OPTIMIZATION APPROACHES FOR GEOMETRIC CONSTRAINTS IN ROBOT MOTION PLANNING

By

Nilanjan Chakraborty

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the Examinining Committee:

Srinivas Akella, Thesis Adviser

Jeff Trinkle, Thesis Adviser

John Mitchell, Member

John Wen, Member

Rensselaer Polytechnic Institute Troy, New York

October 2008
(For Graduation December 2008)
OPTIMIZATION APPROACHES FOR GEOMETRIC CONSTRAINTS IN ROBOT MOTION PLANNING

By

Nilanjan Chakraborty

An Abstract of a Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file in the Rensselaer Polytechnic Institute Library

Excluding Committee:

Srinivas Akella, Thesis Adviser
Jeff Trinkle, Thesis Adviser
John Mitchell, Member
John Wen, Member

Rensselaer Polytechnic Institute
Troy, New York

October 2008
(For Graduation December 2008)
# CONTENTS

LIST OF TABLES ........................................................................... v
LIST OF FIGURES ....................................................................... vi

1. Introduction ........................................................................... 1
   1.1 Thesis Outline and Contributions ...................................... 2

2. Rigid Body Dynamics: Complementarity-Based Modeling .......... 7
   2.1 Introduction ...................................................................... 7
   2.2 Dynamic Model for Rigid Body Systems ......................... 8
      2.2.1 Discrete Time Model .............................................. 13
   2.3 Effect of Linearizing the Distance Function .................... 15
   2.4 Conclusion ...................................................................... 20

3. Dynamic Simulation-based Kinodynamic Motion Planning .......... 21
   3.1 Introduction ...................................................................... 21
   3.2 Related Literature .......................................................... 22
   3.3 Sampling-Based Motion Planning with Differential Constraints .. 24
   3.4 Proposed Planner and its advantages .............................. 26
      3.4.1 Planning using a single potential field ....................... 28
      3.4.2 Effect of choice of $\Delta t$ and input ......................... 29
      3.4.3 Implication for narrow passage problems ................ 30
      3.4.4 Completeness Properties of our Algorithm ............. 30
      3.4.5 Use of our algorithm for purely geometric problems .. 31
   3.5 Simulation Results .......................................................... 31
      3.5.1 Example 1: Planar point robot .................................. 31
      3.5.2 Example 2: 2R Manipulator ...................................... 34
   3.6 Conclusion ...................................................................... 38

4. Proximity Queries between Convex Objects .............................. 44
   4.1 Introduction ...................................................................... 44
   4.2 Related Work ................................................................. 47
   4.3 Mathematical Preliminaries ............................................ 49
LIST OF TABLES

4.1 Sample run times, in milliseconds, for proximity queries between pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times over 100,000 random configurations. All data was obtained on a 2.2 GHz Athlon 64 X2 4400+ machine with 2 GB of RAM. 68

4.2 Sample continuous proximity query run times between pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times over 100,000 pairs of random configurations. For the time of first contact queries, only those configuration pairs that resulted in collisions were used and the reported query time is the total query time for solving both problems. 68

4.3 Sample proximity query run times between deforming pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times at each of 10 steps in the shape change, over 100,000 random configurations. 68

6.1 Performance Comparison of Splitting between Greedy and Matching algorithm 120

6.2 TSP tour obtained in pair space with improved cost given by the order improvement heuristic 125

6.3 Overall Performance gain achieved by using a 2-robot system over a single robot system. See text for details. 127

6.4 Greedy Algorithm Results for four-head machine on example data files 129

6.5 Overall Performance gain achieved by using a 4-robot system over a single robot system 130

7.1 Performance Comparison of Greedy and Matching algorithms for Splitting, with unit processing time for each point. 143

7.2 Performance of Greedy Algorithm on large datasets. 143
### LIST OF FIGURES

2.1  Schematic sketch of different situations of a point mass moving towards a fixed object (a) Curved convex object (b) Curved non-convex object (c) The normal at the closest point on the triangle is not uniquely defined; the line joining the closest point is one choice (d) The particle at time $\ell + 1$ cannot cross the dashed line, since the projection on the old normal added to the old distance becomes 0, even though the actual distance is non-zero at $\ell + 1$.  

2.2  A bar approaching the fixed surface. At the configuration shown the right end of the bar is very near the surface. The bar has a linear velocity toward the surface and a counterclockwise angular velocity. In this case the collision at the left end of the bar may be missed due to the error in evaluating $r$.  

2.3  Reduction of kinetic energy over time for a rolling disc approximated as a uniform regular polygon. As the number of edges of the polygon increases, the energy loss decreases. The computed value obtained by our time-stepper using an implicit surface description of the disc is the horizontal line at the top. The time step used is 0.01 seconds.  

2.4  Effect of the time step on the loss of kinetic energy for a polygon modeled by a fixed number of edges (1000 in this figure). The energy loss decreases with decreasing step size, up to a limit. In this case, the limit is approximately 0.001 seconds (the plots for 0.001, 0.0005, and 0.0001 are indistinguishable).  

3.1  Schematic sketch of a situation where it is difficult to find a path using motions only along two directions.  

3.2  Path of a point robot moving through a narrow passage formed by the wedge shaped obstacles.  

3.3  Enlarged portion of the path of the point robot that shows the safety distance from the obstacle is maintained.  

3.4  Velocity of the point robot.  

3.5  X-component of the forces/impulses acting on the point robot.  

3.6  Y-component of the forces/impulses acting on the point robot.  

3.7  A planar manipulator with 2 revolute joints ($2R$ manipulator).
3.8 Configuration space of the 2R manipulator. The white region shows the free space. The path of the robot in the configuration space from start to goal is shown by the dotted line. The points S and G are the start and goal configurations respectively. 

3.9 Variation of the joint angle rates of the 2R robot for motion from start to goal configuration in Figure 3.8.

3.10 (a) The applied torque at joint 1 used for dynamic simulation (b) The contact impulses transformed to joint 1 obtained from dynamic simulation using the input in subplot (a). (c) The final collision free input torque impulse for joint 1.

3.11 (a) The applied torque at joint 2 used for dynamic simulation (b) The contact impulses transformed to joint 2 obtained from dynamic simulation using the input in subplot (a). (c) The final collision free input torque impulse for joint 2.

3.12 Configuration space of the 2R manipulator. The white region shows the free space. The black squares show configurations along the collision-free trajectory. The positions of the manipulator in the workspace corresponding to the configurations are shown in Figures 3.13 and 3.14. The points S and G are the start and goal configurations respectively.

3.13 The first four configurations in the path of the 2R robot for motion from start to goal configuration in Figure 3.12.

3.14 The last four configurations in the path of the 2R robot for motion from start to goal configuration in Figure 3.12.

4.1 A dexterous manipulation task that requires closest distance computations to predict the contact points of fingers with an object. The fingers and object are represented as superquadrics.

4.2 Three example objects. The closest points of each pair of objects are shown connected by a line segment.

4.3 Schematic illustration of the interior point method for a path following algorithm. The convex region represents the feasible set. The central path is an arc of strictly feasible points that solve Equation 4.14 as the parameter $\mu$ approaches 0. The progress of the iterates generated by the interior point solver is indicated by the polygonal line connecting them. The iterates are guaranteed to lie within a neighborhood, represented by the circular ball, of the central path.

4.4 Example illustrating the sequence of closest point estimates generated by the interior point method for two 2D superquadric objects, with indices $(\frac{21}{11}, \frac{11}{5})$, $(\frac{26}{7}, \frac{71}{7})$ and semiaxes 1. The iterates of the interior point method are mapped to corresponding points in the objects.
4.5  Plot showing observed linear time behavior of the interior point algorithm for polyhedra. .................................................. 60
4.6  Plot showing observed linear time behavior of the interior point algorithm for quadrics. .................................................. 60
4.7  Plot showing observed linear time behavior of the interior point algorithm for superquadrics. ............................................. 61
4.8  Computing the instant of closest distance using the continuous proximity query. The bold blue line connects the closest points on the two objects, as they translate along the indicated line segments. ......................... 63
4.9  Computing the time of first contact using the continuous proximity query gives the solution to the continuous collision detection problem. ..................... 64
4.10 Example objects. Objects I–III are superquadrics, IV is an intersection of superquadrics and halfspaces, and V–VI are hyperquadrics. ......................... 65
4.11 Proximity queries on deforming (superquadric) objects, with the deformation described by monotonic scaling. The deformation is performed in 10 steps. (a) The original objects. (b) The objects midway through the scaling. (c) The scaled objects. .................................................. 69
4.12 Proximity queries on deforming superquadric objects, with the deformation governed by monotonic change of exponents. Object I is transformed to Object III in 10 steps. (a) The original objects. (b) Midway through the deformation, deformed Object I has indices \( \left( \frac{196}{45}, \frac{39}{5}, \frac{97}{13} \right) \). (c) The final objects. 69
5.1  Three Contact cases: (left) Objects are separate (middle) Objects are touching (right) Objects are intersecting. ........................................... 77
5.2  Schematic representation of the deflection at contact. The contact is where the dotted curves touch. ........................................... 83
5.3  Linear and angular velocities for Example 2. All velocities except \( \omega_z \) are zero throughout the simulation. ........................................... 88
5.4  Forces for Example 2. The tangential forces are both 0 for the entire simulation, and the torsional force transitions to zero when the sphere switches from a sliding contact to sticking. ........................................... 89
5.5  A small sphere in contact with two large spheres. ................................. 90
5.6  Velocities of small moving sphere. .................................................. 91
5.7  Force and sliding speed at contact 1. Contact 1 is always sliding until separation, hence the \( \mu \) normal force curve and friction magnitude curve overlap for the duration. The value of \( \mu = 0.2 \) ........................................... 92
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>Force and sliding speed at contact 2. The value of ( \mu = 0.2 )</td>
<td>93</td>
</tr>
<tr>
<td>5.9</td>
<td>Unit disc falling onto a frictionless compliant surface</td>
<td>93</td>
</tr>
<tr>
<td>5.10</td>
<td>Non-penetration force and spring force for a unit disc falling on a half-plane making contact with the rigid core.</td>
<td>98</td>
</tr>
<tr>
<td>5.11</td>
<td>Configuration and velocities for a unit disc falling on a half-plane making contact with the rigid core.</td>
<td>99</td>
</tr>
<tr>
<td>5.12</td>
<td>Energy plot for a unit disc falling on a half-plane making contact with the rigid core.</td>
<td>100</td>
</tr>
<tr>
<td>5.13</td>
<td>Energy and force plots for a unit disc falling on a half-plane without making contact with the rigid core.</td>
<td>101</td>
</tr>
<tr>
<td>5.14</td>
<td>Configuration and velocity plots for a unit disc falling on a half-plane without making contact with the rigid core.</td>
<td>102</td>
</tr>
<tr>
<td>5.15</td>
<td>Energy and force plots for an ellipsoid dropped onto a half-plane. The initial orientation of the ellipsoid is such that its major and minor axis are parallel to the inertial frame. Y-axis is set to ([-25, 100]) in force plot.</td>
<td>103</td>
</tr>
<tr>
<td>5.16</td>
<td>Configuration and velocity plots for an ellipsoid dropped onto a half-plane. The initial orientation of the ellipsoid is such that its major and minor axis are parallel to the inertial frame.</td>
<td>104</td>
</tr>
<tr>
<td>5.17</td>
<td>Force and Energy plots for an ellipsoid dropped onto a half-plane. Y-axis is set to ([-100, 100]) in force plot.</td>
<td>105</td>
</tr>
<tr>
<td>5.18</td>
<td>Configuration and Velocity plots for an ellipsoid dropped onto a half-plane.</td>
<td>106</td>
</tr>
<tr>
<td>5.19</td>
<td>Deformation of the compliant surface for an ellipsoid dropped onto a half-plane.</td>
<td>107</td>
</tr>
<tr>
<td>6.1</td>
<td>Schematic sketch of a 2-robot system used to process points in the plane. The heads can translate along the ( x )-axis and the base plate translates along the ( y )-axis. The square region of length ( 2\Delta ) is the processing footprint for each robot.</td>
<td>110</td>
</tr>
<tr>
<td>6.2</td>
<td>An example input distribution of points where the bold lines show an optimal 2-TSP tour on this set obtained ignoring the geometric constraints. Clearly, the two robots for the system in Figure 6.1 cannot process any pair of points simultaneously while satisfying the geometric constraints.</td>
<td>114</td>
</tr>
<tr>
<td>6.3</td>
<td>Splitting and assignment of points by greedy algorithm for the dataset of 1396 points.</td>
<td>121</td>
</tr>
</tbody>
</table>
6.4 Splitting and assignment of points by greedy algorithm for the dataset of 11109 points.

6.5 The left figure shows the TSP tour of robot 1 whereas the right figure shows the self crossings observed in the TSP tour of robot 2. The initial pairings were (1,a), (2,b), (3,c), (4,d) whereas the new pairing in the lower cost tour is (1,a), (2,c), (3,b), (4,d), provided the new pairs are compatible.

7.1 (1) Schematic sketch of case 1 (2) Schematic sketch of case 2 (3) Schematic sketch of case 3. The bold lines are the pairs in the greedy solution and the dotted lines are the pairs in the optimal solution.
EXTENDED ABSTRACT

This thesis focuses on the development of algorithms and tools used in robot motion planning. We consider two types of motion planning problems: (1) We first look at point-to-point motion planning for a single robot in the presence of geometric, kinematic, and dynamic constraints. (2) We then look at multiple robot path planning problems where the robots are required to visit a set of points in the presence of geometric constraints.

Point-to-point robot motion planning, i.e., obtaining control inputs to move the robot from one state to another, taking into consideration geometric, kinematic, and dynamic constraints is a fundamental problem in realizing autonomous robotic systems. Collision detection and dynamic simulation are two important modules that form an integral part of current sampling based randomized motion planners. The collision detection module ensures that the geometric constraints are satisfied and the dynamic simulation module ensures that the state evolution satisfy the differential constraints. Most research in sampling based motion planning algorithms treat these two modules as a black-box and use them to only obtain an input giving a feasible trajectory; an input is rejected if there is any collision along the trajectory. In the first part of this thesis, we show that using a complementarity based formulation of the dynamics, we can use the collision information to modify the applied inputs and obtain inputs that ensure a collision free trajectory. This is useful in applications where collision avoidance is the primary requirement. However, in the presence of intermittent contact between the robot and objects in the environment as seen in applications like grasping, manipulation, locomotion, the presence of contact makes accurate dynamic simulation challenging. Consequently, we study the sources of errors in current dynamic simulators.

The primary sources of stability and accuracy problems in state-of-the-art time steppers for multibody systems are (a) the use of polyhedral representations of smooth bodies, (b) the decoupling of collision detection from the solution of the dynamic time-stepping subproblem, and (c) errors in model parameters. We focus on formulations, algorithm development, and analysis of time-steppers to eliminate the first two error sources. As a partial solution to problem (a) above, we provide distance computation algorithms for
convex objects modeled as an intersection of implicit surfaces. The use of implicit surfaces to describe objects for dynamic simulation is impaired by the lack of algorithms to compute exact distances between implicit surface objects. In contrast to geometric approaches developed for polyhedral objects, we formulate the distance computation problem as a convex optimization problem and use a primal-dual interior point method to solve the Karush-Kuhn-Tucker (KKT) conditions obtained from the convex program. For the case of polyhedra and quadrics, we establish a theoretical time complexity of $O(n^{1.5})$, where $n$ is the number of constraints; in practice the algorithm takes linear time. We then provide solutions for problem (a) and (b) described above for simulating multibody systems with intermittent contact by incorporating the contact constraints as a set of complementarity and algebraic equations within the dynamics model. This enables us to formulate a geometrically implicit time-stepping scheme (i.e., we do not need to approximate the distance function) as a nonlinear complementarity problem (NCP). The resulting time-stepper is therefore more accurate; further it is the first geometrically implicit time-stepper that does not rely on a closed form expression for the distance function. We first present our approach assuming the bodies to be rigid and then extend it to locally compliant or quasi-rigid bodies. We demonstrate through example simulations the fidelity of this approach to analytical solutions and previously described simulation results. This distance computation and dynamic simulation work may also be of interest outside of robot motion planning to applications in mechanical design and haptic interaction.

For multiple robot systems, the task requirements can also lead to geometric constraints and the system performance depends on task allocation to the robots in the presence of geometric constraints. In the second part of this thesis we study such a problem, namely, path planning for multiple robots (say $K$) required to cover a point set in the presence of inter-robot geometric constraints so that the task completion time is minimized. Robotic point set coverage tasks occur in a variety of application domains like electronic manufacturing (laser drilling, inspection, circuit board testing), automobile spot welding, and data collection in sensor networks. We look at problems where the robots have to (a) spend some time at each point to complete a task (we call this time processing time of each point) and (b) satisfy given geometric constraints (like collision avoidance) while covering the point set. In the absence of the geometric constraints and assuming the pro-
cessing times to be zero, the path planning problem for multi-robot point set coverage tasks can be treated as a $K$-Traveling Salesman Problem ($K$-TSP). However the path planning problem with inter-robot geometric constraints hasn’t been treated extensively in the literature. As example application, we consider an industrial microelectronics manufacturing system with two robots, with square footprints, that are constrained to translate along a line while satisfying proximity and collision avoidance constraints. The $N$ points lie on a planar base plate that can translate along the plane normal to the direction of motion of the robots. The geometric constraints on the motions of the two robots lead to constraints on points that can be processed simultaneously.

We use a two step approach to solve the path planning problem: (1) **Splitting Problem**: Assign the points to the $K$ robots subject to geometric constraints, such that the total processing time is minimized. (2) **Ordering Problem**: Find an order of processing the split points by formulating and solving a multi-dimensional Traveling Salesman Problem (TSP) in the $K$-tuple space with an appropriately defined metric to minimize the total travel cost. For $K = 2$, the splitting problem can be converted to a maximum weighted matching (MWM) problem on a graph and solved optimally in $O(N^3)$ time. The matching algorithm takes $O(N^3)$ time in general and is too slow for large datasets. Therefore, we also provide a $O(N^2)$ time greedy algorithm and prove that the ratio of the total greedy processing time to optimal processing time is within $3/2$. For the special case where all the points have equal processing times we provide a $O(N \log N)$ time greedy algorithm. We provide computational results showing that the greedy algorithm solutions are very close to the optimal solution for typical industrial datasets. We also provide computational results for the ordering problem and propose local search based heuristics to improve the TSP tour. Further, we give computational results showing the overall performance gain obtained (over a single robot system) by using our algorithm. We also extend our approach to a $K$-robot system and give computational results for $K = 4$. 

xiv
CHAPTER 1
Introduction

Robots interacting with their environment need to take geometric, kinematic, and dynamic constraints into account for effective motion planning (e.g., moving without unintended collision with objects in their environment). For single robot systems, these geometric constraints arise from their own mechanical structure and the presence of other objects in the environment. For multiple robot systems, task requirements can also lead to geometric constraints on the allowable states of the robots (e.g., robots may be required to stay within a distance of each other for communication purposes). The mathematical models for the state evolution of such systems consist of systems of differential equations, algebraic constraints, and variables that jump between zero and non-zero values depending upon the state of the system, i.e., these systems are inherently non-smooth. Motion planning, i.e., obtaining the control inputs while satisfying the differential and algebraic constraints, for systems described by such models is challenging and dynamic simulation is used as a tool to aid in planning. Therefore, the performance of the planner depends on the quality of the simulation algorithms, which in turn depends on properly incorporating the geometric information in the dynamic state evolution model. For multiple robot systems, in addition to the quality of motion plans, the system performance depends on task allocation to the robots in the presence of geometric constraints. In this thesis, we use techniques from continuous and combinatorial optimization to properly incorporate the geometric constraints for effective robot motion planning. In the first part we look at the role of dynamic simulation algorithms in point-to-point motion planning, analyze the sources of errors in current dynamic simulation algorithms and provide solutions to key problems. In the second part, we study the problem of motion planning for multiple robots that are required to visit a set of points while satisfying given geometric constraints. In this coverage problem we must assign the points to the robots and specify an order in which the points should be visited in addition to solving the point-to-point motion planning. We provide solutions for the problem of assigning the points and specifying the order of visiting the points while ensuring that the geometric constraints are satisfied.
1.1 Thesis Outline and Contributions

The first part of the thesis uses concepts from rigid body dynamics and complementarity theory in mathematical optimization. Therefore, in Chapter 2 we provide the basic definitions and present the complementarity based instantaneous model of rigid body dynamics. The complementarity equations model the geometric unilateral collision (contact) constraints. They encode the physical constraint that two objects cannot interpenetrate, i.e., there will be a non-zero contact force between two bodies if the distance between them is zero and zero contact force if the distance is non-zero. We present the discrete dynamics model using an Euler time stepping scheme for discretization. Subsequently, we outline the common assumptions that are made when integrating the numerical equations. These are assumptions on the description of object geometry and the explicit or implicit incorporation of geometric information in the discrete formulation. We present some simple examples to analytically show the effect of including the geometric information explicitly in the integration of the equations of motion. We also use a simple example of a disc rolling without slip on a flat plane to illustrate numerical energy loss when (a) we use a polyhedral description of the smooth geometry and (b) use the contact point at the previous time step to determine the contact normal and distance between the plane and the disc (i.e., the geometric information is explicit in the time-stepper). Although these assumptions are very common in numerical integration of the equations of motion of a rigid body, as far as we know there has been no previously published study which illustrates the artifacts produced by these assumptions.

In Chapter 3, we look at the role of the dynamic simulation algorithms in motion planning problems with differential constraints where collision avoidance is the key requirement. We present the basic random sampling based algorithm in which (a) the dynamic simulation module is used to only get inputs that ensure that the state evolution satisfy the differential constraints, and (b) a collision detection module is used for verifying that the trajectory given by the dynamic simulation module is collision free (i.e., satisfies the geometric constraints). We show that (assuming a polyhedral model for the input geometry of the objects) if we use complementarity based models for dynamic simulation, and simulate for a time $\Delta t$ using a given input we will get non-zero contact force values when the objects just touch (we can use a safety distance so that the forces be-
come non-zero when they are a safe distance apart) in that time interval. The end state at time $t + \Delta t$ is collision free. Moreover, we can obtain the input that guarantees that the whole path is collision free by adding the (suitably transformed) virtual contact forces to the input forces. This has the following advantages: (a) We always get a collision-free path and the corresponding input forces (provided the input forces do not violate actuator force constraints) and unlike the conventional method we do not waste any computation when the path is not collision free. (b) When the input set is a set of motion primitives, it may be the case that there is no input in this primitive set that gives a feasible path for that time step. Using our method we can obtain feasible inputs that are outside the set of primitives, thus essentially enhancing the input set available for planning, even if we start with a simple set of hand designed primitives.

The collision detection module that was presumed to be available in Chapter 3 dealt with a polyhedral description of the geometry of the bodies. The main reason for using polyhedral geometry is the lack of efficient algorithms for the distance computation between objects defined by smooth surfaces. Therefore in Chapter 4 we look at the distance computation problem between convex objects described as an intersection of implicit surfaces$^1$. In contrast to geometric approaches developed for polyhedral objects, we formulate the distance computation problem as a convex optimization problem. We use an interior point method to solve the optimization problem and demonstrate that for general convex objects represented as implicit surfaces, interior point approaches are globally convergent, and fast in practice. Further, they provide polynomial-time guarantees for implicit surface objects when the implicit surfaces have self-concordant barrier functions. We use a primal-dual interior point algorithm that solves the KKT conditions obtained from the convex programming formulation. For the case of polyhedra and quadrics, we establish a theoretical time complexity of $O(n^{1.5})$, where $n$ is the number of constraints. We present implementation results for example implicit surface objects, including polyhedra, quadrics, and generalizations of quadrics such as superquadrics and hyperquadrics, as well as intersections of these surfaces. We demonstrate that in practice, the algorithm takes time linear in the number of constraints, and that distance computation rates of about 1 kHz can be achieved. We also extend the approach to proximity queries between de-

$^1$This chapter is joint work with Jufeng Peng.
forming convex objects. Finally, we show that continuous collision detection for linearly translating objects can be performed by solving two related convex optimization problems. For polyhedra and quadrics, we establish that the computational complexity of this problem is also $O(n^{1.5})$.

In principle, we can also use the dynamic simulation based randomized algorithms for motion planning outlined in Chapter 3 to problems where intermittent contact occurs with objects in the environment. However, as illustrated in Chapter 2, the presence of contact makes accurate dynamic simulation much harder compared to cases where contact is absent. The primary sources of stability and accuracy problems in state-of-the-art time steppers for multibody systems are: (a) the use of polyhedral representations of smooth bodies, (b) the decoupling of collision detection from the solution of the dynamic time-stepping subproblem, and (c) errors in model parameters. In Chapter 5 we focus on formulations and analysis of time-steppers to eliminate the first two error sources for simulating multibody systems with intermittent contact. We incorporate the contact constraints as a set of complementarity and algebraic equations within the dynamics model. We assume the input objects to be convex objects described by intersection of implicit surfaces. We write the contact constraints as complementarity constraints between the contact force and a distance function dependent on the closest points on the objects. The closest points satisfy a set of algebraic constraints obtained from the KKT conditions of the minimum distance problem. These algebraic equations and the complementarity constraints taken together ensure satisfaction of the contact constraints. This enables us to formulate a geometrically implicit time-stepping scheme (i.e., we do not need to approximate the distance function) as a nonlinear complementarity problem (NCP). The resulting time-stepper is therefore more accurate; further it is the first geometrically implicit time-stepper that does not rely on a closed form expression for the distance function. We first present our approach assuming the bodies to be rigid and then extend it to locally compliant or quasi-rigid bodies. We demonstrate through example simulations the fidelity of this approach to analytical solutions and previously described simulation results.

The second part of this thesis (Chapter 6 and 7) deals with path planning for multiple robot point set coverage in the presence of inter-robot geometric constraints. Robotic
point set coverage tasks occur in a variety of application domains including electronic manufacturing (laser drilling, inspection, circuit board testing), automobile spot welding, and data collection in sensor networks. The goal of using multiple robots in point set coverage tasks is to reduce the overall task completion time by parallelizing the operations at the points. The path planning problem in such multi-robot point set coverage tasks can be stated as follows: Given a point set, \( S = \{p_i\}, \ i = 1, \ldots, N, \) and \( K \) robots, find an assignment of the points to individual robots and determine the order in which the robots must visit the points so that the overall task completion time is minimized. We look at such path planning problems for multiple robot point set coverage where the robots have to (a) spend some time at each point to complete a task (we call this time the processing time of each point) and (b) satisfy given geometric constraints (like collision avoidance) while covering the point set. More concretely, our work is motivated by an industrial microelectronics manufacturing system with two robots, with square footprints, that are constrained to translate along a line while satisfying proximity and collision avoidance constraints. The \( N \) points lie on a planar base plate that can translate along the plane normal to the direction of motion of the robots. The geometric constraints on the motions of the two robots lead to constraints on points that can be processed simultaneously. In the absence of the geometric constraints and assuming the processing times to be zero, the path planning problem for multi-robot point set coverage tasks can be treated as a \( K \)-Traveling Salesman Problem (\( K \)-TSP). However the path planning problem with inter-robot geometric constraints has not been treated extensively in the literature. We use a two step approach to solve the path planning problem: (1) Splitting Problem: Assign the points to the \( K \) robots subject to geometric constraints, such that the total processing time is minimized. (2) Ordering Problem: Find an order of processing the split points by formulating and solving a multi-dimensional Traveling Salesman Problem (TSP) in the \( K \)-tuple space with an appropriately defined metric to minimize the total travel cost.

