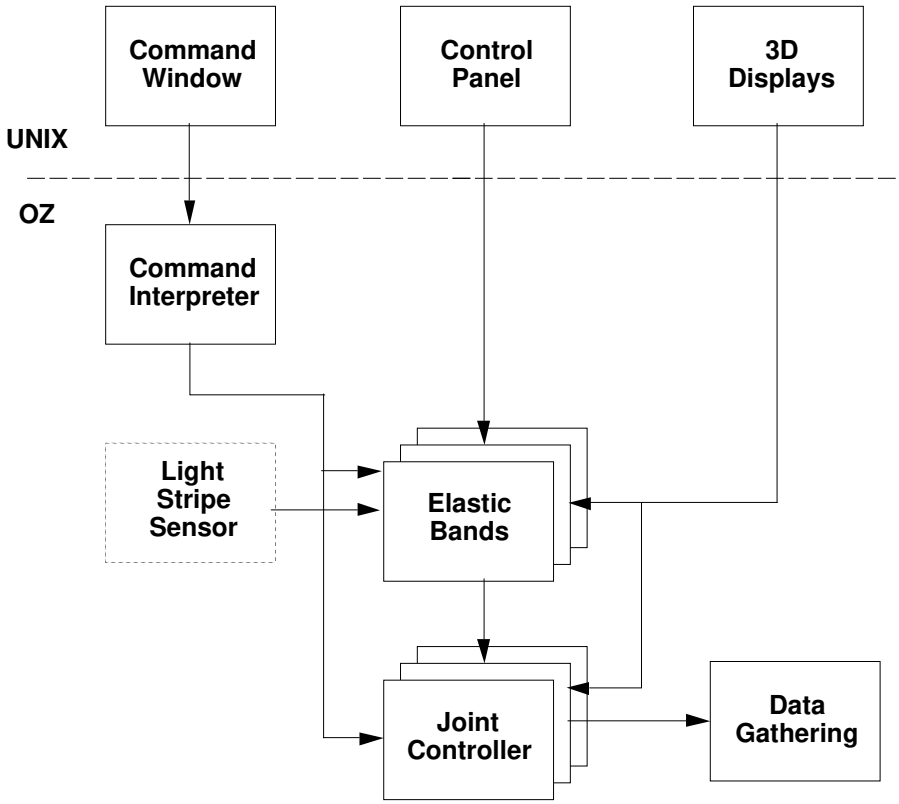


Figure 7.11: The various processes of the system

20 percent to speed the distance computation. The time to compute the distance depends on the configuration of the arms and the obstacles, but typically falls in the range of ten to fifty milliseconds, corresponding to a rate of 20-100Hz.

The elastic band is modified by moving one particle at a time. When suitably implemented, the motion of a particle requires approximately one distance computation on average. The distance computation dominates the computation cost of this operation; one particle can be moved every ten to fifty milliseconds.

When the manipulator is not in motion, the particles of the elastic are moved in a sequential order, starting from each end of the elastic in alternating cycles. For an elastic of approximately twenty particles, the entire elastic is updated at about 1-5Hz. This rate is sufficient to avoid the other manipulators if they move slowly, i.e., less than say 10 degrees per second. If the manipulators move faster, the elastic band may not respond sufficiently quickly to remain collision-free. In such a situation, the part of the elastic that is in collision stops deforming until the cause of the collision is removed.

A major issue that has not been resolved is how the elastic is modified when the associated manipulator is in motion. Modification of the path must be completed before

the manipulator reaches the modified section, otherwise the manipulator will no long be on the path. In addition, as mentioned in the previous section, the interaction between path modification and trajectory generation must be considered if situations where no feasible trajectory exists are to be avoided. The situation is further complicated by the non-deterministic time required to move a particle of the elastic, and the computational requirements of the incremental time-parameterization that occurs concurrently with the motion.

In the current implementation, the shape of the elastic is frozen while the manipulator moves. This "solution" to the above problem is not particularly attractive. After all, a major goal of the elastic band framework is the implementation of reactive behaviors; the freezing of the elastic band precludes changes to the motion of the robot. One the other hand, the Puma manipulators can move along a given path in a short amount of time; a few seconds is typical. Given that the elastic band can only respond to slow changes, little modification to the elastic would occur if it was permitted during motion.