In Chapter 6, we consider the problem where all points have identical processing time. We show that for \( K = 2 \), the splitting problem can be converted to a maximum cardinality matching problem on a graph and solved optimally in polynomial time. The matching algorithm takes \( O(N^3) \) time in general and is too slow for large datasets. Therefore, we also provide a greedy algorithm for the splitting problem that takes \( O(N \log N) \)
time. We provide computational results comparing the two approaches and show that the greedy algorithm is very close to the optimal solution for large datasets. We also provide computational results for the ordering problem and propose local search based heuristics to improve the TSP tour. Further, we give computational results showing the overall performance gain obtained (over a single robot system) by using our algorithm. We also extend our approach to a $K$-robot system and give computational results for $K = 4$.

In Chapter 7, we consider the problem when the points may have different processing times. When the processing times are different, we show that the splitting problem can be converted to a maximum weighted matching (MWM) problem on a graph and solved optimally in $O(N^3)$ time. However this is too slow for large datasets and we also provide a $O(N^2)$ time greedy algorithm and prove that the ratio of the total greedy processing time to optimal processing time is less than $3/2$. Moreover, we demonstrate with an example that this is a tight bound for our algorithm. We also provide computational results for our algorithm on typical industrial datasets, which show that the greedy solution is very close to the optimal solution in practice.
CHAPTER 2
Rigid Body Dynamics: Complementarity-Based Modeling

2.1 Introduction

A rigid body is a system of point masses such that the distance between all pairs of points remains constant through any motion of the system. It is a useful model for objects in many physical situations in robotics where object deformation can be neglected. In applications, one is usually concerned with the motion of a collection of rigid bodies in a (possibly bounded) subset of $\mathbb{R}^3$ or $\mathbb{R}^2$. The rigid bodies have to satisfy the constraints that they cannot interpenetrate (unilateral constraints) and may also be connected to each other by joints (bilateral constraints). We call such a collection of rigid bodies a multi-rigid-body system. The dynamics of multi-rigid-body systems with unilateral contacts can be modeled as differential algebraic equations (DAE) [50] if the contact interactions (sliding, rolling, or separating) at each contact are known. However, in general, the contact interactions are not known a priori, but rather are discovered as part of the solution process. To handle the many possibilities in a rigorous theoretical and computational framework, the problem is formulated as a differential complementarity problem (DCP) [27, 107]. The differential complementarity problem is solved using a time-stepping scheme and the resultant system of equations to be solved at each step is a mixed (linear/nonlinear) complementarity problem.

Let $u \in \mathbb{R}^{n_1}$, $v \in \mathbb{R}^{n_2}$ and let $g : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{n_1}$, $f : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{n_2}$ be two vector functions and the notation $0 \leq x \perp y \geq 0$ imply that $x$ is orthogonal to $y$ and each component of the vectors is non-negative.

**Definition 1** [81] The differential (or dynamic) complementarity problem is to find $u$ and $v$ satisfying

$$
\dot{u} = g(u, v), \quad u, v, f(u, v) \geq 0
$$
Definition 2 The mixed complementarity problem is to find $u$ and $v$ satisfying

$$g(u, v) = 0, \quad u, \text{free}$$

$$0 \leq v \perp f(u, v) \geq 0$$

If the functions $f$ and $g$ are linear the problem is called a mixed linear complementarity problem (MLCP), otherwise, the problem is called a mixed nonlinear complementarity problem (MNCP).

2.2 Dynamic Model for Rigid Body Systems

In complementarity methods, the instantaneous equations of motion of a rigid multibody system consist of five parts: (a) equations describing state evolution, (b) kinematic map relating the generalized velocities to the linear and angular velocities, (c) equality constraints to model joints, (d) normal contact condition to model intermittent contact, and (e) friction law. Parts (a) and (b) form a system of ordinary differential equations, (c) is a system of (nonlinear) algebraic equations, (d) is a system of complementarity constraints, and (e) can be written as a system of complementarity constraints for any friction law that obeys the maximum work dissipation principle. In this chapter we use an elliptic dry friction law [108]. Thus, the dynamic model is a differential complementarity problem. To solve this system of equations, we have to set up a discrete time-stepping scheme and solve a complementarity problem at each time step. We present below the continuous formulation as well as an Euler time-stepping scheme for discretizing the system. To simplify the exposition, we ignore the presence of joints or bilateral constraints in the following discussion. However, all of the discussion below holds in the presence of bilateral constraints.

To describe the dynamic model mathematically, we first introduce some notation. Throughout this thesis, unless otherwise stated, we will use $j$ as an index varying over the number of bodies ($n_b$) and $i$ as an index varying over the number of contacts ($n_c$). Let $Q_j$ be the configuration space of the body (or robot) $j$ and $X_j$ be the state space. Let $x_j = (q_j, \nu_j)$ be the state with $q_j \in Q_j$ the configuration of the body in an inertial frame and $\nu_j$ the generalized velocity. For example, if the body $j$ is a rigid body then $q_j$ is the
concatenated vector of the position and orientation and $\nu_j$ is the concatenated vector of linear velocities $v_j$ and angular velocities $\omega_j$. If the body $j$ is a multi-link robot (say, a serial chain robot) with a reduced coordinate representation of the configuration space (say, the angles of the joints), $q_j$ is the vector of the joint angles and $\nu_j$ is the vector of the joint angle rates ($\dot{q}_j$). We form the vector $q$ and $\nu$ of the whole system by concatenating $q_j$ and $\nu_j$ respectively. The concatenated vector $(q, \nu)$ represents the state of the system.

The symbols $Q$ and $X$ are used for the configuration space and state space of the whole system, where $Q$ is the Cartesian product of the individual configuration spaces $Q_j$ and $X$ is the Cartesian product of the individual state spaces $X_j$. Let $\lambda_{in}$ be the magnitude of the normal contact force at the $i$th contact and $\lambda_n$ be the concatenated vector of the normal contact force magnitudes. Let $\lambda_{it}$ and $\lambda_{io}$ be the magnitudes of orthogonal components of the friction force on the tangential plane at the $i$th contact and $\lambda_t$, $\lambda_o$ be the respective concatenated vectors. Let $\lambda_{ir}$ be the frictional moment magnitude about the $i$th contact normal and $\lambda_r$ be the concatenated vector of the frictional moments. The instantaneous dynamic model can then be written by combining the following components.

**State Evolution Equations:** The state evolution of a rigid-multi-body system is given by

$$M(q)\dot{\nu} = F_{cont} + F_{app} + F_{vp}$$

where $M(q)$ is the symmetric positive-definite mass matrix, $F_{vp}$ is the vector of generalized Coriolis and centripetal forces, $F_{cont}$ is the vector of generalized forces due to the contact forces and moments, $F_{app}$ is the vector of generalized forces due to applied forces and moments (that includes forces due to gravity and actuator inputs).

**Example 1:** If the multi-body system consists of a collection (say $n_b$) of freely moving rigid bodies in $\mathbb{R}^3$, the matrix $M(q)$ is a block diagonal matrix of size $6n_b \times 6n_b$. Each $6 \times 6$ block of $M(q)$ is the mass matrix of a single rigid body. The vector $\nu$ is of size $6n_b \times 1$, since a rigid body has 6 degrees of freedom. The configuration $q_j$ of the $j$th body may be a $6 \times 1$ vector or $7 \times 1$ vector depending on the representation of orientation (it is $7 \times 1$ for unit quaternion representation of orientation, that we will use throughout this thesis). The contact force vector $F_{cont}$ is of size $6n_b \times 1$ formed by concatenating $n_b$
vectors of size $6 \times 1$. Each $6 \times 1$ vector is a concatenated vector of forces and moments\(^3\) at the center of mass of each body due to all the contact forces and moments acting on the body.

**Example 2:** If the multi-body system is a $n-$degree of freedom (DoF) serial chain robot and the configuration space is parameterized by the $n$ joint displacements, the matrix $M(q)$ is of size $n \times n$ and the vectors $q$ and $\nu$ are of size $n \times 1$. The vector $F_{\text{cont}}$ is an $n \times 1$ vector formed by projecting the contact forces and moments to the space of generalized coordinates. The details of this projection are given later in the discussion on non-penetration constraints.

**Kinematic Map:** Generically, for the $j$th body, the relationship between the time derivative of the generalized coordinates, $\dot{q}_j$ and the generalized velocity vector $\nu_j$ can be written as

$$\dot{q}_j = G_j(q_j)\nu_j \quad (2.2)$$

The Jacobian $G_j$ may be a non-square matrix but $G_j^T G_j = I$. For example, if the body $j$ is a rigid body, $G_j$ is a $6 \times 6$ matrix if we use Euler angles to parameterize orientation, whereas it is a $7 \times 6$ matrix if we use unit quaternions. If the body $j$ is a $n-$DoF serial-chain robot, the matrix $G$ is a $n \times n$ identity matrix.

**Nonpenetration Constraints:** The contact forces $F_{\text{cont}}$ in Equation 2.1 are usually unknown and have to be determined as part of the solution process. The magnitude of the normal component of the contact forces is zero if the distance between the two bodies is greater than zero and non-zero if the bodies are in contact. Thus, for the $j$th body, at each potential contact, the product of the magnitude of the normal contact force and distance between the two objects is always zero. This is encoded by the following complementarity constraint (we drop the index $j$ representing the body for convenience)

$$0 \leq \lambda_{\text{in}} \perp \psi_{\text{in}}(q, t) \geq 0 \quad (2.3)$$

\(^3\)More formally, it is the wrench acting at the center of mass of the body due to the contact wrenches.
where $\psi_{in}$ is a signed distance function or *gap function* for the $i$th contact with the property $\psi_{in}(q,t) > 0$ for separation, $\psi_{in}(q,t) = 0$ for touching, and $\psi_{in}(q,t) < 0$ for interpenetration. The above gap function is defined in the configuration space of the system. Note that there is usually no closed form expression for $\psi_{in}(q,t)$.

Let $(\hat{n}_i, \hat{t}_i, \hat{o}_i)$ be the contact frame at the $i$th contact of body $j$ expressed in the world frame. The wrench due to the normal contact force acting, at any point on the body (say the center of gravity) is $\hat{w}_{in}\lambda_{in}$, where $\hat{w}_{in}$ is a $6 \times 1$ concatenated vector of the negative of the unit normal at the contact point, $-\hat{n}_i$, and $-r_i \times \hat{n}_i$, $r_i$ being the vector from the center of gravity to the contact point. This wrench transformed to generalized force is

$$J_i^T \hat{w}_{in}\lambda_{in} \quad (2.4)$$

where $J_i$ is the transformation Jacobian between the twist of a frame attached to the cg and the rate of change of the generalized coordinates. For example, if the $j$th body is a serial-chain manipulator $J_i$ is the manipulator Jacobian up to the cg of the link in contact. For a rigid body, $J_i$ is the $6 \times 6$ identity matrix. Similar expressions can be written for the wrench due to the tangential contact forces by replacing the subscript $n$ in equation 2.4 by $t, o$ with $\hat{w}_{ik} = (\hat{k}_i, -r_i \times \hat{k}_i), k = t, o$ a $6 \times 1$ vector. The contact moment can also be written in the form of equation 2.4 with the $6 \times 1$ vector $\hat{w}_{ir} = (0_{3 \times 1}, -\hat{n}_i)$. The total generalized force on the $j$th body due to the $i$th contact is

$$F_{cont} = \sum_i J_i^T \sum_{k \in \{n,t,o,r\}} \hat{w}_{ik}\lambda_{ik} \quad (2.5)$$

Denoting $J_i^{ht}atw_{ik}$ by $w_{ik}, k \in \{n,t,o,r\}$, the contact force on the $j$th body can be written as

$$F_{cont}^{(j)} = \sum_i F_{cont,i} = \sum_i \sum_{k \in \{n,t,o,r\}} w_{ik}\lambda_{ik} \quad (2.6)$$

In a more compact matrix form the generalized force $F_{cont}$ in equation 2.1 is

$$F_{cont} = W_n\lambda_n + W_t\lambda_t + W_o\lambda_o + W_r\lambda_r \quad (2.7)$$
Friction Model  The tangential forces and contact moment at each contact should satisfy a given friction model. We use a generalization of the Coulomb friction model in this thesis which is also known as a soft finger contact model in robotics literature [79]. The magnitudes of the contact forces and moments can be obtained as a solution of the following optimization problem (a generalization of Moreau’s maximum dissipation principle that also appeared in [108]).

\[
\begin{align*}
\text{max} & \quad - (v_{it} \lambda_{it} + v_{io} \lambda_{io} + v_{ir} \lambda_{ir}) \\
\text{s.t.} & \quad \left( \frac{\lambda_{it}}{e_{it}} \right)^2 + \left( \frac{\lambda_{io}}{e_{io}} \right)^2 + \left( \frac{\lambda_{ir}}{e_{ir}} \right)^2 - \mu^2_i \lambda^2_{in} \leq 0
\end{align*}
\]

where \(v_{it}, v_{io}\) are the relative tangential contact velocities, \(v_{ir}\) is the relative contact angular velocity about the common normal, \(e_{it}, e_{io}\) and \(e_{ir}\) are given positive constants defining the friction ellipsoid and \(\mu_i\) is the coefficient of friction at the \(i^{th}\) contact. We now proceed to derive the equations that the solutions of this optimization problem must satisfy. This has appeared elsewhere in the literature and we include it here for completeness. The Fritz-John optimality conditions that the friction forces and moments must satisfy are given by:

\[
\begin{align*}
0 & \leq - \left( \frac{\lambda_{it}}{e_{it}} \right)^2 + \left( \frac{\lambda_{io}}{e_{io}} \right)^2 + \left( \frac{\lambda_{ir}}{e_{ir}} \right)^2 - \mu^2_i \lambda^2_{in} \quad \perp u_2 \geq 0 \\
u_1 & \geq 0, \quad (u_1, u_2) \neq (0, 0)
\end{align*}
\]

When \(u_2 = 0, u_1 > 0\) and the left part of the complementarity constraint in equation 2.12 is 0. Using the value of \(\lambda_{it}, \lambda_{io}, \lambda_{ir}\) from equations 2.9, 2.10, 2.11, we have

\[
u_1^2(v_{it}^2 e_{it}^2 + v_{io}^2 e_{io}^2 + v_{ir}^2 e_{ir}^2) = 4u_2^2 \mu^2_i \lambda^2_{in}
\]

\[
\Rightarrow \quad u_1 = \frac{2u_2 \mu \lambda_{in}}{\sqrt{v_{it}^2 e_{it}^2 + v_{io}^2 e_{io}^2 + v_{ir}^2 e_{ir}^2}}
\]
Substituting the values of $u_3$ in equations 2.9 to 2.11 we have

\[
\begin{align*}
\mu_i e_{it}^2 \lambda_{in} v_{it} + \sigma \lambda_{it} &= 0 \quad (2.15) \\
\mu_i e_{io}^2 \lambda_{in} v_{io} + \sigma \lambda_{io} &= 0 \quad (2.16) \\
\mu_i e_{ir}^2 \lambda_{in} v_{ir} + \sigma \lambda_{ir} &= 0 \quad (2.17)
\end{align*}
\]

where \( \sigma = \sqrt{v_{it}^2 e_{it}^2 + v_{io}^2 e_{io}^2 + v_{ir}^2 e_{ir}^2} \)

Now, we have to write the complementarity conditions in equation 2.12 in terms of $\sigma$. When $u_2 > 0$, the right hand side of equation 2.13 is positive, hence $\sigma^2$ is positive, implying $\sigma > 0$. When $u_2 = 0$, we have $u_1 > 0$, and from equations 2.9, 2.10 and 2.11, we have $v_{it} = 0$, $v_{io} = 0$, and $v_{ir} = 0$. This implies that $\sigma = 0$. Therefore we can replace $u_2$ in equation 2.12 with $\sigma$ and the system of equations that the contact forces and moments must satisfy are:

\[
\begin{align*}
\mu_i e_{it}^2 \lambda_{in} v_{it} + \sigma_i \lambda_{it} &= 0 \quad (2.18) \\
\mu_i e_{io}^2 \lambda_{in} v_{io} + \sigma_i \lambda_{io} &= 0 \quad (2.19) \\
\mu_i e_{ir}^2 \lambda_{in} v_{ir} + \sigma_i \lambda_{ir} &= 0 \quad (2.20)
\end{align*}
\]

\[
0 \leq - \left( \left( \frac{\lambda_{it}}{e_{it}} \right)^2 + \left( \frac{\lambda_{io}}{e_{io}} \right)^2 + \left( \frac{\lambda_{ir}}{e_{ir}} \right)^2 - \mu^2 \lambda_{in}^2 \right) \perp \sigma_i \geq 0 \quad (2.21)
\]

### 2.2.1 Discrete Time Model

We now write down the discretized equations of motion for numerical integration. We use a velocity-level formulation and an Euler time-stepping scheme to discretize the above system of equations. Let $t_\ell$ denote the current time, and $h$ be the time step. Use the superscripts $\ell$ and $\ell + 1$ to denote quantities at the beginning and end of the $\ell$th time step respectively. Using $\dot{\nu} \approx (\nu^{\ell+1} - \nu^\ell)/h$, $\dot{q} \approx (q^{\ell+1} - q^\ell)/h$, and $h\lambda_{(\cdot)} = p_{(\cdot)}$ we get the
following discrete time system.

\[ M\nu^{\ell+1} = M\nu^\ell + W_n p_n^{\ell+1} + W_t p_t^{\ell+1} + W_o p_o^{\ell+1} + W_r p_r^{\ell+1} + \lambda_{app} + p_{vp} \]

\[ q^{\ell+1} = q^\ell + h G\nu^{\ell+1} \]

\[ 0 \leq p_n^{\ell+1} \perp \psi_n(q^{\ell+1}) \geq 0 \]

\[ \mu_i e_i^2 p_{it} v_{it} + \sigma_i p_{it} = 0 \]

\[ \mu_i e_i^2 p_{io} v_{io} + \sigma_i p_{io} = 0 \]

\[ \mu_i e_r^2 p_{ir} v_{ir} + \sigma_i p_{ir} = 0 \]

\[ 0 \leq - \left( \left( \frac{p_{it}}{e_{it}} \right)^2 + \left( \frac{p_{io}}{e_{io}} \right)^2 + \left( \frac{p_{ir}}{e_{ir}} \right)^2 - \mu_i^2 p_{in}^2 \right) \perp \sigma_i \geq 0 \] (2.22)

The constraints coming from the friction law can be alternatively expressed in vector notation [108]:

\[ E_t^2 U p_n \circ W_t^T \nu^{\ell+1} + p_t \circ \sigma = 0 \]

\[ E_o^2 U p_n \circ W_o^T \nu^{\ell+1} + p_o \circ \sigma = 0 \]

\[ E_r^2 U p_n \circ W_r^T \nu^{\ell+1} + p_r \circ \sigma = 0 \] (2.23)

\[ (U p_n) \circ (U p_n) 0 \leq - (E_t^2)^{-1} (p_t \circ p_t) - (E_o^2)^{-1} (p_o \circ p_o) \]

\[ - (E_r^2)^{-1} (p_r \circ p_r) \perp \sigma \geq 0 \]

where the impulse \( p_{(i)} = h \lambda_{(i)} \), the matrices \( E_t, E_o, E_r \), and \( U \) are diagonal with \( i \)th diagonal element equal to \( e_{ti}, e_{oi}, e_{ri} \), and \( \mu_i \) respectively, \( \sigma \) is a concatenated vector of the Lagrange multipliers arising from the conversion from the argmax formulation and \( \sigma_i \) is equal to the magnitude of the slip velocity at contact \( i \), and \( \circ \) connotes the Hadamard product.

The subproblem at each time step given by equation 2.22 is either an MLCP or an MNCP depending on the time evaluation of \( W_{(i)} \) the approximation used for \( \psi_n(q^{\ell+1}) \), and the representation of the friction model. If \( W_{(i)} \) are evaluated at \( \ell \), and we use a first order Taylor series expansion for \( \psi_n(q) \) and a linearized representation of the friction ellipsoid, we have an LCP. However, the linearization of the distance function introduce simulation artifacts as discussed in the section below. Moreover, the linear approximation
of the friction ellipsoid also leads to certain artifacts. In contrast, if we evaluate $W(\cdot)$ at $\ell + 1$, use a quadratic friction law (Equation (2.23)), and use $\psi_n(q^{\ell+1})$, we have an NCP. We call this formulation a geometrically implicit formulation because it ensures that the contact conditions are satisfied at the end of the time step. However, evaluating $\psi_n(q^{\ell+1})$ is possible only if we have a closed form expression for the distance function, which we do not have in general. Instead, we propose to define the gap function in terms of the closest points between the two objects and provide a set of algebraic equations for finding these closest points during the time step. Chapter 5 discusses this approach in detail and proves that the conditions will enforce satisfaction of contact constraints at the end of the time step.

2.3 Effect of Linearizing the Distance Function

In this section, we discuss the effects of linearization of the distance function. We first provide a few simple examples where we explain analytically the reason for error in the simulations. Subsequently, we provide a simulation of a thin disk rolling on a flat plane without slip, to illustrate the effects of the distance function linearization numerically.

Let us consider two objects in relative motion, with $q$ the concatenated vectors of the generalized coordinates of the two bodies. The distance function between the two bodies is $\psi(q)$ (neglecting the explicit dependence on time $t$ here, to keep the discussion simple). In a discrete time setting, let $q^\ell$ and $q^{\ell+1}$ be the configurations of the two objects at two consecutive discrete time instances and let $h$ be the time step. A first-order approximation of the distance function is given by

$$\psi(q^{\ell+1}) = \psi(q^\ell) + h\frac{\partial\psi}{\partial q}\bigg|_{q^\ell} \nu^{\ell+1}$$  \hspace{1cm} (2.24)

The second term above is the product of the time-step ($h$) and the projection of the relative velocity at time $\ell + 1$ on the common normal at time $\ell$. Thus the first order approximation of the distance at time $\ell + 1$ is the sum of the distance at time $\ell$ and the projection of the relative movement during the time step on the common normal at time $\ell$. Thus, the collision detection algorithm should at least provide us with the closest features (contact point
when the objects are in contact) at time $\ell$ and the common normal through the closest features. There are many collision detection algorithms that return this information so it is safe to assume that we know (a) closest points at time $\ell$ to every convex piece (assuming that the object is decomposed into convex sub-pieces which is also the assumption collision detection algorithms make) (b) the common normal connecting the closest points at time $\ell$.

For two convex objects (or convex pieces) the second term on the right hand side of equation 2.24 is

$$h([\hat{n}^T(r_1 \times \hat{n})]^T \nu_1^{\ell+1} - [\hat{n}^T(r_2 \times \hat{n})]^T \nu_2^{\ell+1})$$

(2.25)

where $\nu_i, i = 1, 2$ are the $6 \times 1$ vector of concatenated linear and angular velocity vectors of body $i$, $r_i$ is the vector from the center of gravity of the $i$th body to the closest point at time $\ell$. The quality of the linear approximation depends on how accurately the equation 2.25 captures the actual relative motion between the two objects. In other words, the goodness of the approximation depends on how well the terms $\hat{n}$ and $r_i$ are approximated at $\ell + 1$ by evaluating them with the value of $q$ at time $\ell$, i.e., $q^{\ell}$. An elementary fact is that the linear approximation is only valid near a neighborhood of $q^{\ell}$. This implies that we can assume that the time step is small enough that $q^{\ell+1}$ is nearby $q^{\ell}$. Within this assumption of small perturbations we now want to understand the effects of approximating $\hat{n}$ and $r$ with a couple of examples.

We first try to understand the effect of approximating $\hat{n}$. Let us consider the simple example of a point mass moving toward a fixed object in the plane (Figure 2.1 shows different cases where the objects are of different shapes). Here $q = (x, y), \nu = (v_x, v_y), G = I, r = 0$. Thus, all linearization error will be due to the error in approximating $\hat{n}$. We assume that the collision detection algorithm returns us the closest points and the normal (that is the line joining the closest points when it is not well defined as in case (c) in the figure 2.1). The sum of the distance at time $\ell$ and the projection of the relative motion during the time step becomes 0 for any point lying on the dashed line. Therefore the approximation of the distance function evaluates to 0 for any point on the line and the complementarity constraint would allow a non-zero contact force if the particle reaches the line and restrict the particle to this dashed line. This implies that for cases (a), (c), and (d), the distance function may falsely indicate a collision and for case (b) it may falsely
indicate interpenetration, i.e., we may not detect collision even when it occurs.

Figure 2.1: Schematic sketch of different situations of a point mass moving towards a fixed object (a) Curved convex object (b) Curved non-convex object (c) The normal at the closest point on the triangle is not uniquely defined; the line joining the closest point is one choice (d) The particle at time $\ell + 1$ cannot cross the dashed line, since the projection on the old normal added to the old distance becomes 0, even though the actual distance is non-zero at $\ell + 1$.

To understand the effect of approximating $r$ numerically, we consider the example in Figure 2.2 showing a bar moving towards a flat surface. In this case, $q = (x, y, \theta)$, with $(x, y)$ the position of the center of gravity of the bar and $\theta$ the orientation of the bar, $\nu = (v_x, v_y, \omega)$, $G$ is identity matrix, and $\hat{n}$ is constant. However, $r$ is a function of the closest point. The right end of the bar is the closest point at time $\ell$ and so $r$ evaluated at time $\ell$ will be the vector from the center to the right end of the bar. If the bar rotates counterclockwise, $h(\nu^{\ell+1} + \omega^{\ell+1} \times r^{\ell})$ gives the displacement of the right end of the bar. The linearized distance function is the sum of the distance of the right end at time $\ell$ added to the displacement above. The linearized distance estimate at time $\ell + 1$ thus remains positive. However, the left end of the bar may actually hit the ground at time $\ell + 1$. Thus,
although $\Delta q$ may be small, the error in $r$ may be quite large (the vector $r$ at time $\ell + 1$ is one joining the cg to the left end of the bar), resulting in missing a collision.

![Figure 2.2: A bar approaching the fixed surface. At the configuration shown the right end of the bar is very near the surface. The bar has a linear velocity toward the surface and a counterclockwise angular velocity. In this case the collision at the left end of the bar may be missed due to the error in evaluating $r$.](image)

To illustrate the effects of geometric approximation, consider the simple planar problem of a uniform disc rolling on a horizontal support surface. For this problem, the exact solution is known, i.e., the disc will roll at constant speed *ad infinitum*. However, when the disc is approximated by a uniform regular polygon, energy is lost a) due to collisions between the vertices and the support surface, b) due to contact sliding that is resisted by friction, and c) due to artificial impulses generated by the approximate distance function that is to be satisfied at the end of the time-step. We simulated this example in dVC [6] using the Stewart-Trinkle time-stepping algorithm [106]. The parametric plots in Figures 2.3 and 2.3 show the reduction of kinetic energy over time caused by the accumulation of these effects. Figure 2.3 shows that increasing the number of edges, with the step-size fixed, decreases the energy loss; the energy loss approaches a limit determined by the size of the time-step. Figure 2.3 shows reducing energy loss with decreasing step size, with the number of vertices fixed at 1000. However, even with the decrease in time-step an energy loss limit is reached. These plots make it clear that the discretization of geometry and linearization of the distance function lead to the artifact of loss in energy in some simulations. Note that there may also be cases where it may lead to increase in energy in some simulations (as would be the case for the example in Figure 2.1(b)).
Figure 2.3: Reduction of kinetic energy over time for a rolling disc approximated as a uniform regular polygon. As the number of edges of the polygon increases, the energy loss decreases. The computed value obtained by our time-stepper using an implicit surface description of the disc is the horizontal line at the top. The time step used is 0.01 seconds.

Figure 2.4: Effect of the time step on the loss of kinetic energy for a polygon modeled by a fixed number of edges (1000 in this figure). The energy loss decreases with decreasing step size, up to a limit. In this case, the limit is approximately 0.001 seconds (the plots for 0.001, 0.0005, and 0.0001 are indistinguishable).
2.4 Conclusion

In this chapter, we reviewed the complementarity based model of the equations of motion of a multi-rigid-body system in intermittent contact with each other. The formulation presented here is general and is valid for any parameterization of the configuration space of the system. We showed that although the satisfaction of the complementarity constraints is necessary and sufficient for contact determination in the continuous model, in the discrete model it may not be the case. We showed by analysis that a linearization of the distance function in the complementarity constraints that is commonly used in numerical simulations may result in false detection of contact or penetration between the bodies. We also showed numerically by simulating a disc rolling on a plane that the linearization of the distance function leads to artificial loss in energy.
CHAPTER 3  
Dynamic Simulation-based Kinodynamic Motion Planning

3.1 Introduction

The motion planning problem for a single robot subject to kinematics, dynamics, and collision constraints can be formulated as an optimal control problem [64, 43]. However, in practice it is not possible to solve this problem except for very simple cases. Finding an exact time-optimal trajectory for a point mass (with bounded velocity and acceleration) moving among polyhedral obstacles in \( \mathbb{R}^3 \) has been proven to be NP-hard [34]. Therefore, the use of sampling-based randomized techniques [51, 65, 66, 64], that try to provide a feasible plan, i.e., provide inputs as a function of time such that the robot’s state satisfies differential constraints (kinematic and dynamic constraints) and collision constraints has been proposed. The basic idea is to form a graph-based representation of the state space starting from some state that satisfies the constraints. In the basic algorithm, an input is randomly chosen from the set of inputs to act for some time \( \Delta t \), the equations of motion are integrated by calling the dynamic simulation module, and the state at time \( t + \Delta t \) is obtained. If the entire path is collision free the state is added as a node to the graph and the input is stored; otherwise, another input is chosen at random from the input set. The process is then repeated until the start and goal state belong to the same connected component of the graph. Any path on this graph from the start to the goal state gives a feasible motion plan. Note that the sampling is done over the space of possible inputs (or actions) to the system and the output of such sampling-based algorithms is a sequence of piecewise inputs.

The role of the dynamic simulation algorithms in sampling-based motion planning methods is to ensure that the state trajectories obtained satisfy the differential constraints. In this chapter, we propose the use of complementarity based model for dynamic state evolution, that was introduced in chapter 2. We point out the advantages of such methods in the context of sampling-based randomized motion planning techniques where collision avoidance is the key requirement. The complementarity conditions encode the physical constraint that two objects cannot interpenetrate, i.e., there will be a non-zero contact
force between two bodies if the distance between them is zero and zero contact force if the distance is non-zero. Using a complementarity based model for dynamic simulation, and simulating for a time $\Delta t^4$ using a given input we get non-zero contact force values when the objects just touch (we can use some safety distance so that the forces become non-zero when the objects are a safe distance apart) in that time interval. The end state at time $t + \Delta t$ is collision free. Moreover, we obtain the input that ensures that the whole path is collision free by adding the (suitably transformed) virtual contact forces to the input forces. This has the following advantages: (a) We always get a collision free path segment along with the corresponding input forces (provided the input forces do not violate actuator force constraints) and thus we do not waste any computation unlike in the conventional method when the path is not collision free. (b) Using our method, we obtain feasible inputs that are outside the set of primitives, thus essentially enhancing the input set available for planning, even when we start with a simple set of hand designed primitives. When the input set is a set of motion primitives, there may be no input in this primitive set that gives a feasible path for that time step.

The outline of the rest of this chapter is as follows: In Section 3.2 we provide a brief overview of the literature on robot motion planning with differential constraints. In Section 3.3 we provide a brief description of random sampling- based kinodynamic motion planning algorithms and in Section 3.4 we describe the changes we propose to the basic algorithm. In Section 3.5 we provide examples that depict the advantages of using our algorithm. Finally, we present our conclusions and outline future work.

### 3.2 Related Literature

The problem of finding an exact time-optimal trajectory, from a given start state to goal state, for a point mass with bounded velocity and acceleration moving among polyhedral obstacles is NP-hard [34]. Approximation algorithms have been proposed for systems with decoupled dynamics that are polynomial in the combinatorial complexity of the number of obstacles [33, 32]. However, these algorithms are exponential in the dimension of the configuration space. For practical purposes there are three basic approaches

---

4This is not the same as the time step used in the dynamic simulation module, i.e., numerical integration of the differential equations which may be much smaller than $\Delta t$. 
for kinodynamic motion planning problems: (a) Decoupled approach (b) Potential field based methods [59, 60, 93] (c) Sampling-based approaches [65, 51]. In the decoupled approach the problem is divided into a path planning and a trajectory planning problem. In the path planning stage a path is obtained for the robot that satisfies the geometric (collision) constraints and in the trajectory planning stage this path is converted to a trajectory while satisfying the dynamics constraints. Both the path planning problem and the trajectory planning problem have been studied extensively [24, 64]. The limitation of this method is that it may not be possible to convert the path into a trajectory satisfying all the dynamics constraints. In potential field based methods an artificial attractive field is generated in the state space with the goal as its minimum such that the robot reaches its goal by following the gradient of this field. This approach guarantees that the goal is reached in the absence of obstacles. But if obstacles are present, moving along the steepest gradient may lead to collision. To avoid this, repulsive potentials are designed around the obstacles; this can lead to the formation of local minima where the robot may get stuck. Thus there is no guarantee that the robot will reach its goal (except for a class of environments called star-shaped environments [93]), although the paths obtained would be feasible. In other words neither of the two approaches described above are complete, i.e., they may not find a feasible solution even though one exists.

Sampling-based approaches build a graphical representation of the free state space such that the start and goal states are nodes of the graph. There are two main approaches in the sampling-based randomized algorithms literature: (a) Rapidly Exploring Random Tree (RRT) algorithm [65] and (b) Algorithm based on the notion of expansive space [51]. The basic difference in the two methods is in the procedure used to bias the connectivity graph (tree) towards unexplored region of the state space (i.e., choice of the node to expand). The RRT algorithm generates a random sample and tries to connect the nearest node on the existing partial tree towards the random node. The algorithm in [51] maintains a weight at the nodes of the tree based on the number of samples that lie within a certain radius of the node. The algorithm then tries to expand the nodes having

---

5Lavalle distinguishes between RRTs and rapidly exploring dense trees (RDT) in [64] where the distinction is made to take into consideration deterministic sampling. For our purposes, this distinction is not important and we will discuss in the context of RRTs. All of the discussion also holds for deterministic sampling.
lesser weight. There are various variations of these two basic approaches [89], but all of these are concerned with heuristics on the choice of nodes to be expanded and the number of trees to be maintained in the exploration of the state space. For example bi-directional RRT [66] is a variation of RRT in which two trees are grown, one from the initial state and one from the goal state. The common feature of all these algorithms is that they choose a random input and use numerical integration of the differential constraints followed by collision checking when expanding a node. In our work we propose a modification to the step of expanding a node and not on the decision of which node to expand. We will present our discussion based on the basic RRT algorithm. The changes that we propose to the basic RRT are also applicable to all the other variations.

3.3 Sampling-Based Motion Planning with Differential Constraints

In this section, we pose the problem of motion planning with differential constraints in a formal setting and outline the basic RRT algorithm of sampling-based motion planning techniques. The presentation of the problem formulation follows [64]. Let $C$ be the configuration space, $X$ be the state space, and $U$ be the input (or action) space of the robot. We assume $X$ to be a smooth manifold and $U \subseteq \mathbb{R}^m \bigcup \{u_T\}$ to be a bounded set, where $u_T$ is the termination input. Without loss of generality we can write the state evolution equation of the robot as a system of first order ordinary differential equations

$$\dot{x} = f(x, u)$$

(3.1)

where $x = (q, \dot{q})$ is the state of the robot with $q$ the configuration of the robot, and $u \in U$ is the control input (or action). The evolution of the state as a function of time $t$, denoted by $x(t)$, is called the state trajectory of the robot and the time history of the applied control input, $u(t)$, is called the action trajectory. The state trajectory is called a feasible state trajectory if it satisfies Equation 3.1 and avoids collision with obstacles at each time instant. We denote the work space of the robot by $W \subseteq \mathbb{R}^n, n = 2, 3$, the set representing the robot by $A$, and the set representing the obstacles by $O$. The motion planning problem with differential constraints can be formally stated as follows [64]:
**Input:** The sets $W, A, O, X, U$ as defined above, feasible initial state $x_I$, feasible set of goal states $X_G$, and a possibly unbounded interval $T = [0, T_f]$.

**Output:** An action trajectory $u(t)$ for which the state trajectory $x(t)$ is feasible, $x(0) = x_I$, and there exists some $t > 0$ such that $u(t) = u_T$ and $x(t) \in X_G$.

---

**Algorithm 1 RRT algorithm**

```
BUILDRRT(x_I)
 Γ.init(x_I);
 for k=1 to K do
   x_rand ← RANDOMSTATE();
   EXTEND(Γ, x_rand);
 end for

EXTEND(Γ, x)
 x_near ← NEARESTNEIGHBOR(x, Γ);
 if NEWSTATE(x, x_near, x_new, u) then
   Γ.addvertex(x_new);
   Γ.addedge(x_near, x_new, u);
   if x_new = x then
     return Reached;
   else
     return Advanced;
 end if
 end if
 return Trapped;
```

The basic RRT algorithm is given in Algorithm 1 [65]. The tree representing the free configuration space is denoted by $Γ$. The algorithm starts from the initial node and at each iteration a new state that is biased towards a random state $x_{rand}$ is attempted to be added to the RRT (by calling the function EXTEND). The choice of the nearest vertex on the already existing tree $x_{near}$ in function NEARESTNEIGHBOR depends on the definition of a metric in the state space. The modifications that we propose are in the function NEWSTATE, which consists of the following steps:

1. An input $u \in U$ is applied to the robot at state $x(t) = x_{near}$ for some time $\Delta t$ and the equations of motion of the robot are numerically integrated to obtain the state trajectory from $x(t)$ to $x(t + \Delta t)$. The time $\Delta t$ is an input parameter to be chosen. The geometric constraints of collision avoidance are ignored at this step.
2. A collision detection algorithm is used to check the trajectory from $x(t)$ to $x(t+\Delta t)$ for collisions using a discrete sampling of the trajectory.

3. The input $u$ may be chosen at random, or if the set $U$ is finite then the steps 1 and 2 can be repeated for all possible inputs. In the former case, the state at time $t + \Delta t$ is added as a node to the tree and $u$ is stored if the trajectory is collision free. In the latter case, among all the collision free trajectories, the one where $x(t + \Delta t)$ is nearest to the random state is chosen and the corresponding input is stored.

Irrespective of the details, all the sampling-based randomized algorithms have the following limitations:

1. The choice of a proper metric is important for the algorithm because the choice of the neighbors, and hence the node to be expanded depend on this metric. In general, it is very difficult to specify a good choice for the metric and must be done in a problem dependent manner.

2. The geometric constraints (or collision constraints) are not considered during the dynamic simulation step. So, if there is a state violating the collision constraints even at the beginning of the simulation, it is not known until collision checking is performed after the whole numerical integration is completed.

3. A finite set of inputs is usually used in the dynamic simulation step. Thus at each step, the results obtained will be dependent on the input set chosen and the algorithm may not give a collision free trajectory for the step even if one exists.

4. At least one two point boundary value problem needs to be solved for connecting the tree to the goal state (or to a tree rooted at the goal state as is the case in bi-directional RRTs).

### 3.4 Proposed Planner and its advantages

In this chapter we propose solutions to remove limitations 2 and 3 listed above. We propose the integration of the geometric information into the dynamic simulation step by using a complementarity based model for the state evolution of the robot. The collision
constraints are modeled as complementarity constraints as described in Equation 2.3. We can modify these collision constraints by rewriting Equation 2.3 as

\[ 0 \leq \lambda_{in} \perp \psi_{in}(q, t) - \epsilon \geq 0 \]  

(3.2)

where \( \epsilon \geq 0 \) is a parameter specifying the safety distance to the obstacle that the robot must satisfy. Conceptually, we can think of this as a virtual obstacle that the robot can just touch. The discretized equations of motion to be satisfied at each time step are:

\[
\begin{align*}
M\nu^{\ell+1} &= M\nu^\ell + h(W_n\lambda_{n}^{\ell+1} + \lambda_{app}^{\ell} + \lambda_{vp}) \\
q^{\ell+1} &= q^\ell + hG\nu^{\ell+1} \\
0 &\leq h\lambda_{n}^{\ell+1} \perp \psi_{in}(q^{\ell+1}) - \epsilon \geq 0
\end{align*}
\]  

(3.3)

During the numerical integration of the equations of motion (Equation 3.3) we obtain the contact wrenches (\(w_i\lambda_{in}\) for the \(ith\) contact) that arise when the robot comes in contact with the virtual obstacle. Thus, at the end of the simulation we have a time history of the virtual contact wrenches over the interval \([t, t + \Delta t]\) along with a trajectory where the robot maintains a safe distance from the obstacle boundary. If we transform the virtual contact wrenches to the actuator space and add it to the input, then we obtain an input \((\tau_{app} + \sum_{i=1}^{n_c} J_i^T w_i \lambda_{in})\) that produces safe trajectories. If the input is within the set \(U\), then we have a feasible input. Usually the set \(U\) is defined by simple upper and lower bound constraints. Moreover, in the equations of motion we have the values of contact wrenches transformed to the actuation space (i.e., \(J_i^T w_i \lambda_{in}\)). Thus, after each step of the numerical integration process, we can check if the input required to satisfy the safety distance is feasible or not at very little computational cost and terminate the simulation if the input becomes infeasible. Moreover, we can also check at each step of the numerical integration if the state of the robot is changing; if not, we can terminate the simulation because a stationary state implies that the robot is stuck when using the chosen input. Thus our proposed modification (of considering the geometric constraints during the dynamic simulation step) to the sampling-based randomized algorithms has the following advantages:

1. We either get a collision free path and corresponding feasible inputs at the end
of the dynamic simulation step, or can terminate the simulation early if the input required to obtain a collision free trajectory is infeasible. Moreover, we also get the information of whether the robot gets stuck (i.e., its state does not change over two consecutive iterations) for the chosen input and can terminate the simulation after the robot reaches that state. At this state, any input in the conic hull of the negative contact wrenches will not change the state of the robot. Thus, we can easily check if a chosen input vector changes the state of the robot when it gets stuck. Alternatively, we can easily find a vector that releases the robot from the stuck state by checking if it lies in the conic hull formed by the contact wrenches.

2. When the input set of primitives is a finite set that is a subset of the feasible input set obtained by discretization (say), there may be no input in the discretized set that provides a collision free trajectory in the conventional algorithms. However, with our algorithm, we may be able to obtain feasible inputs that lie outside the set of the primitives. Thus, we may not be limited to the set of inputs that we start out with.

We note that in Equation 3.3, if we approximate the distance function $\psi_n(q^{\ell+1})$ with a Taylor’s series expansion, we will have a mixed linear complementarity system to be solved at each step. The variables to be solved for at each step are $\nu^{\ell+1}$ and $\lambda_n^{\ell+1}$. Since the mass matrix is positive definite, the matrix defining the LCP is positive definite. This implies that there exists a unique solution to the LCP at each time step that can be obtained in polynomial time (although for general LCPs the time taken for finding a solution may be exponential in the number of variables).

### 3.4.1 Planning using a single potential field

In our algorithm, when we have an input that takes the robot directly towards the goal, the robot may reach the goal even if the line joining the start and goal intersects an obstacle. In other words, there are certain sets of obstacle arrangements and initial and final configurations where we can obtain a feasible trajectory to the goal state with a single attractive potential function with its minimum at the goal. This happens if the obstacles are arranged such that the line joining the robot’s current state to the goal is not normal to any of the obstacles it is intersecting with. For example, in Figure 3.1(b),
if the robot moves along the line joining \( S \) and \( G \) it will reach the goal. However, in Figure 3.1(a), the robot will not reach the goal. Note that if the projection of the motion of the robot in a time-step on the line joining the start and goal is always positive then the robot always moves toward the goal. When the line joining the robot’s state and goal is not normal to the surface of the obstacles it intersects, the above condition is true. Hence the robot always reaches its goal. The input required to reach the goal is obtained from the applied input plus the transformed contact wrenches.

### 3.4.2 Effect of choice of \( \Delta t \) and input

As is evident from the discussion on sampling-based algorithms in Section 3.3 the simulation time \( \Delta t \) is a parameter to be chosen. Moreover, when the input set is given as a compact set, the number of possible inputs is infinite. Thus, at each simulation step it is not possible to check all possible inputs, and a subset of the available inputs is chosen at each step. This subset may be randomly drawn or formed by discretizing the available input set. To illustrate the effect of this let us consider a simple example as shown in Figure 3.1(a). We consider a point particle that has to move from the start position \( S \) to goal \( G \). The particle can be independently actuated along the \( x \) and \( y \) axis (horizontal and vertical directions in the figure). As can be seen from Figure 3.1(a), if we use the Newton-Euler equations only for the dynamics, the planning algorithms will find a solution if and only if the value of \( \Delta t \) is chosen small enough and the input is small enough such that an end state lies in the dashed triangular area. In contrast, the solution we will get from our algorithm is shown by the bold line that is independent of the choice of \( \Delta t \). In our case the choice of \( \Delta t \) determines how fast the solution is found, but not whether the solution can be found or not.

We also note that in our algorithm there is more flexibility to the choice of inputs. In the example in Figure 3.1(a), we will get a solution if our input set consists of five primitive inputs, one input each along the positive and negative \( x \)–axis and \( y \)–axis respectively and an input directly towards the goal. This is what we use in our implementations; two inputs along each of the independently actuated coordinates (one toward the positive and one toward the negative axis) and an input directly towards the goal from the current state. Using the inputs along the coordinates only is not enough, as can be seen
in Figure 3.1(b). When the normal to the obstacle at the point of impact directly opposes the input, the robot gets stuck.

3.4.3 Implication for narrow passage problems

The choice of the simulation time $\Delta t$ and the input set becomes critical in cluttered environments because the probability of collision is very high. In such environments, it is difficult to find a feasible input and it is well known that sampling-based algorithms for kinodynamic planning perform poorly in such environments. Thus, in cluttered environments, i.e., environments characterized by the presence of narrow passages, our modification would improve the performance of a sampling-based algorithm since it always finds a collision free trajectory and the corresponding input. More specifically, if we arrive near a narrow passage during the construction of the graph, then we are likely to find inputs that will take us through the passage. Our algorithm does not say anything about how to get near the narrow passage, which is very difficult to know.

3.4.4 Completeness Properties of our Algorithm

The change we are proposing to the existing sampling-based randomized planning techniques enhances the reachability set at each point. In other words, for any variation of the randomized sampling-based techniques, the reachability set from any given state, using our complementarity based dynamic simulation algorithm is a superset of the reachability set using the conventional dynamic simulation. Thus all the probabilistic completeness and resolution completeness results proved for any sampling-based algorithm still holds with our variation.
3.4.5 Use of our algorithm for purely geometric problems

Although our algorithm is designed for problems where dynamics constraints need to be considered, it may be useful even for motion planning problems with purely geometric constraints. This is because in problems characterized by the existence of narrow passages, it is difficult to get a sample in the narrow passages. However using a fake dynamics for the robot and the complementarity constraints, we are guaranteed to get samples in the free space, since all the configurations of the robot satisfying the complementarity constraints lie in the free space. Thus it may be easier to obtain samples in the narrow passage once we are near the narrow passage.

3.5 Simulation Results

In this section we present some simulation examples illustrating our method. The first example shows a scenario where a point robot reaches the goal under the action of a single attractive potential field in the presence of obstacles. The obstacles hinder the motion along the direction of steepest descent of the potential function. The environment in this case is not a star shaped environment and hence it is not possible to design navigation functions that ensure that the goal can be reached. The second example is that of a $2R$ planar manipulator moving among three obstacles. The problem is hand-designed so as to contain two narrow passages in the free configuration space connected by a relatively large free space region. The start and the goal configurations are placed at the two ends of the narrow passage. In this case we use our modified version of the RRT algorithm to obtain the solution trajectory.

3.5.1 Example 1: Planar point robot

This example illustrates planning with a single potential field at the goal in the presence of obstacles. We consider a $2D$ point robot moving in a rectangular environment with saw tooth shaped obstacles (see Figure 3.2). The saw tooth shaped obstacles form a narrow passage in the environment and the start and goal configuration are located on opposite sides of the obstacles. Let $q = (x, y)$ be the configuration and $\nu = \dot{q}$ be the velocity of the robot. The discrete time dynamic model for state evolution of the system
Figure 3.2: Path of a point robot moving through a narrow passage formed by the wedge shaped obstacles

is

$$0 = -M\nu^{\ell+1} + M\nu^{\ell} + W_n p_n + p_{\text{app}}$$

$$0 \leq p_n \perp \psi_n(q^{\ell}) + W_n^T \nu^{\ell+1} \geq \epsilon$$

(3.4)

where $M$ is a $2 \times 2$ diagonal matrix with the mass $m$ of the robot as the diagonal entries, $W_n$ is a $2 \times 1$ vector giving the normal at the contact point, $\psi_n(q^{\ell})$ is the distance of the robot to the obstacle at time $\ell$, and $\epsilon$ is the safety distance from the obstacle. Both the distance and the normal at time $\ell$ are obtained from a collision detection algorithm. The applied impulse is given by:

$$p_{\text{app}} = -h(K_p(q^{\ell} - q_g) + K_d q^{\ell})$$

(3.5)

where $K_p$ and $K_d$ are diagonal matrices with positive entries, $h$ is the time step, and $(q, \nu) = (q_g, 0)$ is the goal state. The unknowns in Equation 3.4 are $\nu^{\ell+1}$ and $p_n$. Thus, we have a system of 3 equations and 3 unknowns.
Figure 3.3: Enlarged portion of the path of the point robot that shows the safety distance from the obstacle is maintained

For generating the results, we have used the following data: $m = 1, \epsilon = 0.01, \alpha = 0.01, h = 0.01$, each diagonal entry of $K_p$ and $K_d$ is 1, initial state is $(0.4, 0.9, 0, 0)$ and goal state is $(5, 0.5, 0, 0)$. We simulated the system in Equation 3.4 using $p_{app}$ given by Equation 3.5. The contact impulse obtained from the simulation $W_n p_n$ is then added to the applied impulse to obtain the collision-free input impulse. The path taken by the point robot to reach the goal under the action of the collision free input impulse is shown in Figures 3.2 and 3.3. Note the approach of the robot to the goal state (when the $x$-coordinate of the robot is greater than 4.5) that clearly shows the effect of dynamics. The robot does not go straight to the goal because of its non-zero momentum in the direction orthogonal to the goal direction. Ignoring the dynamics would give a solution path straight to the goal (when there is no obstacle along the line joining the robot position to goal). Figure 3.4 gives the velocities of the robot as it moves towards the goal. When the robot reaches the safety distance from the wall the $x$-component of its velocity drops to 0 and then becomes less than 0 as the $y$-momentum carries it along the obstacle. Near the end, when there are no obstacles, the velocities decay to 0 because of the damping term in the
applied force. The top two subplots of Figures 3.5 and 3.6 show the $x$ and $y$ components of the applied force and contact impulses obtained from the simulation step. The last subplot in the figures show the $x$ and $y$ components of the collision-free input. The basic RRT method could not find a solution to this problem with $\Delta t = 0.1$ seconds and the input set consisting of a random vector, a force towards the goal given by Equation 3.5, and

$$u = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$$

(3.6)

3.5.2 Example 2: 2R Manipulator

The second example is that of a $2R$ manipulator (a planar manipulator with 2 revolute joints, see Figure 3.7) that has to move from a start to goal configuration in a gravity free environment with three circular obstacles (see Figure 3.13). The free space of the manipulator relevant to the problem along with the start and goal configurations is shown in Figure 3.8. Let the joint angles $q = (q_1, q_2)$ be the configuration of the robot and $\nu = \dot{q}$ be the joint angle rates. The discrete time dynamic model for state evolution of the system

Figure 3.4: Velocity of the point robot
Figure 3.5: X-component of the forces/impulses acting on the point robot

is

\[ 0 = -M \nu^{\ell+1} + M \nu^{\ell} - p_{vp} + W_n p_n + p_{app} \]
\[ 0 \leq p_n \perp \psi_n(q^\ell) + W_n^T \nu^{\ell+1} \geq \epsilon \]  \hspace{1cm} (3.7)

where the mass matrix and Coriolis force term are given by

\[ M = \begin{bmatrix} \alpha + 2\beta \cos(q_2) & \delta + \beta \cos(q_2) \\ \delta + \beta \cos(q_2) & \delta \end{bmatrix} \]
\[ p_{vp} = \begin{bmatrix} -\beta \sin(q_2)q_2^\ell & -\beta \sin(q_2)(q_1^\ell + q_2^\ell) \\ \beta \sin(q_2)q_1^\ell & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1^\ell \\ \dot{q}_2^\ell \end{bmatrix} \]
\[ \alpha = I_{z1} + I_{z2} + m_1r_1^2 + m_2(r_1^2 + r_2^2) \]
\[ \beta = m_2l_1r_2 \]
\[ \delta = I_{z2} + m_2r_2^2 \]  \hspace{1cm} (3.8)
where $m_i, l_i, r_i, I_{z_i}, i = 1, 2$, are the mass, length, distance of center of gravity (cg) from joint frame, and moment of inertia about an axis passing through the cg and perpendicular to the plane for the two links respectively. The vector of contact impulse magnitudes in Equation 3.7 is of size $6 \times 1$ and that of $W_n$ is $2 \times 6$. Each column of $W_n$ is formed by the product of $J_k^T \hat{w}_k$ with $\hat{w}_k$ being the $2 \times 1$ unit normal vector at the contact point and $J_k$ being the $2 \times 2$ Jacobian up to the contact point. Depending upon the link on which the contact occurs the Jacobian is one of the following two forms:

$$
J_{\text{link1}} = \begin{bmatrix} x_{c1} & 0 \\ y_{c1} & 0 \end{bmatrix} \quad J_{\text{link2}} = \begin{bmatrix} x_{c2} & x_{c2} - l_1 \cos(q_1) \\ y_{c2} & y_{c2} - l_1 \sin(q_1) \end{bmatrix}
$$

(3.9)

The input set consists of a force directed towards the goal given by Equation 3.5 and those given by Equation 3.6. From the initial state, we apply the 4 inputs given by Equation 3.6 for a chosen time $\Delta t$ and the input directed towards the goal until the robot gets stuck (i.e., the joint angle rates become zero). We then iterate this procedure starting with the states obtained at the previous level and repeat till the robot reaches the goal. If
the robot is at a stuck state and a particular input does not move it from that state, we reject the input after two time-steps of dynamic simulation. Note that this is a very basic version of RDT where we are growing the tree by expanding all the nodes. We obtain a solution to our problem in three iterations. The number of nodes in the tree is 27. In our simulations we have used the following data: $m_1 = m_2 = 1$, $l_1 = l_2 = 1$, $r_1 = r_2 = 0.5$, $\Delta t = 2s$, $h = 0.01s$, initial state $[0.22\pi, 1.94\pi, 0, 0]$, and goal state $[0.5\pi, 0.1\pi, 0, 0]$. The proportional and derivative gains, $K_p$ and $K_d$, in Equation 3.5 are assumed to be $2 \times 2$ diagonal matrices. Each diagonal entry of $K_p$ is 3 and $K_d$ is 4. The basic RRT could not find a solution to this problem with $\Delta t = 0.1$ seconds.