In simulation, we have combined the modification of an elastic band and the motion of the robot for both a mobile robot and the Puma manipulators. With simulation, the real-time constraints of the system can be relaxed. In particular, the modification of the elastic band is allowed to complete a full cycle before simulated time is advance on time-step; this is only possible because the robot is not actually moving. Also, the simulation of the robot system can be implemented on current workstation technology, which has considerably more computational power than our four-year-old real-time hardware. These simulations suggest that improved computational power and the development of a more sophisticated scheduling algorithm for the modification of the elastic should enable the full elastic band architecture to be implemented on actual robots.

During the motion of the manipulator, the real-time incremental time-parameterization algorithm, described early in the chapter, is used to compute the desire speed of the robot along the path. This scheme requires each iteration of the time-parameterization algorithm to be completed in less time than the interval Δt associated with the algorithm. For the current implementation, Δt is 0.1 seconds.

To increase the robustness of the system, each iteration of the time-parameterization algorithm generates a trajectory that brings the robot to rest. The first Δt seconds of the trajectory correspond to the approximation to the time-optimal parameterization. The remainder of the trajectory simply decelerates the robot at the maximum rate. The next iteration of the time-parameterization algorithm should replace the second segment of the trajectory. If the iteration fails to complete before the required dead-line, the control process will not receive an updated trajectory and will follow the remainder of the previous trajectory; i.e, the robot will decelerate.

Evaluation and future work

The robotic system described above demonstrates that the elastic band concept can be implemented for complex real-world robots such as the Puma 560. The current implementation enables a path to be specified by the user. The path deforms to improve its shape and to react to the motion of the other manipulators. The deformation occurs in real-time, although the update rate of the elastic is slower than we would have preferred. The real-time trajectory generation algorithm has been implemented successfully, allowing the robot to move quickly from one end of the elastic to the other.

On the other hand, the robotic system we have developed is only a first prototype. There are many areas where the system could be improved, including:

1. *Modifying the elastic while the manipulator is moving*. The modifications must be performed in some fixed amount of time, perhaps equal to the time-parameterization interval Δt. Scanning the entire elastic, moving one particle at a time, is not feasible and more sophisticated strategies are needed for determining which particles to move. Increased computational power would also help.

2. *Building the environment model from sensor data*. The addition of this capability would add greatly to robustness and autonomy of the system. The issues involved, however, are complex. What representation should be use? How are the obstacles differentiated from the manipulators? How is conflicting data resolved? How should the light strip system scan the environment? This problem is interesting, but requires substantial work to solve satisfactorily.

3. *Improved task specification*. In the current implementation, the robots can only move back and forth along an elastic. The initial shape of the elastic is specified by moving the robot along the desired path. A more realistic system would allow a much richer specification of the desired motions. For example, we envision a system with a powerful robot programming language that incorporates the elastic band capability. Also, as mentioned in the text, users should be able to interactively "push" and "pull" an elastic into a desired shape by varying the various potentials that determine the applied artificial forces.

4. *Integration of path planning*. Enabling the system to plan the initial shape of an elastic would allow a user to specify tasks at a higher level. Path planning could also be used to generate non-local modifications to the elastic. In the current implementation, changes to the path are restricted to a single homotopic class; the path will never change its initial topology. If the environment is changing, it may not be possible for the elastic to remain collision-free. This problem may be overcome by enabling the system to to plan an alternative path for a segment of the elastic.

The implementation of the prototype robotic system demonstrates that there are many issues to be explored. The elastic band concept is feasible, and provides a useful capability. Integrating this capability into a practical robotic system that can be used for real-world applications, however, remains a challenging problem. We feel this problem is worthy of further study.

Chapter 8

Conclusion

In the following, the major ideas of this thesis are summarized.

Elastic bands form the basis for an effective framework to deal with real-time collision-free motion control for a robot operating in an evolving environment. A planner provides an initial path that is a solution to the problem of moving a robot between a start and goal configuration. Incremental adjustments to the path are made while maintaining a global path in the free-space. These modifications are based on sensory data about the environment and desired criteria concerning the path, such as length, smoothness, and obstacle clearance. Implemented as a real-time servo-loop, an elastic band provides many of the benefits of reactive systems without sacrificing global planning.