Figure 3.8 shows the collision-free path of the robot in configuration space that we obtained. Figure 3.9 shows the variation of joint angle rates in our solution as the manipulator moves from start to goal state. Figures 3.10(a) and 3.11(a) show the input torques at joint 1 and joint 2 respectively, during the search stage. The contact impulses projected to the actuator space are shown in Figures 3.10(b) and 3.11(b). The final collision-free input impulses at each joint that takes the manipulator from start to goal state are shown in Figures 3.10(c) and 3.11(c). Figures 3.12, 3.13, and 3.14 show some snapshots of the
path of the manipulator in configuration space and work space. Note that Figures 3.12 and 3.8 are the same figure; the former is drawn at a lesser resolution so that the points in the configuration space can be clearly identified visually.

Figure 3.8: Configuration space of the 2R manipulator. The white region shows the free space. The path of the robot in the configuration space from start to goal is shown by the dotted line. The points S and G are the start and goal configurations respectively.

3.6 Conclusion

In this chapter, we have shown the use of complementarity based dynamic simulation algorithm in kinodynamic motion planning problems and pointed out its advantages in solving certain narrow passage problems. We note that we have restricted our attention to problems without contact (or for problems involving collision avoidance only). In principle our method also applies to applications where the robot may be in contact with
Figure 3.9: Variation of the joint angle rates of the 2R robot for motion from start to goal configuration in Figure 3.8.

other objects. However, in problems involving contacts, the complementarity based simulations suffer from some drawbacks as depicted in Chapter 2. In the next two chapters we present solutions that overcome these limitations.
Figure 3.10: (a) The applied torque at joint 1 used for dynamic simulation (b) The contact impulses transformed to joint 1 obtained from dynamic simulation using the input in subplot (a). (c) The final collision free input torque impulse for joint 1.
Figure 3.11: (a) The applied torque at joint 2 used for dynamic simulation (b) The contact impulses transformed to joint 2 obtained from dynamic simulation using the input in subplot (a). (c) The final collision free input torque impulse for joint 2.
Figure 3.12: Configuration space of the 2R manipulator. The white region shows the free space. The black squares show configurations along the collision-free trajectory. The positions of the manipulator in the workspace corresponding to the configurations are shown in Figures 3.13 and 3.14. The points S and G are the start and goal configurations respectively.
Figure 3.13: The first four configurations in the path of the 2R robot for motion from start to goal configuration in Figure 3.12.

Figure 3.14: The last four configurations in the path of the 2R robot for motion from start to goal configuration in Figure 3.12.
CHAPTER 4
Proximity Queries between Convex Objects

4.1 Introduction

In this chapter we study the problem of computing the closest points on two convex objects, when each object is described as an intersection of implicit surfaces. Distance computation algorithms are usually used in the narrow phase of a collision detection algorithm in applications where knowledge of the closest points is required rather than just a yes/no answer for collision. Such applications are characterized by existence of intermittent contact, i.e., phases of contact and no contact between the objects, with a concomitant need to predict potential contact points. Some example applications are dynamic simulation [3, 77, 106], dexterous manipulation [79, 10], computer animation [14], and haptics [73, 25]. Other applications where collision avoidance is the primary goal may also make use of the knowledge of the closest distance information. Examples of such applications include robot path planning [90] and spacecraft safe volume computations [26].

It is well known that the evolution of contact points for continuous contact states depends on the relative curvature of the two contacting bodies [78, 62]. Polygonalization may lead to poor approximation of the object curvature and consequently affect the accuracy of the dynamic simulation. For applications in motion planning and engineering analysis of objects in intermittent contact [104], such discretization may not be desirable (see rolling disc example in Chapter 2). More generally, polygonalized representations of smooth objects can lead to intermittent loss of contact and bouncing in the simulations. The effect of shape and surface curvature on the dynamic motion of contacting objects is dramatically illustrated by a spinning boiled egg and toys such as the rattleback (or Celtic wobblestone) [97, 40]. Similarly, exact representation of shape is important when modeling robot fingers in contact with smooth objects during multi-finger dexterous manipulation (Figure 4.1). Determining the first contact point correctly when fingers reposition during finger gaighting is useful since the manipulation operations are sensitive to the contact point and the normal and curvature at that point [10, 85].
The general problem of distance computation between two objects $X$ and $Y$ can be written as

$$\text{Minimize} \quad \|x_g - y_g\|_2$$

subject to: $x_g \in X, \quad y_g \in Y$ (4.1)

where the two objects $X$ and $Y$ are represented as compact (closed and bounded) sets in $\mathbb{R}^2$ or $\mathbb{R}^3$ and the points $x_g$ and $y_g$ are points in the two objects. This problem has been extensively studied [53, 71], mainly for polyhedral object representations [31, 45, 69, 76]. In this thesis, we focus on representing the sets $X$ and $Y$ as intersections of implicit surfaces, including planes, quadrics, superquadrics, and hyperquadrics. We assume that an implicit surface model of each object is given to us. Our choice of object representation is motivated by the goal of simulating systems with smooth objects, where polygonal discretizations may not be desirable. (While parametric representations can also represent smooth objects, they provide a surface description of objects with nonlinear equations and thus the resulting problem is not a convex optimization problem even for convex objects.) The literature on distance computation between general implicit surfaces is relatively sparse because, with a few notable exceptions [44, 109], methods for polyhedral representations do not easily generalize to implicit surfaces. Having a smooth representation of objects and an algorithm to perform distance computation between such representations will enable the study of the effects of shape and polygonalization on dynamic simulation of systems with intermittent contact.

Contributions: This chapter focuses on the problem of computing the minimum distance between two convex objects, where each object is described as an intersection of implicit surfaces. This class of convex objects includes for example, convex polyhedra, quadrics, superquadrics, and hyperquadrics. While the distance computation problem for convex objects represented by convex inequalities has been known to be a convex optimization problem [7, 9], to the best of our knowledge, interior point algorithms have not been previously applied to this problem. Interior point methods are well suited for this class of optimization problems since they are guaranteed to converge to the global optimum for convex problems. Further, they exhibit polynomial convergence for special classes of functions called self-concordant functions. We apply a recently developed in-
A dexterous manipulation task that requires closest distance computations to predict the contact points of fingers with an object. The fingers and object are represented as superquadrics.

The chapter is organized as follows. After a discussion of related work in Sec-

terior point algorithm [11, 116] to compute the distance between convex implicit surface objects and demonstrate that it is particularly effective for this class of problems. For polyhedral and quadric surfaces, the algorithm takes $O(n^{1.5})$ time, where $n$ is the number of constraints. We also illustrate the approach on surfaces such as superquadrics and hyperquadrics. To the best of our knowledge, this is the first approach with this demonstrated capability (without discretization). Another important advantage of this method is that it provides a uniform framework for proximity queries between objects described as intersections of convex polyhedra, quadrics, or any arbitrary convex implicit surface. Further, these proximity queries can be used in the narrow phase of hierarchical collision detection for implicit surfaces. We present implementation results for example implicit surface objects that show that the algorithm exhibits linear time performance in practice, and demonstrate that distance computation rates of about 1 kHz can be achieved. We also extend the approach to proximity queries between deforming convex objects. Finally, we show that continuous collision detection for linearly translating objects can be performed by solving two related convex optimization problems. For polyhedra and quadrics, we establish that the time complexity of this continuous collision detection problem is $O(n^{1.5})$. 
tion 4.2, we review the mathematical background for our work in Section 4.3. We present the formulation of the closest distance problem in Section 4.4 and describe how it can be solved using interior point algorithms in Section 4.5. Section 4.6 provides theoretical and practical results on the complexity of the closest point algorithm. Section 4.7 extends the approach to continuous proximity queries for linearly translating objects. We present our implementation results in Section 4.8 and conclude with a discussion of future work in Section 4.9. The work in this chapter has appeared in [21, 22].

4.2 Related Work

Proximity queries for polyhedra: Proximity queries and collision detection algorithms have an extensive literature in computational geometry [31], robotics [45, 69], and computer graphics [109]. We provide a sampling of the related work in these areas; see [71, 53] for an overview of collision detection and proximity queries. When collision detection algorithms estimate the distance between two objects, they typically use a geometric approach. Popular algorithms for convex polyhedra include GJK [45], Lin-Canny [69], and V-Clip [76]. GJK [45] is an iterative algorithm for distance computation between two convex polyhedra. It uses a support function description of the polyhedra and takes time linear in the number of vertices. Lin-Canny [69] efficiently computes the distance between two convex polyhedra and tracks the closest points using adjacency of features. Its running time is linear in the number of features (faces, edges, and vertices). Both algorithms can track the closest points in (almost) constant time when there is temporal coherence [13]. Bobrow [7] proposed an optimization based approach for computing the distance between two convex polyhedra. He formulated the problem as a quadratic programming problem and used a gradient projection algorithm to solve the problem. However this approach can suffer from convergence issues [122].

Proximity queries for quadrics and NURBS: Distance estimation between non-polyhedral shapes has focused primarily on quadrics and NURBS surfaces. Among the algorithms for polyhedra, only GJK has been extended directly for smooth convex objects [44]; van den Bergen [109] discusses in detail a GJK implementation for convex quadric objects. However this GJK algorithm does not guarantee convergence in a finite number of steps. Further, computing the support mapping is difficult for superquadrics
with fractional (non-integer) exponents due to the difficulty of solving polynomial equations with fractional exponents. Turnbull and Cameron [28] extended GJK to convex NURBS surfaces. They describe a procedure to calculate the support mapping for the NURBS surfaces which reduces to solving two nonlinear polynomial equations in two parameters. They present results for 2D and describe the theory for 3D. Baraff [3] described collision detection algorithms for implicit and parametric curved convex surfaces. He uses the collinearity property of the surface normals of the closest points to numerically compute closest points at the initial configuration. He exploits geometric coherence to compute closest points at subsequent configurations. Lin and Manocha [70] consider curved models described as NURBS surfaces and piecewise algebraic surfaces. Using the collinearity property of the surface normals, they describe the closest points using a set of polynomial equations. However, the number of roots can be prohibitively large as it depends on the degree of the polynomials describing the surfaces; the roots must be examined to identify the closest points. Note also that it is not possible to obtain bounds on the number of roots for systems of equations with fractional indices (as would arise with superquadrics). Schomer et al. [98] describe a collision detection algorithm for curved objects bounded by quadric surface patches by finding roots of univariate polynomials of degrees 4 and 8. Johnson and Cohen [54] give a lower-upper bound tree framework for distance computation between any two object representations for which the following set of operations is available: bounding volume generation, lower and upper bound on distance, bounding volume refinement, and determination of computation termination. They have demonstrated their method on polyhedra as well as NURBS surfaces. Patoglu and Gillespie [83] perform real-time tracking of the closest points between two objects modeled with parametric surfaces by formulating it as a control problem and exploiting spatial and temporal coherence.

The literature on distance computation between general implicit surfaces is relatively sparse because, with the exception of GJK, methods for polyhedral representations do not easily generalize to implicit surfaces. In fact, no closed form solution exists even for the distance between a point and an ellipsoid. Most closely related is recent work on computing the distance between two ellipsoids and other conic sections [68, 26, 100]. Sohn et. al. [100] exploit the fact that the closest points on two surfaces are where their
common normals intersect the surfaces. They apply their line geometry approach to ellipsoids, for which the minimum distance computation is reduced to finding the common roots of two polynomial equations of degree 8 and 16. Coppola and Woodburn [26] formulate the problem as an optimization problem. They iteratively solve the problem of closest distance from a point to an ellipsoid to arrive at the optimal solution. Rimon and Boyd [92] use convex optimization techniques to find the minimum volume enclosing ellipsoids to model objects, and then compute a conservative distance estimate between ellipsoids by treating it as an eigenvalue problem. Choi et al. [23] present a continuous collision detection algorithm for two elliptical disks moving in the plane. Collisions are identified by checking for the real roots of the univariate discriminant of the characteristic equation of the two moving ellipses. Although superquadrics are a generalization of quadrics, the problem in generalizing the methods in [3, 45, 26, 100] to superquadrics is that they all lead to polynomial equations with fractional exponents, which are very difficult to solve. In general, we do not know the total number of roots, and even when it is possible to simplify the polynomials, they may have large integer exponents.

4.3 Mathematical Preliminaries

We now review the mathematical terminology that will be used in the rest of the chapter.

Convex Set: A set $U \subseteq \mathbb{R}^n$ is called a convex set if for any two points $u_1, u_2 \in U$ and any $\lambda$ with $0 \leq \lambda \leq 1$, we have

$$\lambda u_1 + (1 - \lambda)u_2 \in U.$$ 

Convex Function: A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if the domain of $f$ is a convex set and for all $u_1, u_2 \in \text{dom } f$ and any $\lambda$ with $0 \leq \lambda \leq 1$, we have

$$f(\lambda u_1 + (1-\lambda)u_2) \leq \lambda f(u_1) + (1-\lambda)f(u_2).$$
**Convex Programming Problem:** Consider the general nonlinear programming problem given by:

\[
\begin{align*}
\text{Minimize} & \quad f_0(x) \\
\text{subject to:} & \quad x \in U
\end{align*}
\]

(4.2)

This nonlinear programming problem is called a convex programming problem if the objective function \(f_0\) is a convex function and the feasible set \(U\) is a convex set [5]. Usually the set \(U\) is defined by a set of inequality and/or equality constraints. If the inequality constraints defining \(U\) are convex functions and the equality constraints are linear, then \(U\) is a convex set [9].

**Superquadric:** A superquadric [49] is defined by the equation

\[
f(x) = \left| \frac{x_1}{a_1} \right|^{n_1} + \left| \frac{x_2}{a_2} \right|^{n_2} + \left| \frac{x_3}{a_3} \right|^{n_3} - 1 = 0
\]

\[
n_i = l_i/m_i, \quad l_i, m_i \in \mathbb{Z}^+, \quad i \in \{1, 2, 3\}
\]

(4.3)

\[
f(x) \ \text{convex if} \quad 1 \leq n_i < \infty
\]

\[
f(x) \ \text{nonconvex if} \quad 0 < n_i < 1
\]

Although the definition here differs slightly from that in [4], the two definitions are equivalent [49]. Convex superquadrics are a broad class of shapes that include cuboids, rounded cuboids, ellipsoids, spheres, and (rounded) octahedra. The planes \(\left| \frac{x_i}{a_i} \right| \leq 1, i = 1, 2, 3\) define a bounding cube for the superquadric and the indices control the roundedness of the shape. Different shapes can be obtained by varying \(n_i\). The shape is a rhomboid when \(n_i = 1\), and the shape becomes a cube as \(n_i\) tends to infinity.
Hyperquadric: A hyperquadric [49] is defined by the equation

\[ f(x) = \sum_{i=1}^{N} |H_i(x)|^{n_i} - 1 = 0 \]

where \( N \geq 3 \) and

\[ H_i(x) = (a_i x_1 + b_i x_2 + c_i x_3 + d_i) \]

\[ n_i = l_i/m_i, \quad l_i, m_i \in \mathbb{Z}^+ \]

(4.4)

Hyperquadrics are a more general class of shapes than superquadrics. In particular, they include asymmetric shapes. In this case also, the intersection of the planes \( H_i(x) \leq 1 \) form a bounding polytope for the hyperquadric and the indices control the roundedness of the shape.

Self-concordant functions: A convex function \( f : \mathbb{R} \rightarrow \mathbb{R} \) is self-concordant if

\[ |f'''(x)| \leq 2f''(x)^{3/2} \]

for all \( x \in \text{dom} f \).

A convex function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is self-concordant if it is self-concordant along every line in its domain (see [9] for details).

4.4 Problem Formulation

In this section we present the formulation of the minimum distance computation problem. We first outline a geometric approach to the minimum distance problem that has been popular in prior work on non-polyhedral objects [3, 70, 100] and connect it to an optimization formulation. Let \( f_X \) be an implicit function representing object \( X \) and \( f_Y \) be an implicit function representing object \( Y \), and let \( x_g, y_g \) be the global coordinates of points in \( X \) and \( Y \) respectively. To compute the closest distance between \( X \) and \( Y \), the approach uses the geometric condition that the normals on the two surfaces at the closest points are aligned with each other. Using this and the condition that the closest points should lie on the surfaces of the two objects, we can obtain the closest points by solving
the following system of nonlinear equations

\[ \begin{align*}
  x_g - y_g &= -\lambda_X \nabla f_X(x_g) \\
  x_g - y_g &= \lambda_Y \nabla f_Y(y_g) \\
  f_X(x_g) &= 0 \\
  f_Y(y_g) &= 0
\end{align*} \] (4.5)

where \( \lambda_X \) and \( \lambda_Y \) are scalars. The conditions given by Equation 4.5 are precisely the Karush-Kuhn-Tucker (KKT) conditions for solving the following optimization problem.

\[
\begin{align*}
\text{Minimize} & \quad \|x_g - y_g\|_2 \\
\text{subject to:} & \quad f_X(x_g) = 0 \\
& \quad f_Y(y_g) = 0
\end{align*}
\]

However note that when \( f_X \) and \( f_Y \) are nonlinear, the above problem is nonconvex even when the objects are convex and the solution can therefore get stuck in a local minimum.

We now formulate the problem of minimum distance computation between two convex objects as a convex optimization problem. Each object is assumed to be described as an intersection of a finite number of implicit primitives (i.e., a finite number of algebraic inequalities). In general, each of the intersecting surfaces may be specified in its own reference frame. The distance computation problem of Equation 4.1 can then be written as

\[
\begin{align*}
\text{Minimize} & \quad \|x_g - y_g\|_2^2 \\
\text{subject to:} & \quad x_g = R_{xi}x_{li} + p_{xi} \quad i = 1, \ldots, m \\
& \quad y_g = R_{yj}y_{lj} + p_{yj} \quad j = m + 1, \ldots, n \\
& \quad f_i(x_{li}) \leq 0 \quad i = 1, \ldots, m \\
& \quad f_j(y_{lj}) \leq 0 \quad j = m + 1, \ldots, n
\end{align*}
\] (4.7)

where \( x_g, y_g \in \mathbb{R}^3 \) are the global coordinates of points in the two objects \( X \) and \( Y \) respectively; \( R_{ki}, p_{ki}, k = x, y \), are the rotation matrix and position of the reference frame of each of the intersecting surfaces with respect to the global frame, \( x_{li}, y_{lj} \in \mathbb{R}^3 \).
are the coordinates of the points in the local reference frames of the surfaces, and \( f_k \) \((k = i, j)\) are the functions representing the implicit surfaces in the local reference frames. The above system has 3\(n\) linear equality constraints and \(n\) inequality constraints. Note that the linear constraints in Equation 4.7 can be any affine transformation (not necessarily a rigid body transformation). In particular, we can handle global deformations like nonuniform scaling by post multiplying the rotation matrix with a scaling matrix.

From the linear constraints in Equation 4.7 we can easily evaluate \(x_{li}\) and \(y_{lj}\) and so the inequality constraints can be expressed in terms of \(x_g\) and \(y_g\). Therefore, without loss of generality, we can assume that the implicit surface describing the objects is described in a global reference frame. The distance computation problem of Equation 4.1 is then given by

\[
\text{Minimize} \quad \|x_g - y_g\|_2^2 \\
\text{subject to:} \quad f_i(x_g) \leq 0 \quad i = 1, \ldots, m \\
\quad f_j(y_g) \leq 0 \quad j = m + 1, \ldots, n 
\]

where \(f_k \) \((k = i, j)\) are the functions representing the implicit surfaces in the global reference frame. The above system has \(n\) inequality constraints. The objective function in Equation 4.8 is convex, and if the inequalities represent a convex set (i.e., the objects are convex), the minimum distance computation problem is a convex programming problem. For general convex surfaces, the distance computation problem is a nonlinear program (NLP). For objects described as convex quadric surfaces, the problem reduces to a quadratically constrained quadratic program (QCQP), and if the objects are convex polyhedra (intersections of planes), the closest distance problem becomes a quadratic programming (QP) problem.

The solution to the minimum distance problem of Equation 4.8 gives two closest points that lie on the surfaces of the two objects (i.e., boundaries of the two sets). We use an interior point algorithm [11] for solving this problem. See Figure 4.2 for an example solution generated using an interior point algorithm. Interior point methods [118] are a class of optimization algorithms for nonlinear programming problems. In contrast to algorithms for finding the closest points that generate iterates that lie on the surface of the objects (gradient projection [7], for example), feasible interior point methods generate
iterates that are guaranteed to lie inside the objects and converge towards the closest points on the boundaries of the objects. This is the main conceptual difference between interior point methods and other methods. Sequential quadratic programming (SQP) is another method for solving general nonlinear programming problems [46]. In contrast to SQP, interior point methods have polynomial time convergence guarantees for certain convex problems, as we describe in Section 4.6. Moreover, an informal comparison of SQP implementations with interior point algorithm implementations on the NEOS server [29] shows the interior point methods to be slightly faster. Therefore, we choose to solve the optimization problem with an interior point algorithm.

4.5 Interior Point Algorithm

In this section, we present the **primal-dual** interior point algorithm for solving the optimization problem described in Equation 4.8. The Karush-Kuhn-Tucker (KKT) conditions give necessary and sufficient conditions for solving the minimum distance problem in Equation 4.8, since it is a convex optimization problem and satisfies Slater constraint qualification. For ease of presentation, we rewrite Equation 4.8 in a general nonlinear
program format as

Minimize \( f_0(x) \)

subject to: \( f(x) + s = 0 \)
\[ s \geq 0 \] (4.9)

where \( f_0(x) = \|x_g - y_g\|_2^2, x = [x_g^T, y_g^T]^T \) is a 6 \times 1 column vector, \( s \) is an \( n \times 1 \) column vector of slack variables, and \( f : \mathbb{R}^6 \to \mathbb{R}^n \) is the vector of inequality constraints. The Lagrangian for the above constrained optimization problem can be written as

\[ f_0(x) + \lambda^T (f(x) + s) \] (4.10)

where \( \lambda \) is an \( n \times 1 \) vector of Lagrange multipliers. The KKT conditions for Equation 4.9 are the system of nonlinear equations below:

\[ \nabla f_0(x) + (\nabla f(x))^T \lambda = 0 \]
\[ f(x) + s = 0 \]
\[ LSe = 0 \] (4.11)

Here \( L \) is an \( n \times n \) diagonal matrix of the \( \lambda \) variables, \( S \) is an \( n \times n \) diagonal matrix of the slack variables \( s \), and \( e \) is an \( n \)-vector of ones. The above is a system of \( 2n + 6 \) nonlinear equations in the \( 2n + 6 \) variables \( x, \lambda, s \). Equation 4.11 can be solved by Newton’s method for solving systems of nonlinear equations. It converges to the correct solution if the initial guess is near enough [80]. However, in general, it is very difficult to supply a good initial guess and there is then no guarantee that Newton’s method will converge. The main difficulty in using Newton’s method is that we have to ensure \( s \geq 0 \), which may lead to very small step lengths that result in convergence problems.

Interior point methods are a general class of algorithms for solving nonlinear programming problems. In essence, these methods approximately solve a sequence of systems of nonlinear equations that are formed by perturbing the complementarity equations \((LSe = 0)\) in the KKT conditions. Following [9], we present the interior point method by
reformulating Equation 4.9 as a *barrier* problem.

\[
\text{Minimize } f_0(x) - \mu \sum_{i=1}^{n} \ln(s_i) \quad (4.12)
\]

subject to: \( f(x) + s = 0 \)

The formulation in Equation 4.12 is the *barrier* formulation [9] of the minimum distance problem of Equation 4.9 and \( \mu \) is called the barrier parameter, with \( \mu > 0 \). Equation 4.12 differs from Equation 4.9 in that the non negativity constraints on \( s \) are not present explicitly, but are implicit in the objective. The Lagrangian for the above constrained optimization problem can be written as

\[
f_0(x) - \mu \sum_{i=1}^{n} \ln(s_i) + \lambda^T (f(x) + s) \quad (4.13)
\]

where \( \lambda \) is the vector of Lagrange multipliers. Thus, the KKT conditions can be written as

\[
\nabla f_0(x) + (\nabla f(x))^T \lambda = 0 \\
f(x) + s = 0 \\
LSe - \mu e = 0 \quad (4.14)
\]

The above equation represents a system of \( 2n + 6 \) nonlinear equations in \( 2n + 6 \) variables and can be approximately solved for a given \( \mu \). Note that Equation 4.11 and Equation 4.14 differ in the complementarity conditions. As the barrier parameter \( \mu \) approaches 0, the KKT conditions for the barrier problem (Equation 4.14) approach the KKT conditions of the original problem (Equation 4.11). See Figure 4.3 for a schematic illustration of the interior point method.

For our proximity query problem, feasible interior point methods generate iterates that yield points guaranteed to lie inside the objects and converge towards the closest points on the boundaries of the objects. See the example proximity query in Figure 4.4.

The general structure of interior point methods is indicated in Algorithm 2, where termination criterion 1 is the ending condition for the whole problem (Equation 4.11) and termination criterion 2 is the ending condition for approximately solving Equation 4.14.
Figure 4.3: Schematic illustration of the interior point method for a path following algorithm. The convex region represents the feasible set. The central path is an arc of strictly feasible points that solve Equation 4.14 as the parameter $\mu$ approaches 0. The progress of the iterates generated by the interior point solver is indicated by the polygonal line connecting them. The iterates are guaranteed to lie within a neighborhood, represented by the circular ball, of the central path.

for the current value of $\mu$. The outer while loop determines the number of times $\mu$ has to be updated, i.e., the number of times Equation 4.14 has to be approximately solved for the sequence of $\mu$ values. The inner while loop is a variant of Newton’s method used for approximately solving Equation 4.14 for a fixed value of $\mu$. The different interior point implementations (KNITRO [11, 116], LOQO [111], IPOPT [115]) vary in the way they calculate the step lengths for a particular value of $\mu$, the termination criteria they use, and the way in which they update $\mu$.