Bubbles enable elastic bands to be implemented efficiently. A bubble is a local region of free-space around a configuration of the robot. The bubbles are in the configuration space of the robot, but the global free-space is not computed. The bubbles are generated from distance information obtained directly from a model of the environment and the robot. We believe that bubbles have potential applications in many areas of robotics. A bubble provides information about where a robot can move from a given configuration. This information is generated efficiently, even for robots with many degrees of freedom.

The computational effort required to generate a bubble is dominated by the necessary distance computations. We have developed a novel algorithm for efficiently computing the distance between non-convex objects. The algorithm is based on a combination of a hierarchical bounding representation, a simple search routine, and a convex distance algorithm.

To move a robot along a path requires a time-parameterization. We have developed an incremental time-parameterization algorithm that generates a discrete approximation to the time-optimal trajectory. The generation of the parameterization can be overlapped with the motion of the robot, effectively eliminating the time-cost of the algorithm.

To demonstrate the ideas presented in the thesis, we have implemented a prototype robotic system. The system controls three Puma 560 manipulators, operating in a shared

environment. Although the prototype system could be improved, it demonstrates that a path for a complex robot such as the Puma can be modified in real-time. The system also demonstrated the overlapping of the time-parameterization algorithm with the motion of the manipulator.

Appendix A

The Functional Gradient Operator $\overline{\nabla}$

In this appendix, the functional gradient operator $\overline{\nabla}$ is developed. This operator represents a generalization of the standard gradient operator ∇. In particular, $\overline{\nabla}$ can be applied to a class of functionals (functions of functions), whereas the standard gradient operator can be applied only to functions of a finite number of variables.

Suppose we define some quantity that is determined by the shape of a given curve. Since a curve can be represented as a parameterized function, such a quantity can be defined by a functional, a function of a function. For example, the length of a curve can be defined as the functional $L[\mathbf{c}]$, where

$$L[\mathbf{c}] = \int_0^1 \|\mathbf{c}'(s)\|\, ds. \tag{A.1}$$

The problem we wish to address is: how does the value of a functional vary with small changes of the curve? Such questions are connected with the *calculus of variations*, a classical branch of analysis [16]. In particular, the calculus of variations is concerned with finding extrema of functionals, that is, finding curves that maximize or minimize the value of the functional. In the following, we present the calculus of variations as a generalization of the gradient operator ∇.

The class of functionals we are interested in have the form of an integral of some known function that depends on the given curve and the derivatives of the curve. The domain of the functional is a set of parameterized curves in some n-dimensional space, with continuous derivatives up to the $2m$-th order. Assuming the curves are defined over the domain $[0, 1]$, the functional $V[\mathbf{c}]$ has the form

$$V[\mathbf{c}] = \int_0^1 v(\mathbf{c}, \mathbf{c}', \mathbf{c}'', \ldots, \mathbf{c}^{(m)})\, ds,$$

where v is a function that has continuous partial derivatives up to the $2m$-th order with respect to its arguments $\mathbf{c}, \mathbf{c}', \mathbf{c}'', \ldots, \mathbf{c}^{(m)}$. As an example, the length functional defined in equation (A.1) has the required form as long as the domain of curves has continuous second derivatives.

Now, consider a function $\boldsymbol{\eta}(s)$ that maps the interval [0, 1] into the same n-dimensional space as $\mathbf{c}(s)$, and also possesses continuous derivatives up to the $2m$-th order. We can construct a series of variations of a curve $\mathbf{c}$ using the equation

$$\overline{\mathbf{c}}(\epsilon) = \mathbf{c} + \epsilon\boldsymbol{\eta},$$

where ϵ parameterizes the variations.

For a given $\mathbf{c}$ and $\boldsymbol{\eta}$, the functional $V[\overline{\mathbf{c}}]$ may be regarded as a function $V(\epsilon)$. At $\epsilon = 0$, the derivative of V with respect to ϵ describes how V varies when $\mathbf{c}$ is moved with a velocity of $\boldsymbol{\eta}$. By the method of differentiating under the integral sign and applying the chain rule, one finds that

$$\frac{dV}{d\epsilon} = \int_0^1 \left(\frac{\partial v}{\partial \overline{\mathbf{c}}} \cdot \frac{\partial \overline{\mathbf{c}}}{\partial \epsilon} + \frac{\partial v}{\partial \overline{\mathbf{c}}'} \cdot \frac{\partial \overline{\mathbf{c}}'}{\partial \epsilon} + \frac{\partial v}{\partial \overline{\mathbf{c}}''} \cdot \frac{\partial \overline{\mathbf{c}}''}{\partial \epsilon} + \cdots + \frac{\partial v}{\partial \overline{\mathbf{c}}^{(m)}} \cdot \frac{\partial \overline{\mathbf{c}}^{(m)}}{\partial \epsilon} \right) ds. \tag{A.2}$$