### 4.6 Computational Complexity

The total cost of solving a problem using Algorithm 2 is the product of the total number of iterations (considering both loops) and the computational effort in solving the system of linear equations to determine the Newton direction in each iteration. The system of linear equations to be solved to determine the Newton direction is

$$
\begin{pmatrix}
A_1 & A_2^T & 0_{6\times n} \\
A_2 & 0_{n\times n} & I_{n\times n} \\
0_{n\times n} & S & L
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta \lambda \\
\Delta s
\end{pmatrix}
=
\begin{pmatrix}
-F_1 \\
-F_2 \\
-F_3
\end{pmatrix}
$$

(4.15)
Figure 4.4: Example illustrating the sequence of closest point estimates generated by the interior point method for two 2D superquadric objects, with indices $(23, 11), (76, 71)$ and semiaxes 1. The iterates of the interior point method are mapped to corresponding points in the objects.

where $A_1 = \nabla^2 f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla^2 f_i(x)$ is a $6 \times 6$ matrix, $A_2 = \nabla f(x)$ is a $n \times 6$ matrix, the definitions of $S$ and $L$ are the same as before, and $F_1 = \nabla f_0(x) + (\nabla f(x))^T \lambda$, $F_2 = f(x) + s$, $F_3 = LSe - \mu e$.

In general, the computational cost of solving a system of $n$ linear equations in $n$ unknowns is $O(n^3)$. However, we now establish that the system of linear equations can be solved in $O(n)$ time. By simple algebraic manipulation of the above equations, we obtain the following formulas for $\Delta x$, $\Delta \lambda$, and $\Delta s$:

\[
\begin{align*}
\Delta x &= G^{-1}(-F_1 - A_2^T (S^{-1}L)(F_2 + L^{-1}F_3)) \\
\Delta \lambda &= (S^{-1}L)(A_2 \Delta x - F_2 + L^{-1}F_3) \\
\Delta s &= -L^{-1}(F_3 + S\Delta \lambda)
\end{align*}
\]  

(4.16)

where $G = A_1 + A_2^T (S^{-1}L)A_2$. Since $G$ is a $6 \times 6$ matrix, $G^{-1}$ can be computed in constant time. Moreover, $S$ and $L$ are $n \times n$ diagonal matrices, so $S^{-1}$ and $L^{-1}$ can be computed in linear time. Thus we can compute all the inverses in $O(n)$. Moreover, noting the dimensions of $A_1$ and $A_2$, we can see by inspection that the matrix multiplication also requires $O(n)$ operations. So Equation 4.16 can be evaluated in $O(n)$ time, or in other
Algorithm 2 Interior point algorithm

Input: initial strictly feasible \( x_0 \), initial barrier parameter \( \mu_0 \), specified tolerance \( \epsilon \), and KKT equations

Output: Closest points solution \( x \)

\[ k \leftarrow 0 \]

while termination criterion 1 not satisfied do

while termination criterion 2 not satisfied do

Solve the system of linear equations // determine Newton direction

Determine step length \( \alpha_k \) by line search

\[ x_{k+1} \leftarrow x_k + \alpha_k \Delta x_k \]

\[ s_{k+1} \leftarrow s_k + \alpha_k \Delta s_k \]

\[ \lambda_{k+1} \leftarrow \lambda_k + \alpha_k \Delta \lambda_k \]

\[ k \leftarrow k + 1 \]

end while

\[ \mu \leftarrow c\mu \quad // \ c < 1, \ may \ be \ constant \ or \ adaptive \]

end while

return \( x_k \)

words, the computation of the Newton step takes \( O(n) \) time. Note that we have not made any assumptions regarding the primitive surface describing the object. Thus this analysis is valid for any implicit surface (including planes, quadrics, superquadrics, hyperquadrics, etc.) and for intersections of these implicit surfaces.

For general functions, there is no known bound on the total number of iterations (including both while loops). However, if the log barrier function of the implicit surface constraints is a self-concordant function (refer to Section 4.3), the number of Newton iterations (which is the number of times Equation 4.15 must be solved) is polynomial in a parameter depending on the structure of the function. For polyhedral constraints and quadric constraints the number of Newton iterations required for converging to the optimal solution is \( O(n^{0.5}) \). This implies that the theoretical complexity of our approach for polyhedra and quadrics is \( O(n^{1.5}) \). Although algorithms with theoretical linear time guarantees are available for polyhedra, for the case of quadrics, as far as we know, this is the best known bound. Moreover, our experiments indicate the algorithm exhibits linear time behavior in practice, as shown in Figure 4.5 and Figure 4.6.

For superquadrics and hyperquadrics, the log barrier function might not be self-concordant in the general case. However, note that superquadric and hyperquadric functions have self-concordant barriers because they define convex regions, and every convex
region has a self-concordant barrier (its universal barrier). If this barrier is too hard to find to be useful computationally, an alternative is to decompose the function into simpler functions (for example, as second order cones [86]) such that the sum of the barriers for the simpler functions gives a barrier for the original superquadric or hyperquadric. However these representations may lead to computationally slower solutions due to the increased number of variables and constraints. Glineur and Terlaky [47] provide self-concordant formulations for $l_p$-norm minimization that apply to superquadrics and hyper-
quadrics. However the computational performance of these formulations has not yet been explored in the literature. Moreover, the observed time complexity of the interior point algorithm is linear for this class of shapes (Figure 4.7), which implies that in practice the number of iterations is constant, i.e., independent of the size of the problem. The observed linear time behavior of the interior point algorithm even without self-concordant representations further justifies the use of an interior point-based solver for this generic nonlinear programming formulation.

4.7 Continuous Proximity Queries for Translating Objects

We now address the problem of continuous proximity queries for two linearly translating objects. Such queries can be useful in identifying feasible object motions during assembly planning. The goal is to determine the exact time at which the two moving objects are closest, without discrete sampling of their configurations. This computation of the closest distance between two swept objects is closely related to the problem of continuous collision detection ([15], [12], [120], [95], [91], [110]), where the time of first contact between two colliding objects is to be determined; however most prior work has been restricted to polyhedral objects. The advantage of such continuous collision detection methods is the ability to detect collisions even for fast moving objects in the presence of thin obstacles.
We address the continuous collision detection problem for linearly translating objects by solving two related convex optimization problems. Assume the objects are moving along piecewise linear paths. Let \( X \) and \( Y \) be two objects described by \( f_X(x_i) \leq 0 \) and \( f_Y(y_i) \leq 0 \). Let the two objects be linearly translating along the directions specified by the unit vectors \( \hat{g}_x \) and \( \hat{g}_y \) with constant velocities \( v_x \) and \( v_y \) respectively. The following optimization problem finds the minimum distance between the two objects in the time interval \([0, t_{\text{max}}]\), where each object is moving along a single line segment. If the minimum distance is greater than zero, the solution provides the closest points and the time \( t \) at which the objects are closest. See Figure 4.8.

\[
\begin{align*}
\text{Minimize} & \quad \|x_g - y_g\|_2^2 \\
\text{subject to:} & \quad x_g = R_x x_l + p_x + v_x t \hat{g}_x \\
& \quad y_g = R_y y_l + p_y + v_y t \hat{g}_y \\
& \quad f_X(x_i) \leq 0 \\
& \quad f_Y(y_i) \leq 0 \\
& \quad t \leq t_{\text{max}} \\
& \quad t \geq 0
\end{align*}
\]

(4.17)

When the objects intersect, the problem above has multiple solutions corresponding to zero distance. The time of first contact can be obtained by solving another convex optimization problem using the solution of Equation 4.17. Let \( Q \) be the set of intersecting points and \( q \in Q \) be a point specified in the global coordinate system. To obtain the time of first contact, we solve the problem below, starting from an initial feasible guess that is the solution of Equation 4.17.
Figure 4.8: Computing the instant of closest distance using the continuous proximity query. The bold blue line connects the closest points on the two objects, as they translate along the indicated line segments.

Minimize \( t \)

subject to:

\[
\begin{align*}
q &= R_x x_t + p_x + v_x t \hat{g}_x \\
q &= R_y y_t + p_y + v_y t \hat{g}_y \\
f_X(x_t) &\leq 0 \\
f_Y(y_t) &\leq 0 \\
t_p &\geq t \geq 0
\end{align*}
\]  

(4.18)

where \( t_p \) is the time obtained from the solution of Equation 4.17. See the example in Figure 4.9.

We now establish that the computational complexity of solving the Newton step in the continuous collision detection problem along a single linear segment is \( O(n) \), which is the same as for the static query problem. We can eliminate \( x_t \) and \( y_t \) from Equation 4.17 and can write it in the form of Equation 4.9 where \( x \) is a \( 7 \times 1 \) column vector that includes \( t \). Thus in Equation 4.15, \( A_1 \) is a \( 7 \times 7 \) matrix and \( A_2 \) is an \( n \times 7 \) matrix. This implies that \( G \) is still a constant sized \( 7 \times 7 \) matrix and the complexity argument for solving Equation 4.16 in Section 4.6 applies directly. Similarly in Equation 4.18, \( x \) is a \( 4 \times 1 \) vector consisting of \( q \) and \( t \). Thus \( A_1 \) is a \( 4 \times 4 \) matrix and \( A_2 \) is an \( n \times 4 \) matrix and
Computing the time of first contact using the continuous proximity query gives the solution to the continuous collision detection problem.

\( G \) is a \( 4 \times 4 \) matrix. Hence the computation of the Newton step in both problems takes \( O(n) \) time irrespective of the implicit primitive. Moreover, as the constraints have a self-concordant log barrier function for the case of planes and quadrics, the overall complexity is \( O(n^{1.5}) \) in these cases.

**Obtaining the collision interval for motion with known linear paths:** Computing the path intervals over which two robots can collide is useful in multiple robot coordination problems where the robots travel along known paths [86]. When we are given the paths traversed by the objects and wish to determine the path intervals over which the objects could collide with each other, we solve the following optimization problem.

\[
\begin{align*}
\text{Minimize} & \quad s_x \\
\text{subject to:} & \quad x_g = R_x x_l + p_x + s_x \hat{g}_x \\
& \quad y_g = R_y y_l + p_y + s_y \hat{g}_y \\
& \quad f_X(x_l) \leq 0 \\
& \quad f_Y(y_l) \leq 0 \\
& \quad f_X(y_g) \leq 0 \\
& \quad f_Y(x_g) \leq 0 \\
& \quad s_x, s_y \geq 0
\end{align*}
\] (4.19)
Here $s_x$ and $s_y$ represent the path lengths traversed by the objects. The inequalities $f_X(y_g) \leq 0$ and $f_Y(x_g) \leq 0$ describe all points in the intersection of both objects, and also encode the constraint that the distance between the objects is zero. In fact, we have to solve three more variations of the above problem with the objective function minimizing $-s_x$, $s_y$, and $-s_y$ to find the minimum and maximum values of $s_x$ and $s_y$. These define corresponding collision intervals over the path for each object.

### 4.8 Results

![Figure 4.10: Example objects. Objects I–III are superquadrics, IV is an intersection of superquadrics and halfspaces, and V–VI are hyperquadrics.](image)

We now present results illustrating our approach. To solve the distance computation problem, we used KNITRO 5.0, a commercially available interior point based solver ([11, 116]). We use the primal-dual feasible interior point method in KNITRO, where all the iterates are feasible. We have an initial feasible solution trivially from points at the centers of the objects. The barrier parameter $\mu$ is initially set to 0.1 and reduced by an adaptive factor at each iteration based on the complementarity gap. For each value of $\mu$ the system
of nonlinear equations is approximately solved by Newton’s method with the step size determined by a trust region method \[80\].

We depict six example objects in Figure 4.10, three of which are superquadrics. The indices and semiaxes of the three superquadrics are \((\frac{4}{3}, \frac{7}{5}, \frac{15}{13})\) and \((1, 0.7, 1.5)\) for Object I (a diamond), \((\frac{23}{11}, \frac{11}{5}, \frac{17}{7})\) and \((1, 1, 1.7)\) for Object II (a soda can), and \((\frac{26}{7}, \frac{71}{5}, \frac{179}{13})\) and \((1, 1, 1.5)\) for Object III (a rounded cuboid). Object IV models a computer mouse and is represented as an intersection of a superquadric and 4 half spaces. The indices and semiaxes of the superquadric are \((\frac{23}{11}, \frac{11}{5}, \frac{17}{7})\) and \((2, 1, 1.7)\). The half spaces are \(x_1 \geq \frac{-1}{2}\), \(x_1 \leq \frac{1}{2}\), \(x_2 \geq -0.75\), and \(x_3 \geq 0.4\) where \(x_1, x_2, x_3\) are the local coordinates of the object. Object V is (the convex hull of) a rounded hexagonal nut modeled as the hyperquadric

\[
|\eta_2|^6 + |\eta_1 + 0.5\eta_2|^6 + |\eta_1 - 0.5\eta_2|^6 + |2.5\eta_3|^2 \leq 1.
\]

Object VI is a pyramid modeled as the hyperquadric

\[
|x_1 + x_3|^6 + |x_2 + x_3|^6 + |x_3|^6 + |x_1 - x_3|^6 + |x_2 - x_3|^6 \leq 1.
\]

The run time performance of the algorithm on the example objects is shown in Table 4.1, with some test cases depicted in Figure 4.2. All data was obtained on a 2.2 GHz Athlon 64 X2 4400+ machine with 2 GB of RAM. The running times demonstrate that the distance computation rate is about 1 kHz, which is sufficiently fast for real-time dynamic simulations and interactive haptic simulations. We also generated triangulations of these objects with about 19,000 triangles and found that the distance computation time (not collision detection time) taken by PQP \[63\], a popular collision detection software, was comparable for our examples. Note that our randomly generated object configurations do not provide the benefits of coherence. We also compared our algorithm on quadric surfaces against SOLID \[109\], which supports proximity queries for quadrics without discretization. SOLID runs about 80 times faster than our approach for the case of ellipsoids. However, our algorithm has a theoretical guarantee, and SOLID cannot deal with general implicit surfaces like superquadrics or hyperquadrics without discretization.

The timing data for translation-only continuous collision detection is shown in Table 4.2. In cases where there is no collision, the query time is the time to solve Equa-
tion 4.17. In cases with collision, we compute the exact time of first contact by solving the two optimization problems described in Equation 4.17 and Equation 4.18.

**Deforming Objects:** As stated in Section 4.4, the linear constraints in Equation 4.7 can represent a general affine transformation. Thus this framework can easily handle global deformations such as nonuniform scaling. Figure 4.11 shows nonuniform scaling of Object I and Object III in 10 steps. The scaling matrices used for the two objects are diagonal matrices whose diagonal entries are (1, 0.5, 2) and (0.5, 3, 0.67) respectively. The approach can also handle any global deformation where the convexity of the object is preserved. For example, consider the global shape deformation for superquadrics (or hyperquadrics) due to changes in the indices. Note that if instead a polygonal representation of the object had been used for this kind of shape change, the polygonal representation would have to be recomputed. Figure 4.12 shows three snapshots of Object I being deformed to Object III in 10 steps. Table 4.3 shows the average distance computation times for both nonuniform scaling and index deformation. This data shows that both affine and index changing deformations can be performed with similar running times to the rigid object proximity query.

**Numerical Robustness Issues:** We have observed a small number of cases where KNITRO failed to converge to the optimal solution. For most objects, the failure rates were typically less than 0.01%. The largest failure rate observed was 0.4% when Object I was one of the objects. This may be because (as is evident from our formulation) the algorithm needs the second derivatives of the functions representing the objects, but the second derivative does not exist everywhere for Object I. Despite this, the solver converges to the optimal solution in most cases. Switching to a variant of the interior point method that uses a conjugate gradient method, available within KNITRO, enabled the solver to converge for some of the failure cases. Therefore adaptively using the two variants of the method could improve robustness even further.

### 4.9 Conclusion

This chapter demonstrates that recently developed interior point algorithms are particularly effective for computing the distance between two or more convex objects, where each object is described as an intersection of implicit surfaces. This proximity query
Table 4.1: Sample run times, in milliseconds, for proximity queries between pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times over 100,000 random configurations. All data was obtained on a 2.2 GHz Athlon 64 X2 4400+ machine with 2 GB of RAM.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Number of constraints</th>
<th>Proximity query time (millisecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 I, II</td>
<td>2</td>
<td>0.84</td>
</tr>
<tr>
<td>2 I, III</td>
<td>2</td>
<td>0.91</td>
</tr>
<tr>
<td>3 II, III</td>
<td>2</td>
<td>0.70</td>
</tr>
<tr>
<td>4 III, IV</td>
<td>6</td>
<td>0.85</td>
</tr>
<tr>
<td>5 II, IV</td>
<td>6</td>
<td>0.76</td>
</tr>
<tr>
<td>6 III, V</td>
<td>2</td>
<td>0.78</td>
</tr>
<tr>
<td>7 V, VI</td>
<td>2</td>
<td>0.89</td>
</tr>
<tr>
<td>8 III, VI</td>
<td>2</td>
<td>0.89</td>
</tr>
</tbody>
</table>

Table 4.2: Sample continuous proximity query run times between pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times over 100,000 pairs of random configurations. For the time of first contact queries, only those configuration pairs that resulted in collisions were used and the reported query time is the total query time for solving both problems.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Query type</th>
<th>Number of constraints</th>
<th>Query time (millisecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I, III</td>
<td>Closest distance</td>
<td>2</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>Time of first contact</td>
<td>2</td>
<td>2.03</td>
</tr>
<tr>
<td>II, III</td>
<td>Closest distance</td>
<td>2</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>Time of first contact</td>
<td>2</td>
<td>1.51</td>
</tr>
</tbody>
</table>

Table 4.3: Sample proximity query run times between deforming pairs of objects using KNITRO 5.0. The run times were computed for each pair by averaging the run times at each of 10 steps in the shape change, over 100,000 random configurations.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Type of Deformation</th>
<th>Number of constraints</th>
<th>Proximity query time (millisecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I, III</td>
<td>Nonuniform Scaling</td>
<td>2</td>
<td>1.04</td>
</tr>
<tr>
<td>II, III</td>
<td>Nonuniform Scaling</td>
<td>2</td>
<td>0.75</td>
</tr>
<tr>
<td>I → III, III</td>
<td>Index Change</td>
<td>2</td>
<td>0.87</td>
</tr>
<tr>
<td>II → III, III</td>
<td>Index Change</td>
<td>2</td>
<td>0.84</td>
</tr>
</tbody>
</table>
Figure 4.11: Proximity queries on deforming (superquadric) objects, with the deformation described by monotonic scaling. The deformation is performed in 10 steps. (a) The original objects. (b) The objects midway through the scaling. (c) The scaled objects.

Figure 4.12: Proximity queries on deforming superquadric objects, with the deformation governed by monotonic change of exponents. Object I is transformed to Object III in 10 steps. (a) The original objects. (b) Midway through the deformation, deformed Object I has indices \((\frac{196}{45}, \frac{39}{5}, \frac{97}{13})\). (c) The final objects.

approach complements the proximity query approaches of GJK [45], Lin-Canny [69], and similar algorithms, since they focus on polyhedra while we focus on smooth implicit surfaces. We demonstrated our algorithm on example implicit surface objects including convex polyhedra, quadrics, superquadrics, hyperquadrics, and their intersections. The global convergence properties of interior point algorithms make them robust even in the absence of any initial information about the closest points. For the class of (convex polyhedra and) convex quadric surfaces, this approach has a theoretical complexity of \(O(n^{1.5})\), where \(n\) is the number of implicit function constraints. To the best of our knowledge, this is the first bound on the running time of proximity queries for convex quadrics. Moreover, the practical running time behavior is linear in the number of constraints for all the
classes of implicit surfaces that we have studied. The speed at which distance computations can be performed enables real-time dynamic simulations and haptic interactions at 1 KHz rates.

An important additional advantage of this method is that it provides a uniform framework for distance computation between convex objects described as arbitrary intersections of polyhedra, quadrics, or any convex implicit surface. Furthermore, within this framework we can handle global affine deformations of implicit surface objects, and index change deformations of superquadrics (or hyperquadrics) without significant computational overhead. Finally, we show that continuous collision detection for linearly translating implicit surface objects can be performed by solving two related convex optimization problems. For polyhedra and quadrics, we establish that the computational complexity of this continuous collision detection problem is $O(n^{1.5})$.

There are several directions for future work. One direction is to explore alternative interior point algorithms (LOQO [111] and IPOPT [115], for example) to test their performance on the minimum distance problem. Performing warm starts, where a good initial estimate for the solution is available, can potentially improve the running time when there is coherence. This may be best achieved by a combination of interior point methods and sequential quadratic programming approaches. Another future direction is to extend this approach to nonconvex objects, modeled as unions of convex shapes and incorporate it in a hierarchical framework. Longer term directions for future research include tracking closest points continuously for haptics applications, and extending this approach to performing continuous collision detection with both rotational and translational motion. In the next chapter, we integrate the distance computation problem with a nonlinear complementarity formulation of multibody dynamics that allows us to develop a geometrically implicit time-stepper.
CHAPTER 5
An Implicit Time-Stepping Method for Multibody Systems with
Intermittent Contact

5.1 Introduction

To automatically plan and execute tasks involving intermittent contact, one must be able to accurately predict the object motions in such systems. Applications include haptic interactions, collaborative human-robot manipulation, such as rearranging the furniture in a house, as well as industrial automation, such as simulation of parts feeders. Due to the intermittency of contact and the presence of stick-slip frictional behavior, dynamic models of such multibody systems are inherently (mathematically) nonsmooth, and are thus difficult to integrate accurately. In fact, commercially available software systems such as Adams, have a difficult time simulating any system with unilateral contacts. Users expect to spend considerable effort in a trial-and-error search for good simulation parameters to obtain believable, not necessarily accurate, results. Even the seemingly simple problem of a sphere rolling on a horizontal plane under only the influence of gravity is challenging for commercial simulators.

The primary sources of stability and accuracy problems are polyhedral approximations of smooth bodies, the decoupling of collision detection from the solution of the dynamic time-stepping subproblem, and approximations to the quadratic Coulomb friction model. This chapter focuses on the development of geometrically implicit optimization-based time-stepper for dynamic simulation. More specifically, the state-of-the-art time-steppers [106, 103, 72] use geometric information obtained from a collision detection algorithm at the current time, and the state of the system at the end of the time step is computed (by solving a dynamics time step subproblem) without modifying this information. Thus, state-of-the-art time-steppers can be viewed as explicit methods with respect to geometric information. We develop the first time-stepping method that is implicit in the geometric information (when the distance function is not available in closed form) by incorporating body geometry in the dynamic time-stepping subproblem. In other words, our formulation solves the collision detection and dynamic stepping problem in the same
time-step, which allows us to satisfy contact constraints at the end of the time step. The resulting subproblem at each time-step will be a mixed nonlinear complementarity problem and we call our time-stepping scheme a \textit{geometrically implicit} time-stepping scheme. We assume the objects to be convex objects described as an intersection of implicit surfaces. We first present the method for rigid bodies and then extend it to locally compliant or quasi-rigid bodies (where each body consists of a rigid core surrounded by a thin compliant shell [101, 102, 103, 84]). This method also takes into consideration other important nonlinear elements such as quadratic Coulomb friction. This method will provide a baseline for understanding and quantifying the errors incurred when using a geometrically explicit method and when making various linearizing approximations. Our ultimate goal is to develop techniques for automatically selecting the appropriate method for a given application, and to guide method switching, step size adjustment, and model approximations on the fly.

This chapter is organized as follows. In Section 5.2, we survey the relevant literature. In Section 5.3, we present the non-penetration condition for the contact constraints assuming the objects to be rigid. Thereafter, in Section 5.4, we modify these contact constraints to include compliant contacts with limits on the maximum allowable deflection. The discrete time dynamics model along with the contact constraints form a mixed nonlinear complementarity problem at each time-step. In Section 5.5, we give examples that validate and elucidate our time-stepping scheme. Finally in section 6.7, we present our conclusions and lay out the future work. The work in this chapter has appeared in [19, 20].

5.2 Related Work

The three primary modeling approaches for multibody systems with unilateral contacts are based on three different assumptions about the flexibility of the bodies. The assumptions from most to least realistic (and most to least computationally complex) are: 1) the bodies are fully deformable, 2) the bodies have rigid cores surrounded by compliant material, 3) the bodies are fully rigid. The first assumption leads to finite element approaches, for which one must solve very large difficult complementarity problems or variational inequalities at each time step. The second assumption leads to smaller subproblems that can be solved more easily [103, 84], but suitable values of the parameters of
the compliant layer are difficult to determine. The assumption of rigid bodies leads to the smallest subproblems and avoids the latter problem of determining material deformation properties.

Independent of the rigidity assumptions of the bodies, the methods developed to date for dynamic simulation have one problem in common that fundamentally limits their accuracy – they are not implicit with respect to the relevant geometric information. For example, at the current state, a collision detection routine is called to determine separation or penetration distances between the bodies, but this information is not incorporated as a function of the unknown future state at the end of the current time step. A goal of a typical time-stepping scheme is to guarantee consistency of the dynamic equations and all model constraints at the end of each time step. However, since the geometric information at the end of the current time step is approximated from that at the start of the time step, the solution will be in error.

Early time-steppers used linear approximations of the local geometry at the current time [106, 1]. Thus each contact was treated as a point on a plane or a line on a (non-parallel) line and these entities were assumed constant for the duration of the time step. Besides being insufficiently accurate in some applications, some unanticipated side-effects arose [37].

Increased accuracy can be obtained in explicit schemes by including curvature information. This was done by Liu and Wang [72] and Pfeiffer and Glocker [87] by incorporating kinematic differential equations of rolling contact (Montana [78]). Outside the complementarity formalism, Kry and Pai [62] and Baraff [3] also make use of the contact kinematics equations in dynamic simulations of parametric and implicit surface objects respectively.

The method of Tzitzouris [56] is the only geometrically implicit method developed to date, but unfortunately it requires that the distance function between the bodies and two levels of derivatives be available in closed form. However, it is possible that this method could run successfully after replacing the closed-form distance functions with calls to collision detection algorithms and replacing derivatives with difference approximations from multiple collision detection calls, but polyhedral approximations common to most collision detection packages would generate very noisy derivatives. To our knowledge,
such an implementation has not been attempted. One other problem with Tzitzouris’ method is that it adapts its step size to precisely locate every collision time. While this is a good way to avoid interpenetration at the end of a time step, it has the undesirable side-effect of forcing the step size to be unreasonably small when there are many interacting bodies [75]. The method we propose does not suffer from this problem.

**Collision Modeling for Nominally Rigid Body Systems:** Frictional collisions between rigid bodies have a long history in mechanics [94, 58]. Here, we give an overview of the basic approaches and refer the reader to a recent survey article [42] for a more comprehensive review. There are two primary approaches to modeling collisions: coefficient of restitution based approaches and force based methods. In the former, the process of energy transfer and dissipation during collision is modeled by various coefficients relating the velocity (or impulses) before contact to that after contact. However, the extension of these concepts to situations with multiple contacts is not straightforward. The force based approaches use a compliant contact model to compute the contact forces where the contact compliance is modeled as a (linear/nonlinear) spring-damper system. In the simplest model (known as Kelvin-Voigt model or linear spring-damper model), the normal contact force is given by a linear function of the deformation and the rate of deformation \( F = k\delta + c\dot{\delta} \) i.e., the flexibility of the body is lumped as a linear spring (with spring constant \( k \)) and damper (with damping coefficient \( c \)). The limitations of the linear model are documented in [42]. Hertz introduced a nonlinear model of the form \( F = k\delta^n \), where \( n \) is a constant [55]. This model was extended to a nonlinear spring-damper model by Hunt and Crossley [52] of the form \( F = k\delta^n + c\delta^p\dot{\delta}^q \), where \( p, q \) are constants. The models presented above are believed to be of increasing accuracy but there are more unknown constants dependent on geometry of the objects and material properties that have to be determined experimentally (except for some simple cases). This is a general feature of all proposed contact compliance models. In [117] a continuum model of the deformations at each contact is used. Song and Kumar [102] have used a 3D linear distributed contact model to compute the contact forces. In this chapter we use a lumped 3D linear spring-damper to model the contact compliance similar to [61]. However, we note that we could have replaced this with a lumped nonlinear model if required. We use an elliptic dry friction law [108] that is a generalization of Coulomb’s friction law to model the friction at
the contact.

5.3 Contact Constraint for Rigid Bodies

In this section we rewrite the contact condition (Equation 2.3) as a complementarity condition in the work space, combine it with an optimization problem to find the closest points, and prove that the resultant system of equations ensures that the contact constraints are satisfied. Let us consider the $i$th contact. For ease of exposition, we first present the case where each object is a convex object described by a single implicit surface. A more general formulation where each object is described by an intersection of implicit surfaces is given in the next subsection.

5.3.1 Objects described by a single convex function

Let the two objects be defined by convex functions $f(\xi_1) \leq 0$ and $g(\xi_2) \leq 0$ respectively, where $\xi_1$ and $\xi_2$ are the coordinates of points in the two objects. Let $a_1$ and $a_2$ be the closest points on the two objects. The equation of an implicit surface has the property that for any point $x$, the point lies inside the object for $f(x) < 0$, on the object surface for $f(x) = 0$, and outside the object for $f(x) > 0$. Thus, we can define the gap function in work space as either $f(a_2)$ or $g(a_1)$ and write the complementarity conditions as either one of the following two conditions:

$$0 \leq \lambda_n \perp f(a_2) \geq 0$$
$$0 \leq \lambda_n \perp g(a_1) \geq 0$$

(5.1)

where $a_1$ and $a_2$ are given by

$$\text{argmin} \ \{ \frac{1}{2} \| \xi_1 - \xi_2 \|^2 : f(\xi_1) \leq 0, \ g(\xi_2) \leq 0 \}$$

(5.2)
The Karush-Kuhn-Tucker (KKT) optimality conditions of Equation 5.2 that the solutions \(a_1\) and \(a_2\) must satisfy are given by the following system of algebraic equations.

\[
\begin{align*}
\mathbf{a}_1 - \mathbf{a}_2 &= -\hat{l}_1 \nabla f(\mathbf{a}_1) \\
\mathbf{a}_1 - \mathbf{a}_2 &= \hat{l}_2 \nabla g(\mathbf{a}_2) \\
f(\mathbf{a}_1) + s_1 &= 0 \\
g(\mathbf{a}_2) + s_2 &= 0 \\
0 &\leq \hat{l}_1 \perp s_1 \geq 0 \\
0 &\leq \hat{l}_2 \perp s_2 \geq 0
\end{align*}
\]

(5.3)

where \(\hat{l}_1, \hat{l}_2\) are the Lagrange multipliers and \(s_1, s_2\) are the slack variables. By eliminating the slack variables and some rearrangement, the system of equations can be equivalently written as:

\[
\begin{align*}
\mathbf{a}_1 - \mathbf{a}_2 &= -\hat{l}_1 \nabla f(\mathbf{a}_1) \\
\hat{l}_1 \nabla f(\mathbf{a}_1) &= -\hat{l}_2 \nabla g(\mathbf{a}_2) \\
0 &\leq \hat{l}_1 \perp -f(\mathbf{a}_1) \geq 0 \\
0 &\leq \hat{l}_2 \perp -g(\mathbf{a}_2) \geq 0
\end{align*}
\]

(5.4) (5.5) (5.6) (5.7)

The geometric meaning of the equations 5.4 and 5.5 is that the normals to the two surfaces at their closest points are aligned with the line joining the closest points. The solution to the system of equations 5.4 to 5.7 gives the closest point when the two objects are separate. However, when \(\mathbf{a}_1 = \mathbf{a}_2\), there are multiple solutions to the KKT conditions. The solution is either the touching point of the two surfaces or a point lying in the intersection set of the two objects 5.1. The complementarity conditions in equation 5.1 ensure that the points in the interior of the objects are not feasible solution to the overall problem. However, the points on the intersecting surface as well as the touching points are valid solutions to equations 5.4 to 5.7 and 5.1. Thus, as written, equations 5.4 to 5.7 and 5.1 do not guarantee non-penetration. We want to form a system of equations that is equivalent to the KKT conditions (equations 5.4 to 5.7) when the distance between the objects is non-zero but only gives the touching solution when the distance is zero.
Figure 5.1: Three Contact cases: (left) Objects are separate (middle) Objects are touching (right) Objects are intersecting.

Proposition: The equations 5.8 - 5.11 are equivalent to the KKT conditions when the distance between the objects is non-zero. Moreover, combined with complementarity condition in equation 5.12, it gives only the touching solution when the distance between the objects is zero.

\[ a_1 - a_2 = -l_1 \nabla f(a_1) \quad (5.8) \]
\[ \nabla f(a_1) = -l_2 \nabla g(a_2) \quad (5.9) \]
\[ 0 \leq l_1 \perp -f(a_1) \geq 0 \quad (5.10) \]
\[ 0 \leq l_2 \perp -g(a_2) \geq 0 \quad (5.11) \]
\[ 0 \leq \lambda \perp f(a_2) \geq 0 \quad (5.12) \]

Proof: Let us first consider the case when the distance between the objects is greater than zero. In this case \( a_1 \neq a_2 \). Since the gradient vectors cannot be zero and \( \hat{l}_1, \hat{l}_2 \) in equation 5.6, 5.7 are constrained to be non-negative, \( \hat{l}_1, \hat{l}_2 \) are strictly positive (or non-zero) in this case (from equations 5.4 and 5.5). Therefore equation 5.5 can be written as \( \nabla f(a_1) = -\frac{\hat{l}_2}{\hat{l}_1} \nabla g(a_2) \). Using \( l_1 = \hat{l}_1 \) and \( l_2 = \frac{\hat{l}_2}{\hat{l}_1} \), the equations 5.4 to 5.6 can be written as equations 5.8 to 5.10. Since \( \hat{l}_1 > 0 \) we can rewrite the complementarity condition in equation 5.7 as

\[ \hat{l}_2(-g(a_2)) = 0, \quad \hat{l}_2 \geq 0, \quad -g(a_2) \geq 0 \]
\[ \iff \frac{\hat{l}_2}{\hat{l}_1}(-g(a_2)) = 0 \quad \frac{\hat{l}_2}{\hat{l}_1} \geq 0, \quad -g(a_2) \geq 0 \quad (5.13) \]
\[ \iff l_2(-g(a_2)) = 0 \quad l_2 \geq 0, \quad -g(a_2) \geq 0 \]
Thus equations 5.8 to 5.11 are equivalent to equations 5.4 to 5.7 when the distance between the objects is greater than zero.

When the distance between the objects is equal to zero, \( a_1 = a_2 \). Thus \( l_1 = 0 \) from equation 5.8 which implies \( f(a_1) \leq 0 \) from equation 5.10. The equation 5.9 implies that \( l_2 > 0 \) and that gives \( g(a_2) = 0 \) from equation 5.11. Thus the system 5.8 to 5.11 gives solution points that lies on the object 2 and may lie either inside or on object 1 with the gradients of the functions defining the surfaces in opposite directions at the common point. However, equation 5.12 implies that the point \( a_2 \) cannot lie within the object 1. Hence the only possible solution is that \( a_1 = a_2 \) with the two objects touching each other.

\[\Box\]

**Proposition:** Equations 5.8 to 5.12 together represent the contact constraints, i.e., the two objects will satisfy the contact constraints at the end of each time step if and only if equations 5.8 to 5.12 hold together.

**Proof:** As discussed above. \( \Box \)

### 5.3.2 New Discrete Time Model

We can now rewrite equation 2.22 as a mixed NCP for the geometrically-implicit time-stepper. The vector of unknowns \( z \) can be partitioned into \( z = [u, v] \) where \( u = [\nu, a_1, a_2, p_t, p_o, p_r] \) and \( v = [l, p_n, \sigma] \). The equality constraints in the mixed NCP are:

\[
0 = -M\nu^{\ell+1} + M\nu^\ell + W_n^{\ell+1}p_n^{\ell+1} + W_t^{\ell+1}p_t^{\ell+1} + W_o^{\ell+1}p_o^{\ell+1} + \mathbf{p}_{\text{app}} + \mathbf{p}_{\text{vp}}^\ell \\
0 = (a_1^{\ell+1} - a_2^{\ell+1}) + l_1\nabla f(a_1^{\ell+1}) \\
0 = \nabla f(a_1^{\ell+1}) + l_2\nabla g(a_2^{\ell+1}) \\
0 = E_t^{\ell}U_p^{\ell+1} \circ (W_t^T)^{\ell+1}\nu^{\ell+1} + p_t^{\ell+1} \circ \sigma^{\ell+1} \\
0 = E_o^{\ell}U_p^{\ell+1} \circ (W_o^T)^{\ell+1}\nu^{\ell+1} + p_o^{\ell+1} \circ \sigma^{\ell+1} \\
0 = E_r^{\ell}U_p^{\ell+1} \circ (W_r^T)^{\ell+1}\nu^{\ell+1} + p_r^{\ell+1} \circ \sigma^{\ell+1} \\
\tag{5.14}
\]
The complementarity constraints on \( v \) are:

\[
0 \leq \begin{bmatrix}
  l_1 \\
  l_2 \\
  p_{n}^{\ell+1} \\
  \sigma_{n}^{\ell+1}
\end{bmatrix} \perp \begin{bmatrix}
  f(a_1^{\ell+1}) \\
  g(a_2^{\ell+1}) \\
  f(a_2^{\ell+1}) \\
  \zeta
\end{bmatrix} \geq 0
\]  

(5.15)

where

\[
\zeta = U_{n}^{\ell+1} \circ U_{n}^{\ell+1} - (E_{c}^{2})^{-1} (p_{t}^{\ell+1} \circ p_{t}^{\ell+1}) - (E_{o}^{2})^{-1} (p_{o}^{\ell+1} \circ p_{o}^{\ell+1}) \\
- (E_{c}^{2})^{-1} (p_{r}^{\ell+1} \circ p_{r}^{\ell+1})
\]

In the above formulation, we see \( u \in \mathbb{R}^{6n_{b}+9n_{c}} \), \( v \in \mathbb{R}^{2n_{c}} \), the vector function of equality constraints maps \([u, v]\) to \( \mathbb{R}^{6n_{b}+9n_{c}} \) and the vector function of complementarity constraints maps \([u, v]\) to \( \mathbb{R}^{2n_{c}} \) where \( n_{b} \) and \( n_{c} \) are the number of bodies and number of contacts respectively. If using convex bodies only, the upper bound on the number of contacts can be determined directly from the number of bodies, \( n_{c} = \sum_{i=1}^{n_{b}} i \).

5.3.3 Objects described by intersections of convex functions

We present here the contact conditions for the general case where each convex object is defined as an intersection of convex inequalities. Let \( f_{j}(\xi_{1}) \leq 0, j = 1, \ldots, m, \)
\( g_{j}(\xi_{2}) \leq 0, j = m + 1, \ldots, n, \) be convex functions representing the two convex objects. Since the closest point is outside the object if it is outside at least one of the intersecting surfaces, the complementarity conditions for nonpenetration can be written as either one of the following two sets of conditions:

\[
0 \leq \lambda_{in} \perp \max\{f_{j}(a_{2})\} \geq 0 \quad j = 1, \ldots m \\
0 \leq \lambda_{in} \perp \max\{g_{j}(a_{1})\} \geq 0 \quad j = m + 1, \ldots n
\]  

(5.16)
where \( a_1 \) and \( a_2 \) are the closest points on the two bodies and are given by the KKT conditions

\[
\begin{align*}
\mathbf{a}_1 - \mathbf{a}_2 &= -\sum_{i=1}^{m} \hat{\mathbf{i}}_i \nabla f_i(\mathbf{a}_1) = \sum_{j=m+1}^{n} \hat{\mathbf{j}}_j \nabla g_j(\mathbf{a}_2) \\
0 &\leq \hat{\mathbf{i}}_i \perp -f_i(\mathbf{a}_1) \geq 0 \\
0 &\leq \hat{\mathbf{j}}_j \perp -g_j(\mathbf{a}_2) \geq 0
\end{align*}
\] (5.17)

At the optimal solution only some of the constraints are active. Let \( \mathbb{I} \) be the index set of active constraints. Thus \( \hat{\mathbf{i}}_k = 0 \), if \( k \notin \mathbb{I} \) and the KKT optimality conditions can be written as the following set of nonlinear equations:

\[
\begin{align*}
\mathbf{a}_1 - \mathbf{a}_2 &= -\sum_{i \in \mathbb{I} \cap \{i\}} \hat{\mathbf{i}}_i \nabla f_i(\mathbf{a}_1) \\
\sum_{i \in \mathbb{I} \cap \{i\}} \hat{\mathbf{i}}_i \nabla f_i(\mathbf{a}_1) &= -\sum_{j \in \mathbb{I} \cap \{j\}} \hat{\mathbf{j}}_j \nabla g_j(\mathbf{a}_2) \\
0 &\leq \hat{\mathbf{i}}_i \perp -f_i(\mathbf{a}_1) \geq 0 \\
0 &\leq \hat{\mathbf{j}}_j \perp -g_j(\mathbf{a}_2) \geq 0
\end{align*}
\] (5.18)

Equations 5.16 and 5.18 together represent the contact constraints as long as \( a_1 \neq a_2 \). Using arguments similar to the single surface case in Section IV we can see that it is not possible to distinguish between touching points and intersecting points using the above formulation. In this case also, we can rewrite Equation 5.18 suitably by dividing throughout by one of the non-zero lagrange multipliers to eliminate the intersection point solutions.

\[
\begin{align*}
\mathbf{a}_1 - \mathbf{a}_2 &= -(\nabla f_{k_1}(\mathbf{a}_1) + \sum_{k \in \{1 \setminus k_1\} \cap \{i\}} l_k \nabla f_k(\mathbf{a}_1)) \\
\nabla f_{k_1}(\mathbf{a}_1) + \sum_{k \in \{1 \setminus k_1\} \cap \{i\}} l_k \nabla f_k(\mathbf{a}_1) &= -\sum_{k \in \mathbb{I} \cap \{j\}} l_k \nabla g_k(\mathbf{a}_2) \\
0 &\leq \hat{\mathbf{i}}_i \perp -f_i(\mathbf{a}_1) \geq 0 \\
0 &\leq \hat{\mathbf{j}}_j \perp -g_j(\mathbf{a}_2) \geq 0
\end{align*}
\] (5.19)
Proposition: Equation 5.16 and 5.19 together represent the nonpenetration constraints, i.e., the two objects will satisfy the contact constraints at the end of each time step if and only if Equation 5.16 and 5.19 hold together.

5.4 Contact Constraints for Compliant Bodies

In this section we describe the 3D linear viscoelastic model of contact [61] and modify our contact constraints to include the deflections at the contact. We incorporate this model in our time-stepping scheme and present the mixed NCP problem that we are solving at each time step. We extend the Kelvin-Voigt model with the physically motivated observation that the deformations in the normal direction are bounded by some maximum value. For example, a human finger has a thin compliant layer of muscle and tissue surrounding the rigid core (bone). The application of a force on the finger results in a deformation of the thin compliant layer until the rigid core is reached, at which point the non-penetration response is rigid. Therefore, our model allows for a maximum possible deflection, beyond which the contact behaves as a rigid body contact. The linear model can be replaced by a nonlinear model but this comes at the cost of more unknown modeling parameters to be determined experimentally. For simplicity of exposition, we consider only one of the objects to be flexible at each contact. The general formulation where both the bodies are flexible will contain the additional constraint that the contact forces acting on both the bodies have to be equal. For each contact $i$, the normal impact force $\lambda_{in}$ is the sum of two components:

$$\lambda_{in} = \lambda_{inr} + \lambda_{ins}$$  \hspace{1cm} (5.20)

where $\lambda_{ins}$ is the component of the force that is obtained from the deformation of the spring and $\lambda_{inr}$ is the component from impact with the rigid core. The tangential force at each contact, $\lambda_{tf} = [\lambda_{tt}, \lambda_{to}]$ is also given by a linear spring-damper model. However, we do not have a bound on the maximum displacement in the tangential direction. Concatenating all the individual force components into vectors allows us to write for each contact (we drop subscript $i$ for legibility), $\lambda = K\delta + C\dot{\delta}$, where $\lambda = [\lambda_{ns}, \lambda_{t}, \lambda_{o}]$ and $\delta = [\delta_n, \delta_t, \delta_o]$ are $3 \times 1$ column vectors with $\delta_n, \delta_t, \delta_o$ being the normal and tangential
deflections respectively. The matrices $K$, $C$ are stiffness and damping matrices given by

$$K = \begin{bmatrix}
    K_{nn} & K_{nt} & K_{no} \\
    K_{tn} & K_{tt} & K_{to} \\
    K_{on} & K_{ot} & K_{oo}
\end{bmatrix} \quad \quad \quad C = \begin{bmatrix}
    C_{nn} & C_{nt} & C_{no} \\
    C_{tn} & C_{tt} & C_{to} \\
    C_{on} & C_{ot} & C_{oo}
\end{bmatrix}$$

For systems with multiple contact, the contact forces $\lambda$, and body deformations $\delta$ become concatenations of $n_c$ subvectors, where $n_c$ is the number of contacts. The stiffness and damping matrices are block diagonal matrices of size $3n_c \times 3n_c$, where each diagonal block of size $3 \times 3$ represent one contact.

### 5.4.1 Objects described by a single convex function

When we consider contact compliance the contact constraints in Section 3 need to be modified to take into account the deflection $\delta$. We denote the maximum normal deflection by $\delta^o_n > 0$ and assume that it will be determined experimentally. Figure 5.2 shows two objects in contact with each other. The bold line shows the deformed shapes of the two objects. The point of contact is the point where the virtual objects shown by dotted lines touch. The deflections of the two objects along the normal at the contact point are $\delta_{n1}$ and $\delta_{n2}$ respectively. In the subsequent discussion, we will assume $\delta_{n1} = 0$ for simplicity and drop the subscript 2 from $\delta_{n2}$. Therefore, the constraints for the closest points are given by

$$a_1 - a_2 = -l_1 \nabla f(a_1)$$

$$\nabla f(a_1) = -l_2 \nabla g(a_2)$$

$$0 \leq l_1 \perp -f(a_1) \geq 0$$

$$0 \leq l_2 \perp -(g(a_2) + \bar{\delta}_n) \geq 0$$

where $\bar{\delta}_n$ is the algebraic distance. However, the normal contact force is given in terms of the Euclidean deflection. To obtain the Euclidean deflection from this algebraic deflection we note that the Euclidean deflection is the distance between the point $a_2$ and the point where the normal to $g(\xi_2) + \delta_{n2} = 0$ at $a_2$ intersects $g(\xi_2) = 0$. From the above argument
Figure 5.2: Schematic representation of the deflection at contact. The contact is where the dotted curves touch.

it can be seen that
\[ g(a_2 + \delta_n \frac{\nabla g(a_2)}{\|\nabla g(a_2)\|}) = 0 \]  
(5.22)

The complementarity conditions in Equation (5.1) thus becomes:
\[
\begin{align*}
0 &\leq \lambda_{ns} \perp \psi(a_1, a_2) + \delta_n \geq 0 \\
0 &\leq \lambda_{nr} \perp \delta_n^o - \delta_n \geq 0
\end{align*}
\]  
(5.23)

where \( \psi(a_1, a_2) = f(a_2) \) or \( g(a_1) \) for implicit surfaces. When the two bodies are not in contact the right hand side of both the complementarity constraints are positive and hence we do not have any contact force. The above system of equations are to be written for each of the contacting bodies. This formulation ensures that we satisfy the contact constraints at the end of the time step taking into consideration the possibility of the deflection of the body. It does not require the computation of penetration depth for obtaining the deflection as required in [103]. It ensures that we get a collision response in a fixed time-step scheme.

We can now formulate the mixed NCP for the geometrically-implicit lumped compliant contact time-stepper. The vector of unknowns \( z \) can be partitioned into \( z = [u, v] \)
where \( u = [\nu, a_1, a_2, \delta_n, \delta_n, \delta_o, p_t, p_o, p_c] \) and \( v = [l, p_{ns}, p_{nr}, \sigma] \). The equality constraints in the mixed NCP are:

\[
0 = -M \nu^{\ell + 1} + M \nu^{\ell} + W_{n}^{\ell + 1} p_{n}^{\ell + 1} + W_{t}^{\ell + 1} p_{t}^{\ell + 1} + W_{o}^{\ell + 1} p_{o}^{\ell + 1} \\
+ W_{r}^{\ell + 1} p_{r}^{\ell + 1} + p_{app} + p_{\nu}^{\ell + 1} \\
0 = p^{\ell + 1} - (hK \delta^{\ell + 1} + C(\delta^{\ell + 1} - \delta^{\ell})) \\
0 = (a_1^{\ell + 1} - a_2^{\ell + 1}) + l_1 \nabla f(a_1^{\ell + 1}) \\
0 = \nabla f(a_1^{\ell + 1}) + l_2 \nabla g(a_2^{\ell + 1}) \\
0 = g(a_2^{\ell + 1} + \delta_n^{\ell + 1} \nabla g(a_2^{\ell + 1}) \| \nabla g(a_2^{\ell + 1}) \|) \\
0 = E_{t}^{2} U p_{n}^{\ell + 1} \circ (W_{t}^{L})^{\ell + 1} \nu^{\ell + 1} + p_{t}^{\ell + 1} \circ \sigma^{\ell + 1} \\
0 = E_{o}^{2} U p_{n}^{\ell + 1} \circ (W_{o}^{L})^{\ell + 1} \nu^{\ell + 1} + p_{o}^{\ell + 1} \circ \sigma^{\ell + 1} \\
0 = E_{l}^{2} U p_{n}^{\ell + 1} \circ (W_{l}^{L})^{\ell + 1} \nu^{\ell + 1} + p_{r}^{\ell + 1} \circ \sigma^{\ell + 1}
\]

(5.24)

where \( p^{\ell + 1} = [p_{n}^{\ell + 1} p_{t}^{\ell + 1} p_{o}^{\ell + 1}]^T, p_{n}^{\ell + 1} = p_{ns}^{\ell + 1} + p_{nr}^{\ell + 1} \) and \( \delta^{\ell + 1} = [\delta_n^{\ell + 1} \delta_t^{\ell + 1} \delta_o^{\ell + 1}]^T \).

The complementarity constraints on \( v \) are:

\[
0 \leq \begin{bmatrix}
  l_1 \\
  l_2 \\
  p_{ns}^{\ell + 1} \\
  p_{nr}^{\ell + 1} \\
  \sigma^{\ell + 1} \\
\end{bmatrix} \perp \begin{bmatrix}
  -f(a_1^{\ell + 1}) \\
  -(g(a_2^{\ell + 1}) + \bar{\delta}_n^{\ell + 1}) \\
  \psi(a_1^{\ell + 1}, a_2^{\ell + 1}) + \bar{\delta}_n^{\ell + 1} \\
  \delta_n^{\ell + 1} - \delta_o^{\ell + 1} \\
  \zeta
\end{bmatrix} \geq 0
\]

(5.25)

where

\[
\zeta = U p_{n}^{\ell + 1} \circ U p_{n}^{\ell + 1} - (E_{t}^{L})^{-1} (p_{t}^{\ell + 1} \circ p_{t}^{\ell + 1}) - (E_{o}^{L})^{-1} (p_{o}^{\ell + 1} \circ p_{o}^{\ell + 1}) - (E_{l}^{L})^{-1} (p_{r}^{\ell + 1} \circ p_{r}^{\ell + 1})
\]

In the above formulation, we see \( u \in \mathbb{R}^{6n_b+13n_c}, v \in \mathbb{R}^{3n_c}, \) the vector function of equality constraints maps \([u, v]\) to \(\mathbb{R}^{6n_b+13n_c}\) and the vector function of complementarity constraints maps \([u, v]\) to \(\mathbb{R}^{3n_c}\) where \(n_b\) and \(n_c\) are the number of bodies and number of contacts respectively. If using convex bodies only, the number of contacts can be determined directly from the number of bodies, \(n_c = \sum_{i=1}^{n_b} i\).
5.4.2 Objects described by intersections of convex functions

Using the results obtained in Section 5.3.3 and the previous section, we can now formulate the contact constraints for quasi-rigid or locally compliant bodies where each body is represented as an intersection of convex implicit surfaces. In this case the complementarity conditions for nonpenetration can be written as either one of the following two sets of conditions:

\[
0 \leq \lambda_{in} \perp \max\{f_j(a_2) + \bar{\delta}_{jn}\} \geq 0 \quad j = 1, \ldots, m
\]
\[
0 \leq \lambda_{in} \perp \max\{g_j(a_1) + \bar{\delta}_{jn}\} \geq 0 \quad j = m + 1, \ldots, n
\]

Moreover, the closest points on the two objects are given by

\[
a_1 - a_2 = -(\nabla f_{k_1}(a_1) + \sum_{k \in \{I \setminus k_1\} \cap \{i\}} l_k \nabla f_k(a_1))
\]
\[
\nabla f_{k_1}(a_1) + \sum_{k \in \{I \setminus k_1\} \cap \{i\}} l_k \nabla f_k(a_1) = -\sum_{k \in \{I \setminus k_2\} \cap \{j\}} l_k \nabla g_k(a_2)
\]

\[
0 \leq l_i \perp -f_i(a_1) \geq 0
\]
\[
0 \leq l_j - (g_j(a_2) + \bar{\delta}_{jn}) = 0
\]

which is a modification of Equation 5.19 and the notation is identical to the discussion in Section 5.3.3. To determine the actual deflection from the algebraic distance, we need to assume that the normal at the point of contact is well defined. Assuming this to be true, we can then obtain the deflection using Equation 5.22 and the forces on the two bodies can be obtained using the appropriate constitutive laws.

5.5 Illustrative Examples for Rigid Bodies

In this section we present three examples to validate our technique against known analytical results and previous approaches. The first example is the same example of a disc rolling without slip on a plane that we studied in Section 2.3. The second example, taken from [108], consists of a sphere spinning on a horizontal surface that comes to rest due to torsional friction. The time taken by the sphere to come to a complete stop is known analytically and we show that our simulation results agree with the analytical predictions. The final example consists of a small ball moving in contact with two larger fixed balls.
We include it here to compare our solutions with those based on earlier approaches [108, 72]. All of our numerical results were obtained by PATH [38], a free solver that is one of the most robust complementarity problem solvers available.