Consider the second term in this integral:

$$\int_0^1 \frac{\partial v}{\partial \overline{\mathbf{c}}'} \cdot \frac{\partial \overline{\mathbf{c}}'}{\partial \epsilon}\, ds = \int_0^1 \frac{\partial v}{\partial \overline{\mathbf{c}}'} \cdot \frac{\partial^2 \overline{\mathbf{c}}}{\partial s \partial \epsilon}\, ds.$$

Integrating by parts we have

$$\int_0^1 \frac{\partial v}{\partial \overline{\mathbf{c}}'} \cdot \frac{\partial^2 \overline{\mathbf{c}}}{\partial s \partial \epsilon}\, ds = \left. \frac{\partial v}{\partial \overline{\mathbf{c}}'} \cdot \frac{\partial \overline{\mathbf{c}}}{\partial \epsilon} \right|_0^1 - \int_0^1 \frac{d}{ds} \left(\frac{\partial v}{\partial \overline{\mathbf{c}}'} \right) \cdot \frac{\partial \overline{\mathbf{c}}}{\partial \epsilon}\, ds. \tag{A.3}$$

If we require that the endpoints of $\overline{\mathbf{c}}$ remain fixed with respect to ϵ, the first term on the right hand side of (A.3) vanishes.

By repeated integration by parts, and requiring that the $(m-1)$-th order derivatives of $\overline{\mathbf{c}}$ remain constant at the endpoints, successive term of equation (A.2) can be transformed such that the variation of V is given by

$$\frac{dV}{d\epsilon} = \int_0^1 \left(\frac{\partial v}{\partial \overline{\mathbf{c}}} - \frac{d}{ds}\frac{\partial v}{\partial \overline{\mathbf{c}}'} + \frac{d^2}{ds^2}\frac{\partial v}{\partial \overline{\mathbf{c}}''} - \cdots + (-1)^m \frac{d^m}{ds^m}\frac{\partial v}{\partial \overline{\mathbf{c}}^{(m)}} \right) \cdot \frac{\partial \overline{\mathbf{c}}}{\partial \epsilon}\, ds.$$

At $\epsilon = 0$, this equation reduces to

$$\left. \frac{dV}{dt} \right|_0 = \int_0^1 \left(\frac{\partial v}{\partial \mathbf{c}} - \frac{d}{ds}\frac{\partial v}{\partial \mathbf{c}'} + \frac{d^2}{ds^2}\frac{\partial v}{\partial \mathbf{c}''} - \cdots + (-1)^m \frac{d^m}{ds^m}\frac{\partial v}{\partial \mathbf{c}^{(m)}} \right) \cdot \boldsymbol{\eta}\, ds. \tag{A.4}$$

Using the notion

$$\overline{\nabla} v = \frac{\partial v}{\partial \mathbf{c}} - \frac{d}{ds}\frac{\partial v}{\partial \mathbf{c}'} + \frac{d^2}{ds^2}\frac{\partial v}{\partial \mathbf{c}''} - \cdots + (-1)^m \frac{d^m}{ds^m}\frac{\partial v}{\partial \mathbf{c}^{(m)}}, \tag{A.5}$$

we can rewrite (A.4) as

$$\left.\frac{dV}{dt}\right|_0 = \int_0^1 \overline{\nabla} v \cdot \boldsymbol{\eta}\, ds. \tag{A.6}$$

The functional gradient operator $\overline{\nabla}$ can be viewed as a generalization of the standard gradient operator ∇. In the following, we illustrate four properties that the two operators have in common.

1. Given a function $g(\mathbf{x})$ and a unit vector $\mathbf{n}$, the derivative of g in the direction $\mathbf{n}$ is defined as
$$\frac{dg(\mathbf{x}+t\mathbf{n})}{dt}.$$
Using the gradient operator, the directional derivative can be written as
$$\frac{dg(\mathbf{x}+t\mathbf{n})}{dt} = \nabla g \cdot \mathbf{n}. \tag{A.7}$$
When the direction is given in terms of a vector whose norm is not one, the vector must first be divided by its norm before applying the above equation.