5.5.1 Example 1: Disc on a Plane

In this example we revisit the unit disc example from Section 2.3. For illustrative purposes, we explain the formulation of the full dynamic model in detail. The normal axis of the contact frame \( \hat{n} = [0, 1]^T \) always points in the inertial \( y \)-axis direction and tangential axis \( \hat{t} = [1, 0]^T \) always coincides with the \( x \)-direction. The mass matrix, \( M \) is constant and the only force acting on the body is due to gravity. The equation of the disc is given by

\[
f_1(x, y) = (x - q_x)^2 + (y - q_y)^2 - 1,
\]

where \( q = (q_x, q_y) \) is the location of the center of the disc in the inertial frame. Let \( \mathbf{u} = [v_x, v_y, \omega_z] \) be the vector of linear and angular velocities and \( a_1 \) be the closest point on body 1 (the disc) to the plane (defined by \( y = 0 \)). Similarly, let \( a_2 \) be the closest point on the plane to body 1 (\( a_{2y} = 0 \) and can be removed from the system of unknowns). Thus we have \( M = \text{diag}(m, m, 0.5m) \), \( p_{\text{app}} = [0, -mgh, 0]^T \), with \( g = 9.81 \), the acceleration due to gravity, and

\[
\mathbf{r} = \begin{bmatrix}
(a_{1x}^{\ell+1} - q_x) \\
(a_{1y}^{\ell+1} - q_y)
\end{bmatrix}
\quad \mathbf{W}_n = \begin{bmatrix}
\hat{n} \\
\mathbf{r} \otimes \hat{n}
\end{bmatrix}
\quad \mathbf{W}_t = \begin{bmatrix}
\hat{t} \\
\mathbf{r} \otimes \hat{t}
\end{bmatrix}
\quad \nabla_{a_1} f_1(a_1^{\ell+1}) = \begin{bmatrix}
2(a_{1x}^{\ell+1} - q_x) \\
2(a_{1y}^{\ell+1} - q_y)
\end{bmatrix}
\]

where \( \mathbf{r} \) is the vector from the center of gravity of the disc to \( a_1 \) and \( \otimes \) connotes the 2D analog of the cross product (for two vectors \( \mathbf{x} = [x_1, x_2]^T, \mathbf{y} = [y_1, y_2]^T, \mathbf{x} \otimes \mathbf{y} = x_1y_2 - x_2y_1 \).

Assuming that \( y = 0 \) is the equation of the ground plane, there are 11 unknowns for this system: \( \mathbf{z} = [\mathbf{u}, a_1, a_{2x}, l_1, l_2, p_n, p_t, \sigma] \). The system of equations for the unit
The initial configuration of the disc is $\mathbf{q} = [0, 1, 0]$, initial velocity is $\mathbf{\nu} = [-3, 0, 3]$, mass is $m = 1$, and $\mu = 0.4$. Figure 2.3 shows the kinetic energy of the disc for our implicit representation along with the Stewart-Trinkle LCP implementation using various levels of discretization as it rolls along the horizontal surface. When using an implicit curve representation to model the disc and our formulation we get no energy loss (within the numerical tolerance of $10^{-6}$ used for our simulations) as seen by the horizontal line. When using the LCP formulation we have energy loss as discussed earlier.

### 5.5.2 Example 2: Sphere on a Plane

Here we consider a sphere spinning on a plane about the normal of the plane given by the equation $z = 0$. The equation of the sphere is given by $f_1(x, y, z) = (x - q_x)^2 + (y - q_y)^2 + (z - q_z)^2 - 1$ where $[q_x, q_y, q_z]$ is the position of the center of the sphere. The initial configuration of the sphere is $\mathbf{q}_{\text{init}} = [0, 0, 1, 0, 0, 0]^{T}$ where the first three elements are the position of the center of the sphere and the last 4 are the unit quaternion representing the orientation. The initial linear and angular velocities is given by $\mathbf{\nu}_{\text{init}} = [0, 0, 0, 0, 1.962]^{T}$. The friction parameters used were $e_t = 1$, $e_o = 1$, $e_r = 0.4$ and $\mu = 0.2$. A step size of $h = 0.07$ seconds was chosen. The normal and tangential vectors at the contact point are $\mathbf{\hat{n}} = [0, 0, 1]^{T}$, $\mathbf{\hat{t}} = [1, 0, 0]^{T}$, $\mathbf{\hat{o}} = [0, 1, 0]^{T}$. The sphere is assumed to be of unit mass.
Figure 5.3: Linear and angular velocities for Example 2. All velocities except $\omega_z$ are zero throughout the simulation.

The matrices in the Newton-Euler equations are:

$$M = \begin{bmatrix} I_{d_{3 \times 3}} & 0 \\ 0 & \frac{2}{3} I_{d_{3 \times 3}} \end{bmatrix} \quad \nabla f_1(a_{1}^{\ell+1}) = \begin{bmatrix} 2(a_{1x} - q_{1x}) \\ 2(a_{1y} - q_{1y}) \\ 2(a_{1z} - q_{1z}) \end{bmatrix} \quad p_{opp} = \begin{bmatrix} 0_{2 \times 1} \\ -9.81 \times h \\ 0_{3 \times 1} \end{bmatrix}$$

$$W_n = \begin{bmatrix} \hat{n} \\ r^{\ell+1} \times \hat{n} \end{bmatrix} \quad W_t = \begin{bmatrix} \hat{t} \\ r^{\ell+1} \times \hat{t} \end{bmatrix} \quad W_o = \begin{bmatrix} \hat{o} \\ r^{\ell+1} \times \hat{o} \end{bmatrix} \quad W_r = \begin{bmatrix} 0_{3 \times 1} \\ \hat{n} \end{bmatrix}$$

where $I_{d_{3 \times 3}}$ is the $3 \times 3$ identity matrix and $r^{\ell+1}$ is the vector from the sphere’s center to the computed contact location ($a_{1}^{\ell+1}$). From the initial conditions given, the sphere should rotate in place with a constant deceleration due to the torsional friction. Figure 5.3 shows a plot of the velocities for the sphere given by the time-stepping formulation. The analytical solution for this problem predicts that all velocities except $\omega_z$ should be zero, and $\omega_z$ should be decreasing linearly to 0 with a slope of -1.962, reaching 0 at exactly
Figure 5.4: Forces for Example 2. The tangential forces are both 0 for the entire simulation, and the torsional force transitions to zero when the sphere switches from a sliding contact to sticking.

$t = 1$ seconds. The plot shows that we agree with the analytical solution except that we reach zero velocity at $t = 1.05$, since we are using a fixed time step and the time at which spinning stops is in the middle of the time step. The friction forces (Figure 5.4) also follow the analytical solution. The tangential component of friction force is 0. The tangential moment does not drop to 0 at 1.05 s, since we are using an impulse-based time-stepping formulation with a fixed time step and there is a torsional moment between 0.98 to 1 second which contributes to the impulse. Our results match those of the Tzitzouris-Pang and Stewart methods presented in [108].

### 5.5.3 Example 3: Sphere on Two Spheres

This example consists of a small sphere moving in contact with two larger fixed spheres. This example is chosen to compare the results of our geometrically-implicit method to those presented in [108] and [72] ([108] was able to solve this problem with implicit geometric information because a closed form distance function is available be-
Figure 5.5: A small sphere in contact with two large spheres.

Figure 5.5 shows a small unit sphere in simultaneous contact with two larger fixed spheres. The sphere of radius 10 units is located at \((0, 0, 0)\) in the inertial frame and the sphere of radius 9 units is located at \((0, 11.4, 0)\). There is also a constant force of \(\lambda_{app} = [1.0, 2.6, -9.81, 0, 0, 0]^T\) applied to the small sphere. With this force, the sphere initially has one of its contacts rolling while the other contact is simultaneously sliding, the rolling contact transitions to sliding, and both contacts eventually separate. It is important to emphasize that all these transitions are captured using a fixed time step implementation.

The initial configuration and velocity of the small moving sphere is \(q = [0, 6.62105263157895, 8.78417110772903, 1, 0, 0, 0]^T, \nu = 0_{6 \times 1}\). The friction parameters are: \(e_t = 1, e_o = 1, e_r = 0.3\), and \(\mu = 0.2\). There are 32 variables in our NCP formulation (6 velocity variables and 13 variables for each contact: 6 contact point coordinates for the two bodies, 2 lagrange multipliers, 4 contact impulses and \(\sigma\) for the contact). We used a step size \(h = 0.01\) (Tzitzouris-Pang use \(h = 0.1\)).
The generalized velocity of the sphere is shown in Figure 5.6. The smooth velocity profile agrees well with the nonlinear Tzitzouris-Pang formulation [108]. The Liu-Wang formulation [72] experienced non-smooth velocity jumps when the small sphere separated from the larger fixed spheres, which they attributed to an explicit time-stepping scheme. In the LCP Stewart-Trinkle implementation, the velocity profiles were very non-smooth. These results further confirm our belief that both linearization and explicit time-stepping lead to inaccuracies.

The fidelity of our method is further emphasized by Figures 5.7 and 5.8 that show the forces and sliding speed magnitudes at the two contacts. Contact 1 starts as a sliding contact and we see the sliding speed increases as the normal force decreases. Also, the magnitude of the friction force is equal to $\mu_1 n$, consistent with our friction law for a sliding contact. At approximately 3.2 seconds, the small sphere separates from the large sphere at this contact, and all forces acting at contact 1 and the sliding speed drop to zero. Contact 2 on the other hand starts out as a rolling contact until approximately $t = 3$ seconds when it transitions to sliding. During the rolling phase the frictional magnitude is bounded by $\mu_2 n$, as required by the friction law, and the sliding speed is 0. At the
Figure 5.7: Force and sliding speed at contact 1. Contact 1 is always sliding until separation, hence the $\mu$ normal force curve and friction magnitude curve overlap for the duration. The value of $\mu = 0.2$

transition to sliding, the magnitude of the friction force becomes equal to $\mu \lambda n$ and the sliding speed begins to increase. Finally, at approximately $t = 3.6$ seconds, the contact breaks and all forces at this contact and the sliding speed drop to zero.

### 5.6 Illustrative Examples for Compliant Bodies

In this section, we present two examples to validate our method for quasi-rigid bodies. For the quasi-rigid bodies, we use our compliance model where we assume that the body has a compliant outer layer surrounding a rigid inner core. The first example consists of a unit disc falling on a half-plane and the contact between them is assumed to be frictionless. This example gives a simple validation of our method. The second example consists of a ellipsoid falling and moving on a compliant half-plane.
Figure 5.8: Force and sliding speed at contact 2. The value of $\mu = 0.2$

![Image](image.png)

**Figure 5.9: Unit disc falling onto a frictionless compliant surface**

5.6.1 Example 1: Disc falling on a compliant half-plane

In this example, we simulate a rigid unit disc falling onto a compliant horizontal half-plane. The contact is modeled as a single frictionless contact with no damping. Depending on the value of maximum deflection, the disc may or may not make contact with the rigid core of the half-plane. Figure 5.9 illustrates the problem.

There are 12 unknowns in this system, with 4 complementarity constraints: $z = [u, v] = [u_x, u_y, \omega, a_{1x}, a_{1y}, a_{2x}, a_{2y}, l_1, l_2, \delta_n, p_{ns}, p_{nr}]$. The equations of motion for this
The unit disc’s initial position was \( q = [0, 1.5, 0] \) with zero initial velocity \( \nu = [0, 0, 0] \). The only force acting on the disc was gravity. The mass of the disc was 1 kg and the moment of inertia about the center of mass was 0.5 kg·m². We used a step size \( h = 10^{-4} \) s. The spring stiffness we used was \( k = 1000 \) kg/s². The maximum penetration depth was altered for two experiments such that for the first experiment impact with the rigid core occurs, and for the second experiment impact with the rigid core does not occur. For experiment one, \( \delta_n^0 = 0.05 \) m and for experiment two \( \delta_n^0 = 1 \) m.

Figures 5.10, 5.11, and 5.6.1 illustrate the results of the first experiment in which the maximum spring deflection was not large enough to prevent impact with the rigid core. There is a large non-penetration impulse (Fig. 5.10(a)) generated at approximately 0.34 seconds corresponding to when the spring reached maximum deflection and impact with the rigid core occurs. As expected with a rigid impact, we also see an instantaneous change in velocity (Fig. 5.11(b)) to zero and loss of energy (Fig. 5.6.1). Subsequent to the impact, the motion of the disc (Fig. 5.11(a)) becomes oscillatory as it bounces on the undamped spring (Fig. 5.10(b)) and the velocity is smooth. As seen in Figure 5.6.1 the total energy is preserved after impact within a tolerance of \( 10^{-5} \) J, which is acceptable using a time step of \( 10^{-4} \) s and an Euler approximation in the time-stepping formulation.

For the second round of experiments, the maximum spring deflection was set large enough that impact with the rigid core never occurs. We see the oscillatory behavior of the
position over the lifetime of the simulation (Fig. 5.14(a)) as expected with an undamped spring. As guaranteed by our model, no component of the normal force comes from impact with the rigid core; the spring contributes solely to the normal force (Fig. 5.13(b)). Additionally, without any impacts the plot of velocity is smooth with changes occurring only from the force of gravity and the spring force (Fig. 5.14(b)). Since there is no impact nor damping of the spring, we expect there to be no loss of energy in the system. Figure 5.13(a) confirms this, where again the energy is conserved within a numerical tolerance of $10^{-5}$J.

### 5.6.2 Example 2: 3D Ellipsoid on a Compliant Plane with Frictional Contact

In this example we drop an elongated ellipsoid with an initial angular velocity onto a half-plane. There is no closed form distance function between an ellipsoid and half-plane, and the closest points between the two bodies are found explicitly by our formulation. The minor axes of the ellipsoid are 0.01m and the major axis is 0.5m resulting in the following implicit function describing the surface:

$$f(x, y, z) = \frac{(x - q_x)^2}{0.25^2} + \frac{(y - q_y)^2}{0.005^2} + \frac{(z - q_z)^2}{0.005^2} - 1$$

where $[q_x, q_y, q_z]$ is the position of the center of gravity of the ellipsoid in the fixed spatial frame. The mass of the ellipsoid is 1 kg. It has an initial linear velocity of zero and angular velocity of $[0, 0, 5]^T$rad/s. For all experiments the only applied force was gravity and we used a step size of $10^{-4}$ seconds.

Unlike the previous 2D example, the contact in this example is frictional and includes damping of the spring forces. The stiffness $K$ and damping $C$ matrices we used are:

$$K = \begin{bmatrix} 36000 & 0 & 0 \\ 0 & 8000 & 0 \\ 0 & 0 & 8000 \end{bmatrix} \quad C = 2\sqrt{K}$$

The friction parameters are: $e_t = 1$, $e_o = 1$, $e_r = 0.04$ and $\mu = 0.4$. The maximum depth was set to $\delta_n^0 = 0.005$m.

We varied the initial orientation of the ellipsoid for two different experiments. In
the first experiment we set the initial orientation of the body frame to coincide with the fixed spatial frame \( \mathbf{q} = [0, 0, 0.15, 1, 0, 0, 0] \), where the first three elements correspond to the ellipsoid’s center of gravity and the last four are the unit quaternion representing the orientation. In the second experiment, the initial orientation of the body frame was \( \mathbf{q} = [0, 0, 0.15, 0.9962, 0, 0.0872, 0] \)

Figures 5.15 and 5.16 show the results for the first experiment. We would expect the ellipsoid to have a constant angular velocity while falling, hit the half plane, and stop spinning due to the torsional friction. We see this exact behavior as shown in Figure 5.16. There is a large non-penetration impulse occurring just before 0.2 seconds, corresponding to the impact of the ellipsoid on the half-plane. This impact removes most of the energy because of the non-penetration and damping impulses. Corresponding to the rigid impact, there is an instantaneous drop in the angular velocity. The remaining velocity and kinetic energy are slowly dissipated by the torsional friction, and at approximately 0.4 seconds the ellipsoid transitions from sliding to sticking. Unlike the undamped example with the oscillatory behavior of the position, in this example the damping causes the position of the ellipsoid to settle to a resting height on the surface. After settling there is a constant normal spring force equal to the weight of the ellipsoid.

In the second experiment, the initial orientation of the ellipsoid causes moments about the \( x \) and \( y \) axis at the time of impact such that angular velocities in the \( x \) and \( y \) directions appear after impact. This behavior is seen in our simulation and the results are presented in Figures 5.17, 5.18, and 5.6.2. There is a large non-penetration impulse generated from hitting the rigid core. However, in this example the impact causes sliding to occur until approximately 0.5 seconds when the ellipsoid transitions to rolling. Since we are using an exact representation of the surface geometry, the rolling behavior continues ad-infinitum as expected. The energy plots show that energy is conserved during the whole simulation up to a numerical tolerance.

5.7 Conclusion

We presented the first geometrically implicit time-stepping scheme for objects described as intersections of convex inequalities. This approach overcomes stability and accuracy problems associated with polygonal approximations of smooth objects and ap-
proximation of the distance function for two objects in intermittent contact. We developed a new formulation for the contact constraints in the work space which enabled us to formulate a geometrically implicit time-stepping scheme as an NCP. We demonstrated through example simulations the fidelity of this approach to analytical solutions and previously described simulation results.

We see several directions for future work. Addressing the question of existence and uniqueness of solutions of the NCP we formulate is an important future direction. Precisely quantifying the trade offs between the computation speed and physical accuracy of simulations for different object representations (e.g., polyhedral, implicit, spline), friction approximations, and the choice between geometrically explicit or implicit methods is another area of future work. Although we have restricted our discussion to convex objects, we believe that this framework can be extended to non-convex objects described as unions of convex objects as well as parametric surfaces. This is another direction for the future.
Figure 5.10: Non-penetration force and spring force for a unit disc falling on a half-plane making contact with the rigid core.
Figure 5.11: Configuration and velocities for a unit disc falling on a half-plane making contact with the rigid core.
Figure 5.12: Energy plot for a unit disc falling on a half-plane making contact with the rigid core.
Figure 5.13: Energy and force plots for a unit disc falling on a half-plane without making contact with the rigid core.
Figure 5.14: Configuration and velocity plots for a unit disc falling on a half-plane without making contact with the rigid core.
Figure 5.15: Energy and force plots for an ellipsoid dropped onto a half-plane. The initial orientation of the ellipsoid is such that its major and minor axis are parallel to the inertial frame. Y-axis is set to $[-25, 100]$ in force plot.
Figure 5.16: Configuration and velocity plots for an ellipsoid dropped onto a half-plane. The initial orientation of the ellipsoid is such that its major and minor axis are parallel to the inertial frame.
Figure 5.17: Force and Energy plots for an ellipsoid dropped onto a half-plane. Y-axis is set to $[-100, 100]$ in force plot.
Figure 5.18: Configuration and Velocity plots for an ellipsoid dropped onto a half-plane.
Figure 5.19: Deformation of the compliant surface for an ellipsoid dropped onto a half-plane.
CHAPTER 6
Coverage of a Planar Point Set with Multiple Robots subject to Geometric Constrains

6.1 Introduction

Robotic point set coverage tasks occur in a variety of application domains like electronic manufacturing (laser drilling [17], inspection [16], circuit board testing [121, 57]), automobile spot welding [96], and data collection in sensor networks [112]. The goal of using multiple robots in point set coverage tasks is to reduce the overall task completion time by parallelizing the operations at the points. The path planning problem in such multi-robot point set coverage tasks can be stated as follows: Given a point set, \( S = \{p_i\}, \ i = 1, \ldots, N, \) and \( K \) robots, find an assignment of the points to individual robots and determine the order in which the robots must visit the points so that the overall task completion time is minimized. In this chapter we look at such path planning problems for multiple robot point set coverage where the robots have to (a) spend some time at each point to complete a task (we call this time processing time of each point) and (b) satisfy given geometric constraints (like collision avoidance) while covering the point set. Our work is motivated by a system (see Figure 6.1) used by a microelectronics manufacturer for laser drilling. Here we need to process (drill) a set of points with identical processing times by a system of \( K (= 2) \) robots. The architecture of the machine imposes the following geometric constraints: (a) at any instant of time, each robot can process exactly one point within a square region in the plane (called processing footprint) although there may be several points within the region, (b) the robots are constrained to move along a line while avoiding collisions, and (c) the points lie on a base plate that can translate along the \( y \)-axis.

In the absence of the geometric constraints and assuming the processing times to be zero, the path planning problem for multi-robot point set coverage tasks can be treated as a \( K \)-Traveling Salesman Problem (\( K \)-TSP). However the path planning problem with inter-robot geometric constraints hasn’t been treated extensively in the literature. In this chapter, we provide solution algorithms to the path planning problem for point set cover-
age with multiple robots, where the robots are subject to geometric constraints, and the processing time of each point is identical. We divide the problem into that of finding the assignment of points to each robot, and the order in which they will be processed while satisfying the geometric constraints as follows:

**Splitting Problem:** Let $P$ be a set of subsets of $S$ of size less than or equal to $K$ such that each point in $S$ can occur in exactly one element of $P$\(^6\). The splitting problem is to find a set $P$ of minimum cardinality that respects the geometric constraints and each point in $S$ occurs in exactly one element of $P$. Intuitively, such a set allows the maximum possible parallelization of the processing operation.

**Ordering Problem:** Given a set of $K$-tuples, $P$, find a processing order of the points by the robots such that the cost of visiting all the points is minimized.

The splitting problem can be reduced to a clique partitioning problem [41] on a graph for arbitrary $K$ and the ordering problem can be reduced to a traveling salesman problem (TSP). Both of the above problems are NP-hard. However, for two robots (i.e., $K = 2$) we show that the splitting problem is equivalent to a maximum cardinality matching (MCM) problem on a suitably constructed graph and can thus be solved optimally in polynomial time. The maximum cardinality matching algorithm takes $O(N^3)$ time in general ([67, 82]) and is not suitable for large data sets (in our application, approximately $10^5$ points). Therefore, we provide a greedy algorithm that exploits the geometric nature of the constraints and takes $O(N \log N)$ running time. We provide results comparing the greedy algorithm with the matching algorithm for small datasets. Our computational experiments show that for typical industrial datasets the greedy algorithms give solutions that are very close to the optimal solution.

We model the ordering problem as a multi-dimensional TSP ([57, 121]) in the set of point pairs (pair space) obtained from the splitting problem. The solution of this TSP gives a tour of the individual robots. We identify necessary conditions in the tour of the individual robots that improves the tour in the pair space. We provide results showing an improvement of 5% to 8% in the TSP tour using the tour improvement heuristic. To include points in the tour that were not paired in the splitting stage we give a cheapest insertion heuristic. We also give computational results showing the improvement of

\(^6\)We will refer to each element of $P$ as a $K$-tuple with the understanding that if there are less than $K$ points present in an element we add virtual points to make its cardinality equal to $K$. 
Figure 6.1: Schematic sketch of a 2-robot system used to process points in the plane. The heads can translate along the \( x \)-axis and the base plate translates along the \( y \)-axis. The square region of length \( 2\Delta \) is the processing footprint for each robot.

performance obtained by using a two robot system and the path planning algorithms described above over a single robot system. Finally, we extend our algorithms for the two robot system to a \( K \) robot system and provide some computational results for a four robot system.

This chapter is organized as follows: In Section 6.2, we briefly summarize the relevant literature. In Section 6.3 we give the general mixed integer optimal control formulation for the \( K \)-robot system and justify division of the problem into two subproblems. In Section 6.4, we provide solution algorithms and results for the two robot point splitting problem. In Section 6.5, we formulate the ordering problem as a TSP in the pair space and provide local search based heuristics to improve the tour. In Section 6.6, we extend the algorithms for two robot system to a \( K \)-robot system. Finally, in Section 6.7, we present our conclusions and outline future work. A preliminary version of this work appeared in [17].

### 6.2 Related Literature

The general motion planning problem for the minimum-time multiple robot point set coverage problem can be stated as follows:

**Input:** A set of points \( S = \{p_i\} \) with processing times \( \tau_i, \ i = 1, 2, \ldots N \), that must be
processed by $K$ robots where each robot has a limited footprint and the robots must satisfy given geometric, kinematic, and dynamic constraints.

**Output:** A trajectory for each robot satisfying the constraints such that the total time (process time plus travel time) required to cover the point set is minimized.

This is a hybrid discrete-continuous optimization problem because we have to simultaneously optimize over (a) the feasible discrete choices involved in the assignment of points to the robots and the order in which the points are visited by the robots, (b) the feasible continuous choices involved in specifying the position and velocity of the robots as a function of time. The problem is hard to solve even for $K = 1$, even without the geometric constraints.

There are two distinct approaches to solving hybrid discrete-continuous optimization problems like the one above: (1) Form a mixed integer nonlinear optimal control problem [114] or (2) Use a two stage approach: (a) Solve the discrete optimization problem of finding the path and (b) Solve the continuous optimization problem of converting the path into a trajectory. The first approach is very general although the resultant problem is very hard to solve in practice. von Stryk and Glocker [114] used this approach to find the trajectories of two cooperating robots (cars), with given kinematic motion model, visiting a set of points. They used a two level iterative scheme to find the optimal trajectories. The outer level iteration used a branch and bound framework to search the space of discrete variables (in this case, the variables corresponding to assignment and order of the points). The inner level iteration solved a nonlinear optimal control problem over the space of continuous variables (in this case, the position and velocity of the robots). This approach can require the solution of an exponential (in the number of points to be visited) number of inner level nonlinear optimal control problems, each of which is nontrivial to solve. Hence, this approach is limited to a small number of points.

The literature for the second approach usually focuses on either the discrete optimization problem of covering a point set or the continuous optimization problem of trajectory generation. The problem of covering a point set by a single robot with collision avoidance constraints has been studied for industrial robots [96, 119, 105, 8]. Saha et al. [96], and Wurll and Henrich [119] address motion planning of a fixed base manipulator to process a set of points avoiding static obstacles in the workspace. The points
are assumed to be partitioned into groups and the motion is assumed to be point-to-point. In [105], Spitz and Requicha consider the point set processing problem for a coordinate measuring machine. Since there is only one robot, the processing time is constant and the main focus of these papers is to find a minimum cost collision free path covering all the points. The collision avoidance problem is nontrivial in these cases and all of the above papers use a discrete search of the configuration space ([96] and [105] use different versions of probabilistic roadmaps whereas [119] uses $A^*$ search) for computing a collision free path. On the other hand, we have multiple robots and algebraic equations that give collision avoidance constraints. Thus we focus on assigning the points to the robots (to reduce processing time) as well as obtaining an order of processing them (to reduce traveling time) while avoiding collisions among the robots.

Dubowsky and Blubaugh (see [35], Section IV) considered the problem of multiple manipulators processing a set of points. However, they assumed that the manipulators will never be in collision with each other and formulated the problem as a $K$-TSP. Their solution approach was to find a tour for one manipulator and then divide it into $K$ tours for $K$ manipulators such that the maximum of the $K$ tour costs is minimized. Here, we need to satisfy collision avoidance constraints, and such an approach is not suitable.