   We can extend the notion of a directional derivative to functionals. Suppose $\boldsymbol{\eta}(s)$ has the property that
$$\int_0^1 \|\boldsymbol{\eta}\| = 1. \tag{A.8}$$
We can view $\boldsymbol{\eta}$ as a unit function, and define the *derivative of* V *in the direction of* $\boldsymbol{\eta}$ as
$$\frac{dV[\mathbf{c}+t\boldsymbol{\eta}]}{dt}.$$
Using equation (A.6), the direction derivative is given by the equation
$$\frac{dV[\mathbf{c}+t\boldsymbol{\eta}]}{dt} = \int_0^1 \overline{\nabla} v \cdot \boldsymbol{\eta}\, ds. \tag{A.9}$$
Note the similarity between right hand sides of equations (A.7) and (A.9). We can view (A.9) as the dot product of the gradient and the direction $\boldsymbol{\eta}$ over the infinite number of degrees of freedom inherent in the curve $\mathbf{c}$. As with the discrete case, if the direction is given in terms of a function that does not satisfy the condition (A.8), the function must first be normalized.

2. For a function $g(\mathbf{x})$, the direction of maximal increase is given by the vector ∇g. Similarly, the direction that causes the greatest increase in V is given by the function $\overline{\nabla} v(s)$.

The function $\overline{\nabla}v(s)$ will not, in general, have unit norm. To find the directional derivative using (A.9), let $\boldsymbol{\theta}$ be the normalized version of $\overline{\nabla}v(s)$, i.e.,

$$\boldsymbol{\theta}(s) = \frac{1}{c}\overline{\nabla}v(s),$$

where

$$c = \int_0^1 \|\overline{\nabla}v\|\, ds.$$

Now, suppose $\boldsymbol{\eta}$ is another unit function. Given that the square of a number is always positive, it can be seen that

$$\int_0^1 (\boldsymbol{\theta} - \boldsymbol{\eta})^2\, ds \geq 0.$$

Expanding and recalling that $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ are unit functions, we have

$$2 - 2\int_0^1 \boldsymbol{\theta} \cdot \boldsymbol{\eta}\, ds \geq 0.$$

Rearranging terms gives

$$\int_0^1 \boldsymbol{\theta} \cdot \boldsymbol{\eta}\, ds \leq 1.$$

Using equation (A.9), the derivative in the direction $\boldsymbol{\theta}$ is

$$\int_0^1 \overline{\nabla}v \cdot \boldsymbol{\theta}\, ds = c\int_0^1 \boldsymbol{\theta} \cdot \boldsymbol{\theta}\, ds = c.$$

For the direction $\boldsymbol{\eta}$, the derivative is

$$\int_0^1 \overline{\nabla}v \cdot \boldsymbol{\eta}\, ds = c\int_0^1 \boldsymbol{\theta} \cdot \boldsymbol{\eta}\, ds \leq c.$$

Thus, the directional derivative is greatest in the direction $\boldsymbol{\theta}$ which is equivalent to the direction $\overline{\nabla}v(s)$.

3. If $\mathbf{x}$ is a function of time t, the derivative of a function $g(\mathbf{x}(t))$ with respect to time is given by the equation

$$\dot{g} = \nabla g \cdot \dot{\mathbf{x}}.$$

In an analogous fashion, if $\mathbf{c}$ is a function of both s and t, then $V[\mathbf{c}] = V(t)$ is a function of t. If at time t, we let $\boldsymbol{\eta} = \dot{\mathbf{c}}$, then using equation (A.6),

$$\left.\frac{dV[\overline{\mathbf{c}}]}{d\epsilon}\right|_0 = \int_0^1 \overline{\nabla}v \cdot \dot{\mathbf{c}}\, ds.$$

Noting that $d\overline{\mathbf{c}}/d\epsilon = \dot{\mathbf{c}}$, at time t we have

$$\dot{V}(t) = \frac{dV[\mathbf{c}]}{dt} = \left.\frac{dV[\overline{\mathbf{c}}]}{d\epsilon}\right|_0 = \int_0^1 \overline{\nabla}v \cdot \dot{\mathbf{c}}\, ds. \tag{A.10}$$

4. The vector $\mathbf{x}$ is a stationary value of g if

$$\nabla g(\mathbf{x}) = 0.$$

A curve $\mathbf{c}$ is a stationary value of $V[\mathbf{c}]$ if any infinitesimal variation in $\mathbf{c}$ will not result in a change in V. The fundamental lemma of the calculus of variations states that $\mathbf{c}$ is a stationary value iff, for all values of s,

$$\overline{\nabla}\mathbf{c}(s) = 0. \tag{A.11}$$

The proof of this lemma may be found in most texts on the calculus of variations [16].

Appendix B

The Curvature Vector

One of the properties of a curve is how much the curve "bends" at a given point, referred to as the *curvature*. From differential geometry [19], the curvature $\kappa(s)$ for a curve $\mathbf{c}(s)$ parameterized by arc length is defined as

$$\kappa(s) = \left\| \frac{d^2\mathbf{c}}{ds^2} \right\| = \|c''(s)\|.$$

An informal justification for the above definition is as follows. For a point $\mathbf{p}$ on a curve, consider the circle that has second order contact with the curve as shown in Figure B.1. This circle, referred to as the *osculating circle*, always exits and is unique as long as the curve is not linear at the point $\mathbf{p}$. The radius ρ of the osculating circle indicates how much the curve bends at $\mathbf{p}$; intuitively, as the radius increases, the curvature decreases. In the next paragraph, we show that $\kappa(s) = 1/\rho(s)$.

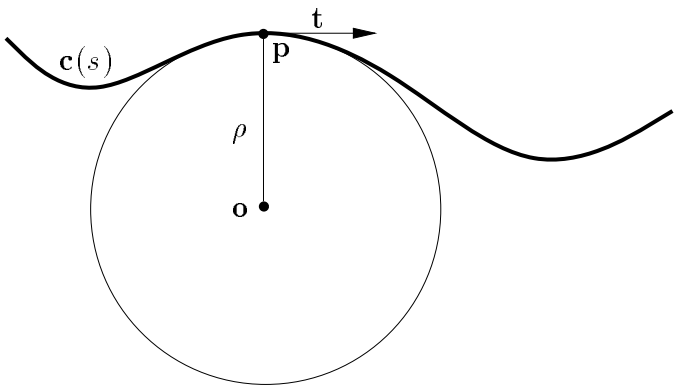


Figure B.1: The osculating circle for a point $\mathbf{p}$ on the curve $\mathbf{c}(s)$. The circle is centered at the point $\mathbf{o}$ and has radius ρ. The vector $\mathbf{t}$ is the tangent to the curve at $\mathbf{p}$

The arc length parameterization for a circle in the x–y plane, of radius ρ, and centered at the origin, can be written as follows:

$$\begin{aligned} x &= \rho \cos \frac{s}{\rho}, \\ y &= \rho \sin \frac{s}{\rho}. \end{aligned}$$

Differentiating with respect to s we get

$$\begin{aligned} x' &= -\sin \frac{s}{\rho}, \\ y' &= \cos \frac{s}{\rho}. \end{aligned}$$

As required for an arc length parameterization, the magnitude of the velocity vector (x', y') is equal to one for all s. Differentiating again with respect to s we get

$$\begin{aligned} x'' &= -\frac{1}{\rho} \cos \frac{s}{\rho}, \\ y'' &= -\frac{1}{\rho} \sin \frac{s}{\rho}. \end{aligned}$$

The magnitude of the acceleration vector (x'', y'') equals $1/\rho$. Now, the osculating circle at a given point is required to have second order contact with the curve. If the curve is also parameterized by arc length, the magnitude of $\mathbf{c}''$ should equal the magnitude of (x'', y''), i.e.,

$$\kappa(s) = \|c''(s)\| = \frac{1}{\rho(s)}.$$

Related to the curvature κ is the curvature vector $\boldsymbol{\kappa}$, defined as the vector of magnitude κ that points from the point $\mathbf{p}$ on the curve towards the center $\mathbf{o}$ of the osculating circle. One can see that the vector (x'', y'') points towards the center of the associated circle. If $\mathbf{c}$ is parameterized by arc length, the second order contact between the curve and the osculating circle implies that the vector $\mathbf{c}''$ points towards the center $\mathbf{o}$. Thus, we have

$$\boldsymbol{\kappa}(s) = \mathbf{c}''.$$

Now suppose, that curve $\mathbf{c}(s^*)$ is not parameterized by arc length. Differentiating $\mathbf{c}(s^*)$ twice with respect to arc length s, we have

$$\begin{aligned} \frac{d^2\mathbf{c}}{ds^2} &= \frac{d}{ds}\left(\mathbf{c}'\frac{ds^*}{ds}\right) \\ &= \mathbf{c}''(\frac{ds^*}{ds})^2 + \mathbf{c}'\frac{d^2s^*}{ds^2} \end{aligned} \tag{B.1}$$

Since s is the arc length of $\mathbf{c}$, the function relating s and s^* is given by

$$s(s^*) = \int_0^{s^*} \|\mathbf{c}'\| ds^*.$$

Thus,

$$\frac{ds^*}{ds} = \|\mathbf{c}'\|,$$

giving

$$\frac{ds^*}{ds} = \frac{1}{\|\mathbf{c}'\|}.$$

Differentiating with respect to s, we have

$$\frac{d^2 s^*}{ds^2} = \frac{\mathbf{c}' \cdot \mathbf{c}''}{\|\mathbf{c}'\|^4}.$$

Substituting into equation (B.1), we have

$$\boldsymbol{\kappa}(s) = \frac{d^2\mathbf{c}}{ds^2} = \frac{1}{\|c'\|^2}\left(c'' + \frac{\mathbf{c}' \cdot \mathbf{c}''}{\|\mathbf{c}'\|^2}\mathbf{c}'\right). \tag{B.2}$$

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] D. Baraff. "Analytical methods for dynamic simulation of non-penetrating rigid bodies," *Computer Graphics (Proc. SIGGRAPH)*, 23(3), pp. 223–232, July 1989.

[3] D. Baraff. "Rigid body simulation," *SIGGRAPH Course Notes*, 19, 1992.

[4] J. Barraquand and J. C. Latombe. "Robot motion planning: a distributed representation approach," *The International Journal of Robotics Research*, 10(6), 1991.

[5] R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An Introduction to Splines for use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann, 1987.

[6] J. E. Bobrow. "Optimal robot path planning using the minimum-time criterion," *IEEE Journal of Robotics and Automation*, 4(4), pp. 443–50, August 1988.

[7] J. E. Bobrow. "A direct minimization approach for obtaining the distance between convex polyhedra," *The International Journal of Robotics Research*, 8(3), pp. 65–76, 1989.

[8] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, 4(3), pp. 3–17, 1985.

[9] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice Hall, 1973.

[10] D. L. Brock and J. K. Salisbury. "Implementation of behavioral control on a robot hand/arm system," In *Experimental Robotics 2*, edited by R. Chatila and G. Hirzinger. Springer-Verlag, 1993.

[11] R. A. Brooks. "Solving the find-path problem by good representation of free-space," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3), pp. 190–197, 1983.

[12] R. A. Brooks. "A robust layered control system for a mobile robot," *Proc. IEEE International Conference on Robotics and Automation*, RA-2(1), 1986.

[13] R. A. Brooks and T. Lozano-Pérez. "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(2), pp. 244–233, 1985.

[14] J. F. Canny. *The complexity of robot motion planning*. MIT Press, 1988.

[15] W. Choi. *Contingency-tolerant motion planning and control*. PhD thesis, Stanford University, 1993.

[16] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Interscience, 1953. Republished by Wiley in 1989.

[17] J. J. Craig. *Introduction to Robotics: Mechanics and Control*, 2nd ed. Addison-Wesley, 1989.

[18] A. P. del Pobil, M. A. Serna, and J. Lovet. "A new representation for collision avoidance and detection," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 246–251, 1992.

[19] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentice-Hall, 1976.

[20] G. E. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A practical Guide*, 2nd ed. Academic Press, 1990.

[21] B. Faverjon. "Hierarchical object models for efficient anti-collision algorithms," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 333–340, Scottsdale, 1989.

[22] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practices*, 2nd ed. Addison-Wesley, 1990.

[23] K. S. Fu, R. C. Gonzalez, and C. S. G Lee. *Robotics: control, sensing, vision, and intelligence*. McGraw-Hill, 1987.

[24] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, 4(2), April 1988.

[25] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[26] H. Goldstein. *Classical Mechanics*, 2nd ed. Addison-Wesley, 1980.

[27] Th. Horsch, F. Schwarz, and H. Tolle. "Motion planning with many degrees of freedom—random reflections at C-space obstacles," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 3318–3323, San Diego, 1994.

[28] K. Kant and S. W. Zucker. "Towards efficient trajectory planning: path velocity decomposition," *The International Journal of Robotics Research*, 1(5), pp. 72–89, 1986.

[29] K. Kant and S. W. Zucker. "Planning smooth collision-free trajectories: path, velocity, and splines in free-space," *The International Journal of Robotics Research*, 2(3), pp. 117–126, 1987.

[30] M. Kass, A. Witkin, and D. Terzonpoulos. "Snakes: active contour models," *International Journal of Computer Vision*, pp. 321–331, 1988.

[31] L. Kavraki and J. C. Latombe. "Randomized preprocessing of configurationspace for fast path planning," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 2138–2145, San Diego, 1994.

[32] O. Khatib. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Esapce, Toulouse, 1980. (in French).

[33] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, 5(1), pp. 90–98, 1986.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing," *Science*, 220, pp. 671–680, 1983.

[35] B. H. Krogh and C. E. Thorpe. "Integrated path planning and dynamic steering control for autonomous vehicles," In *Proc. IEEE International Conference on Robotics and Automation*, San Franciscoo, 1986.

[36] C. Lanczos. *The Variational Principles of Mechanics*, 4th ed. University of Toronto Press, 1970. Republished by Dover in 1986.

[37] S. Lang. *Calculus of Several Variables*, 2nd ed. Addison-Wesley, 1979.

[38] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[39] M. C. Lin and J. F. Canny. "A fast algorithm for incremental distance calculation," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 1008–1014, 1991.

[40] T. Lozano-Pérez. "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, C-32(2), pp. 108–120, 1983.

[41] V. Lumelsky. "Algorithmic issues of sensor-based robot motion planning," In *Proceedings of the 26th conference on decision and control*, pp. 1796–1801, Los Angeles, 1987.

[42] V. J. Lumelsky. "On fast computation of distance between line segments," *Information Processing Letters*, 21, pp. 55–61, 1986.

[43] N. J. Nilsson. "A mobile automaton: an application of artificial intelligence techniques," In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pp. 509–520, Washington D.C., 1969.

[44] C. Ó'Dúnlaing and C. K. Yap. "A retraction method for planning the motion of a disc," *Journal of Algorithms*, 1(6), pp. 104–111, 1982.

[45] G. Pardo-Castellote. *Experimental Integration of Planning in a Distributed Robotic System*. PhD thesis, Stanford University, 1995. In preparation.

[46] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[47] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*, 2nd ed. Cambridge Press, 1992.

[48] S. Quinlan. "Oz reference manual," In preparation.

[49] S. Quinlan and O. Khatib. "Elastic bands: connecting path planning and control," In *Proc. IEEE International Conference on Robotics and Automation*, Atlanta, 1993.

[50] S. Quinlan and O. Khatib. "Towards real-time execution of motion tasks," In *Experimental Robotics 2*, edited by R. Chatila and G. Hirzinger. Springer-Verlag, 1993.

[51] J. H. Reif. "Complexity of the mover's problem and generalizations," In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pp. 144–154, 1979.

[52] E. Rimon and D. E. Koditschek. "Exact robot navigation using cost functions: the case of distinct spherical boundaries in E^n," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 1791–1796, Philadelphia, 1988.

[53] E. Rimon and D. E. Koditschek. "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," In *Proc. IEEE International Conference on Robotics and Automation*, pp. 21–26, Scottsdale, 1989.

[54] D. F. Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[55] J. T. Schwartz and M. Sharir. "On the 'piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, 1(4), pp. 293–351, 1993.

[56] K. G. Shin and N. D. McKay. "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, AC-30(6), pp. 531–541, 1985.

[57] W. Choi D. J. Zhu and J. C. Latombe. "Contingency-tolerant motion planning and control," In *Proceedings of IROS 89*, Tsukuba, Japan, 1989.