### 6.3 Problem Formulation

The motion planning problem for minimum time multiple robot point set coverage can be formulated as a mixed integer optimal control problem (see Appendix A). The solution of this formulation gives the trajectories of the $K$ robots such that the overall time taken in covering the point set is minimized. However, as observed in [114], it is difficult to solve problems of this type directly and this is especially true for the large datasets that we have to deal with. Therefore, we follow the usual approach in robotic motion planning literature and divide the motion planning problem to a path planning problem and trajectory optimization problem. In this thesis we are concerned with the formulation and solution of the path planning problem only. There are two main questions that arise in the formulation of the path planning problem:

1. Do we need to incorporate the inter-robot geometric constraints in the path planning problem?
2. What is a measure of the travel cost in presence of the inter-robot geometric constraints?

Before we proceed to answer the above two questions we first look at the minimum cost path planning problem for $K$ robots, without geometric constraints, where the robots have some processing time at each point. If the processing times are all zero then the problem is equivalent to a $K$-TSP problem defined below:

*Given a weighted complete graph, $G = (V, E)$, where the set of vertices $V$ consists of the set of all the points and the weight on an edge is the travel cost between the two points, find $K$ subtours on this graph such that the cost of the most expensive subtour (i.e., sum of weights of the edges in the subtour) among the $K$ subtours is minimized.*

If the processing times are non-zero the problem can still be set up as a $K$-TSP with suitable weights. Let $S = \{p_i\}, i = 1, \ldots, N$, be the point set that is to be processed in minimum time by $K$ robots where each robot has to spend time $\tau_i$ at point $p_i$. Let $d_{ij}$ be the travel cost on the edge $(i,j)$, $\tau_i$, $\tau_j$, be the processing times of points $p_i$, $p_j$ respectively and $w > 0$ be an arbitrary positive number.$^7$ Define the new cost on the edges as

$$\tilde{d}_{ij} = d_{ij} + \frac{w(t_i + t_j)}{2}$$

(6.1)

The solution of a $K$-TSP with the edge weights defined above gives the tours for each robot such that the maximum sum of processing cost and travel cost is minimized. Moreover, if $d_{ij}$ satisfies the triangle inequality then $\tilde{d}_{ij}$ also satisfies the triangle inequality. Consider three vertices $i, j, k$. From Equation 6.1 we have:

$$\tilde{d}_{ij} + \tilde{d}_{jk} = d_{ij} + \frac{w(t_i + t_j)}{2} + d_{jk} + \frac{w(t_j + t_k)}{2}$$

$$= d_{ij} + d_{jk} + w\tau_j + \frac{w(t_i + t_k)}{2}$$

(6.2)

$$\geq d_{ik} + \frac{w(t_i + t_k)}{2} \geq \tilde{d}_{ik}$$

Thus the new cost metric defined by Equation 6.1 satisfies the triangle inequality. Therefore, we can use the algorithm given in [39] to obtain a constant factor approximation

---

$^7$ $w$ is a problem domain dependent scale factor that accounts for different units that the travel cost and processing cost may have, e.g., travel cost may have units of length whereas processing cost may have units of time.
algorithm for the problem. The constant factor is $5/2$ if we use Christofides algorithm to solve the 1-TSP problem.

In the presence of inter-robot geometric constraints the solution obtained above by ignoring the constraints may be arbitrarily bad, i.e., the $K$-TSP solution may result in no parallelization of the operations. We illustrate this with a simple example. Figure 6.2 shows a simple input point distribution where no two points lie simultaneously in the processing zone of a single robot. Assume the geometric constraints on the robot is the same as that in Figure 6.1. The bold line shows the optimal paths of the two robots obtained ignoring the geometric constraints. However, because of the constraints, no two points assigned to the two robots in this solution can be processed simultaneously and the time taken is same as that would be achieved by a single robot. Thus, solving a standard $K$-TSP ignoring the geometric constraints can give an arbitrarily bad solution. Thus the answer to our first question is that we indeed do need to take the inter-robot geometric constraints into consideration at the path planning stage. To answer the second question, we first note that in the presence of geometric constraints the maximum cost subtour among the $K$ subtours is not the right measure of the overall cost. This is because the inter-robot constraints imply that the robots may not be able to simultaneously traverse parts of their own tours. The total travel cost incurred for task completion in this case is the sum of the simultaneous travel costs of the robot and the individual travel cost of the robots (i.e., costs of parts of the tour where they cannot move simultaneously).

![Figure 6.2: An example input distribution of points where the bold lines show an optimal 2-TSP tour on this set obtained ignoring the geometric constraints. Clearly, the two robots for the system in Figure 6.1 cannot process any pair of points simultaneously while satisfying the geometric constraints.](image-url)
Although we need to take care of the inter-robot collision avoidance constraints, it is not clear how to do that at the path planning stage because these constraints should be satisfied at all times whereas there is no time information in the path planning problem. Therefore, we pose the path planning problem in the space of $K$-tuples of points and define the feasible path for this problem in the $K$-tuple space as follows: A feasible path is defined as an ordered set of tuples of points of size less than or equal to $K$ such that the robots satisfy the inter-robot geometric constraints if they are present at points in the same tuple. The path in the $K$-tuple space induces a path in the Euclidean space for each robot. This definition ensures that when the robots move along their respective paths, they would satisfy the geometric constraints if they are at the points belonging to the same tuple at the same time. Thus, once we have an ordered set of tuples, we can ensure that the geometric constraints are satisfied when moving along the path by writing the constraints as simple velocity constraints of the robots in the point-to-point motion trajectory optimization problem. To solve the above path planning problem, we take the following two step approach:

1. Divide the set of points into $K$-tuples of points that satisfy the geometric constraints and assign the points in each tuple to a robot.

2. Find the order in which the tuples should be processed by the robots (this produces a tour for each robot on the set of points assigned to it).

We can also view the two step approach as follows: we first find a collection of points that can be processed simultaneously, thus obtaining an optimal solution for the processing time and then we find a travel path for the robots that minimizes the travel cost without changing the processing cost. Therefore, in cases where the processing time dominates the travel time this approach will give a good solution to the overall problem.

### 6.4 Splitting Problem

In this section we look at the splitting problem for the two robot system shown in Figure 6.1. The splitting problem consists of assigning the set of points to the robots so as to maximize the number of points that can be processed together. We call a pair of points a compatible pair (of points) if they can be processed together while respecting the
geometric constraints. Any two compatible pairs are called a *disjoint compatible pair* if the points belonging to the two pairs are distinct. Thus, the splitting problem is equivalent to minimizing the total number of disjoint compatible pairs and singletons while assigning them to the robots. A solution to this problem ensures the maximum parallelization of the processing operation. The formal statement for this problem is given below:

**Problem Statement:** Let $S = \{p_i\} = \{(x_i, y_i)\}_{i=1}^N$, be a set of points in $\mathbb{R}^2$. Let $P$ be a set of ordered subsets of $S$ of size less than or equal to 2 that partitions $S$, i.e.,

$P = \{(p_i, p_j)\} \bigcup \{(p_k, *)\} \bigcup \{(*, p_l)\}$, $i, j, k, l \in \{1, 2, \ldots, N\}, i \neq j \neq k \neq l$, where $*$ denotes a *virtual point* and any pair $(p_i, p_j) \in P$ respects the following constraints

$$\begin{align*}
|x_i - x_j| &\geq s_{\text{min}} - 2\Delta \\
|y_i - y_j| &\leq 2\Delta
\end{align*}$$

(6.3)

where $s_{\text{min}}$ is the minimum distance between the two robots. Find such a $P$ of minimum cardinality.

In the above statement the ordered pair $(p_i, p_j)$ denotes that $p_i$ is assigned to robot 1 and $p_j$ is assigned to robot 2. Moreover $(p_k, *)$ denotes that $p_k$ is a singleton assigned to robot 1, while $(*, p_l)$ denotes that $p_l$ is a singleton assigned to robot 2. The constraint between the $x$-coordinates of the points in Equation 6.3 ensures collision avoidance between the robots. The constraint on the $y$-coordinates indicate that the robots are constrained to move along the $x$-axis but have a square footprint for processing.

### 6.4.1 Optimal Algorithm

The problem above can be solved optimally by converting it to a maximum cardinality matching problem on a graph.

**Definition** Let $G = (V, E)$ be a weighted graph where $V$ is the set of vertices and $E$ is the set of edges. A set $M \subseteq E$ is called a *matching* if no two edges in $M$ have a vertex in common. $M$ is called a *maximal matching* if there is no matching $M'$ such that $M \subset M'$. $M$ is called a *maximum cardinality matching* (MCM) if it is a maximal matching of maximum cardinality. $M$ is called a *maximum weighted matching* (MWM) if the sum of weights of the edges in $M$ is maximum among all matchings.
**Definition** A vertex in $V$ is called a matched vertex if there is one edge incident upon it in $M$, otherwise it is called an unmatched or exposed vertex.

From the given set of $N$ input points, $S$, we construct a graph $G = (V, E)$, where $V$ is the set of all points and $E$ is the set of all edges with an edge existing between two points iff they form a compatible pair, i.e., they satisfy Equation 6.3. We call this graph the compatibility graph of the point set. A maximum cardinality matching on the compatibility graph gives the maximum number of disjoint compatible pairs, i.e., the maximum number of points that can be processed together. The unmatched vertices form the singletons that are to be processed individually. The set $P$ consisting of the matched pairs and unmatched vertices will be of minimum cardinality since the number of singletons are minimum. After we obtain the matching solution, we can use the geometry of our problem to assign the points to the robots. In our problem, robot 1 is always constrained to be on the left of robot 2. Therefore, we order the pairs in the matching so that the point with lower $x$-coordinate is on the left and thus gets assigned to robot 1. For the singletons, we assign points on the left of the median of the point distribution to robot 1 and points on the right to robot 2.

The MCM problem on a graph is a well studied combinatorial optimization problem and can be solved in $O(N^3)$ time (Edmonds [36]). However, slightly faster algorithms do exist (e.g., Micali and Vazirani’s $O(\sqrt{|V||E|})$ algorithm [74]). In our application, $N$ can be of the order of $10^5$ and the matching algorithm is not practical for such large values of $N$. Hence we provide a greedy algorithm that gives a suboptimal solution and runs in $O(N \log N)$ time. We note that although there are greedy algorithms in the matching literature that have linear running time in the number of edges (see [113], and references therein), such algorithms assume the input to be available in the form of a graph. In our problem, the input is a set of points, $S$, and the parameters $\Delta$ and $s_{\text{min}}$ specifying the geometric constraints. Hence, we need to construct the input graph from this information and this may take $O(N^2)$ time in the worst case.

### 6.4.2 Greedy Algorithm

Given the set of input points, $S$, and the parameters, $\Delta$ and $s_{\text{min}}$, we use the geometric structure of our constraints and the distribution of the input points to design a
greedy algorithm. We first divide the points along the $y$-axis into bands of width $2\Delta$ and then divide the points within each band into two almost equal halves using the median of the $x$-coordinates of the points in the band. Then starting from the left most point of the left half, we pair each point in the left half with a compatible point in the right half with minimum $x$-coordinate, breaking ties by choosing points with minimum $y$-coordinate. This is the best possible local choice for a point in the sense that this choice leaves the maximum number of compatible points on the right half for the remaining points on the left half. Algorithm 3 gives a detailed description of our greedy heuristic. The main computational cost in the algorithm is the sorting of the points according to their $y$ coordinates at line 3 of Algorithm 3. Hence, the algorithm has a theoretical worst case running time of $O(N \log N)$.

We have divided the points into horizontal bands and then divided each horizontal band into two halves. These two choices imply that for every point we are restricting our choice of the points it can be paired with (or we are restricting the set of neighbors in the compatibility graph). Theoretically, this may seem to be troublesome and one can devise inputs where this scheme will perform badly. However, for the practical inputs this scheme works very well as shown in Table 6.1. Moreover, one can also deal with this by repeating the algorithm with different starting $y$-coordinates for the bands between $y_{\text{min}}$ and $y_{\text{min}} + 2\Delta$ and taking the best result among them. Another alternative is that instead of dividing the point sets, we consider for each point all the possible unassigned points it can be paired with and use the same criterion as in Algorithm 3 for choosing a partner for this point. This algorithm is a special case of the greedy algorithm for matching where we pick one edge at random from the graph, remove all edges corresponding to the picked vertices and then pick another edge from the graph. The ratio of the number of edges in the optimal matching to the number of edges returned by this algorithm is upper bounded by 2. Thus the ratio of the total processing time of the points paired with this algorithm to the optimal processing time is $3/2$ assuming a constant processing time for each point. However, in experiments on the practical datasets this algorithm performs worse than Algorithm 3 and takes more time. A working scheme that can be used is to run both the algorithms and take the best of the two. This ensures that we always have a splitting result that is practically good and also has a worst case theoretical bound.
Algorithm 3 Greedy algorithm for 2 robots

1: \textbf{Input:} Vector of x and y coordinates of points, \( \mathbf{x}, \mathbf{y} \); Parameters \( s_{\text{min}}, \Delta \). \( [\mathbf{x}, \mathbf{y}] \) denotes the concatenated vectors \( \mathbf{x} \) and \( \mathbf{y} \).
2: \textbf{Output:} Set \( P \) of subsets of \( S \) of cardinality \( \leq 2 \) that partitions \( S \).
3: \[ [\mathbf{x}, \mathbf{y}] = \text{ysort}(\mathbf{x}, \mathbf{y}) ; \quad // \text{Sort according to y-coordinates} \]
4: \[ y_{\text{min}} = \text{minimum}(\mathbf{y}); y_{\text{max}} = \text{maximum}(\mathbf{y}); \]
5: \[ \text{numbands} = \left\lceil \frac{y_{\text{max}} - y_{\text{min}}}{2\Delta} \right\rceil \quad // \text{Number of bands} \]
6: \textbf{for} \( i = 0 \) to \( \text{numbands} - 1 \) \textbf{do}
7: \[ [\mathbf{u}, \mathbf{v}] \leftarrow \text{Points with y-coordinates in the range } (y_{\text{min}} + 2i\Delta, y_{\text{min}} + 2(i + 1)\Delta) \]
8: \[ [\mathbf{u}, \mathbf{v}] = \text{xsort}(\mathbf{u}, \mathbf{v}) \]
9: \[ u_{\text{div}} = \text{xmedian}(\mathbf{u}) \quad // \text{Median of } x-\text{coordinates} \]
10: \[ u_{\text{min}} = \text{minimum}(\mathbf{u}); u_{\text{max}} = \text{maximum}(\mathbf{u}); \]
11: \textbf{if} \( |u_{\text{div}} - u_{\text{min}}| < s_{\text{min}} \) \textbf{then}
12: \[ u_{\text{div}} \leftarrow u_{\text{min}} + s_{\text{min}} \]
13: \textbf{end if}
14: \[ [\mathbf{u}_l, \mathbf{v}_l] \leftarrow \text{Points with } x-\text{coordinates in the range } (u_{\text{min}}, u_{\text{div}}) \]
15: \[ [\mathbf{u}_r, \mathbf{v}_r] \leftarrow \text{Points with } x-\text{coordinates in the range } (u_{\text{div}}, u_{\text{max}}) \]
16: \textbf{for} \( j = 1 \) to \( \text{length}(\mathbf{u}_l) \) \textbf{do}
17: \[ k \leftarrow \text{Index of point on right hand side that has minimum } x-\text{coordinate among all points that respect the constraints. If there is more than one such point we take the one with the minimum } y-\text{coordinate.} \]
18: \textbf{if} such a \( k \) exists \textbf{then}
19: \[ P \leftarrow P \cup \{(u_l[j], v_l[j]), (u_r[k], v_r[k])\} \]
20: \textbf{else}
21: \[ P \leftarrow P \cup \{(u_l[j], v_l[j]), *\} \]
22: \textbf{end if}
23: \textbf{end for}
24: \textbf{end for}
25: \[ L \leftarrow \text{Set of indices of unassigned points on right side} \]
26: \[ P \leftarrow P \cup \{(*, (u_r[k], v_r[k]))\}, \forall k \in L \]
27: \textbf{return} \( P \)

6.4.3 Results

The results of using both the greedy algorithm and optimal (matching) algorithm for the splitting problem, along with the corresponding running times, are shown in Table 6.1. The value of the parameters used for obtaining the results are \( \Delta = 8 \text{ mm}, s_{\text{min}} = 96 \text{ mm}. \)

We have used an implementation of Edmond’s algorithm available in the Boost Graph Library [99] to solve the MCM problem. The datasets used represent typical datasets that are used in the industry. Table 6.1 shows that for smaller datasets (say less than 30000 points), although the matching algorithm performs better, the running time is much
Table 6.1: Performance Comparison of Splitting between Greedy and Matching Algorithm

<table>
<thead>
<tr>
<th>Number of points</th>
<th>Number of pairs</th>
<th>Number of singletons</th>
<th>Time (sec)</th>
<th>Number of pairs</th>
<th>Number of singletons</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1396</td>
<td>695</td>
<td>6</td>
<td>0.45</td>
<td>696</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>11109</td>
<td>3029</td>
<td>5051</td>
<td>0.97</td>
<td>4137</td>
<td>2835</td>
<td>94</td>
</tr>
<tr>
<td>27810</td>
<td>13840</td>
<td>130</td>
<td>5.17</td>
<td>13905</td>
<td>0</td>
<td>1528</td>
</tr>
<tr>
<td>135300</td>
<td>67649</td>
<td>2</td>
<td>107</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167536</td>
<td>83739</td>
<td>58</td>
<td>172</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181758</td>
<td>90866</td>
<td>26</td>
<td>217</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>198570</td>
<td>99279</td>
<td>12</td>
<td>246</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>211856</td>
<td>105845</td>
<td>166</td>
<td>288</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

higher and hence it is not practical to use it for large datasets. In fact, the Boost Graph implementation fails for large datasets (resulting in the blanks in Table 6.1 for larger datasets). Moreover, our computational experiments (last five rows of Table 6.1) show that the ratio of the number of singletons to the number of points is very small, hence for practical purposes the greedy algorithm performs quite well. Figures 6.3 and 6.4 show two example datasets and the assignment of paired points to the two robots. The units of length on the two axes are microns. In Figure 6.4, the spread of the dataset along the $x$-axis is approximately 120 mm. As the minimum distance to be maintained between the two robots $s_{\text{min}} = 96$ mm, we cannot process most of the points in parallel and consequently there are a large number of singletons in the middle.

6.5 Ordering Problem

In this section we present algorithms to find an order of processing the points that minimizes the travel cost while ensuring that the compatible points are processed simultaneously. We formulate the problem as a multi-dimensional TSP (such TSP’s also arise in the circuit testing literature [57, 121]). We use a three step approach to solve this problem: (1) Find a path through the compatible pairs by solving a TSP on the set of point pairs (pair space). The solution of this TSP in the pair space induces a tour for each robot in $\mathbb{R}^2$. Note that even if we find the optimal tour in the pair space the optimality is with
Figure 6.3: Splitting and assignment of points by greedy algorithm for the dataset of 1396 points.

respect to the given pairing of the points and it may be possible that another feasible pairing of the same cardinality gives a better tour. (2) Use a local search heuristic in the tour of each robot to find a better tour in the pair space while respecting the constraints. (3) Incorporate the singletons to be processed by each robot in this tour by using a cheapest insertion heuristic.

6.5.1 TSP in Pair Space (PTSP)

For formulating the TSP in the pair space or pair TSP (PTSP) we have to first define a metric in the pair space between two pairs that is meaningful to our problem. Since the relative motion of the robot and the points are constrained to be only along the $x$-axis and $y$-axis, a natural measure of distance, $d_{ij}$, between two points $[(x_{i1}, y_{i1}), (x_{i2}, y_{i2})]$ and $[(x_{j1}, y_{j1}), (x_{j2}, y_{j2})]$ in the pair space is given by:

$$
\max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, |y_{i1} - y_{j1}|, |y_{i2} - y_{j2}|) \quad (6.4)
$$

This distance gives the cost incurred for robot 1 to reach $(x_{j1}, y_{j1})$ from $(x_{i1}, y_{i1})$ and robot 2 to reach $(x_{j2}, y_{j2})$ from $(x_{i2}, y_{i2})$ simultaneously. This distance measure is symmetric and satisfies the triangle inequality. The formal problem statement for PTSP can
thus be written as:

**Problem Statement:** Given a set of pairs of points, \( P = \{((x_1, y_1), (x_2, y_2))\}_{i=1}^{m} \), and a distance defined on the pairs by Equation 6.4, find a minimum cost tour on the weighted complete graph \( G = (V, E) \), where \( V = \{1, 2, \ldots, m\} \) indexes the elements of \( P \) and weights on the edges in set \( E \) are distances between the pairs.

We note that this problem formulation has the implicit assumption that the two robots start traveling from one pair at the same time toward the next pair and they leave the next pair only when both of them have finished processing, i.e., one robot cannot travel while the other is processing. This ensures that the two robots travel while satisfying the geometric constraints if they travel between the points with the same velocity. The cost obtained is thus an upper bound on the travel cost (as defined in Section 6.3) and can be improved using more sophisticated trajectory optimization schemes. The TSP is an NP-hard problem and, in general, it is not even possible to get a solution within a constant factor of the optimal solution [41]. However, in our case the distance metric is symmetric and satisfies the triangle inequality. For this case, there are polynomial time heuristics some of which guarantee a solution within a constant factor of the optimal solution. A few popular heuristics for solving the TSP [48] are (a) Nearest Neighbour heuristic (b)
Insertion heuristics (c) Minimum Spanning Tree (MST) heuristic (d) Christofides’ heuristic (e) Lin-Kerninghan heuristic. The heuristics (a), (b), (c) and (d) are usually used to construct a tour from scratch whereas (e) is used to improve a given tour. An alternative approach is to solve an integer program formulation of the TSP with a cutting plane method [30]. However, these methods tend to be more computationally expensive. The practical algorithms for TSP with triangle inequality use a combination of these methods to solve the problem. In this thesis, we use the TSP solver Concorde [2] to solve the PTSP, which has implementations of the above heuristics as well as the cutting plane method. For the results in this thesis, we used the Quick-Boruvka heuristic to compute a tour from scratch and the chained Lin-Kerninghan heuristic to improve the tour. We have observed from our computational experiments that when using a different heuristic (say nearest neighbor heuristic) to compute the initial tour even though the initial tour lengths may be different the improved tour lengths did not have any significant differences.

6.5.2 Order Improvement Heuristic

As discussed before, even if we get an optimal tour of the PTSP, the optimality of the solution is with respect to the chosen compatible pairs and it may be possible to improve the tour length by changing the point pairings. We have observed that the individual TSP tours induced by the PTSP tour contain self-crossings (i.e., the tour intersects itself). We note that although the Lin-Kerninghan tour improvement heuristic is intended to remove such crossings, in our problem it does so in the pair space. Therefore, we can further improve the PTSP tour by removing the self-crossings in the individual TSP tours of the robots provided the constraints are satisfied. This removal of self-crossings is equivalent to changing the pairing among the points. Figure 6.5 shows a simple example of a crossing in the TSP tour of robot 2. The initial pairings are given by (1, a), (2, b), (3, c), (4, d). If the pairings (2, c) and (3, b) are feasible then we obtain a new tour in the pair space given by \{\ldots, (1, a), (2, c), \ldots, (3, b), (4, d), \ldots\}. Thus the removal of self-crossings in \( \mathbb{R}^2 \) correspond to changing the pairings among the points.

Note that for a TSP in \( \mathbb{R}^2 \), when using the max metric, removal of a self-crossing is a necessary but not sufficient condition for improving the tour cost (whereas for Euclidean metric it is both a necessary and sufficient condition). Furthermore, in our PTSP formu-
Figure 6.5: The left figure shows the TSP tour of robot 1 whereas the right figure shows the self crossings observed in the TSP tour of robot 2. The initial pairings were (1,a), (2,b), (3,c), (4,d) whereas the new pairing in the lower cost tour is (1,a), (2,c), (3,b), (4,d), provided the new pairs are compatible.

Figure 6.5: The left figure shows the TSP tour of robot 1 whereas the right figure shows the self crossings observed in the TSP tour of robot 2. The initial pairings were (1,a), (2,b), (3,c), (4,d) whereas the new pairing in the lower cost tour is (1,a), (2,c), (3,b), (4,d), provided the new pairs are compatible.

lation, the cost between two consecutive points is determined by one of the two robots. Thus, if the crossing occurs in the tour of the other robot we will not improve the overall tour cost by removing it, although the tour cost of the individual robot may improve. For example, if the cost between the pairs (1, a) and (2, b) given by Equation 6.4, is determined by the distance between points 1 and 2 then removing the self-crossing doesn’t improve the overall tour cost although it may improve the individual tour cost of robot 2.

We implemented this order improvement heuristic in C++ where we consider two sets of two consecutive pairs. The number of pairs between the two sets, i.e., between (2, b) and (3, c) in Figure 6.5 is a parameter (say $k_c$) that we can set. We first use the heuristic on the tour of robot 2 and then use it on the tour of robot 1; reversing the order did not give any significant difference in results. Table 6.2 shows the results of solving the TSP in the pair space. We computed the initial tour using the Quick-Boruvka method and then used chained Lin-Kerninghan heuristic to improve the tour. For both of these we used the implementations available in the TSP solver Concorde [2]. However, the distance function for this multi-dimensional TSP is not available within Concorde and we had to implement our own distance function. The running times were less than 400 seconds for all the cases. For the tour improvement heuristic, we chose the parameter $k_c = 5000$. The running time is dependent on the value of $k_c$ and we did not see any substantial improvements in the tour cost with higher $k_c$. We observe an improvement of 5% to 8% in the tour cost using our tour improvement heuristic on the tested datasets and the running times are less than 600 seconds. All the run times are obtained on an IBM T43p laptop (2.0 GHz processor, 1GB RAM). The final improved tour cost is given in the
last column of Table 6.2.

Table 6.2: TSP tour obtained in pair space with improved cost given by the order improvement heuristic

<table>
<thead>
<tr>
<th>Number of pairs</th>
<th>Quick Boruvka Tour Cost (m)</th>
<th>Lin Kerninghan Tour Cost(m)</th>
<th>Improved Tour Cost (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>67649</td>
<td>124.428</td>
<td>110.343</td>
<td>103.289</td>
</tr>
<tr>
<td>83739</td>
<td>140.898</td>
<td>124.091</td>
<td>115.462</td>
</tr>
<tr>
<td>90866</td>
<td>121.523</td>
<td>106.434</td>
<td>100.253</td>
</tr>
<tr>
<td>99729</td>
<td>165.527</td>
<td>148.353</td>
<td>140.162</td>
</tr>
<tr>
<td>105845</td>
<td>150.255</td>
<td>132.048</td>
<td>121.911</td>
</tr>
</tbody>
</table>

6.5.3 Singleton Insertion Heuristic

We incorporate the singleton points for each robot in its individual tour induced by the PTSP tour using a cheapest insertion heuristic, i.e., we insert a point in the tour so that the total increase in the tour cost is minimum. Let \( i = (i_1, i_2) \) and \( j = (j_1, j_2) \) be two consecutive pairs where the subscript 1 denotes that the point is to be processed by robot 1 and subscript 2 is for robot 2. Let \( k_1 \) be a point to be inserted in the tour of robot 1. Suppose that we want to insert \( k_1 \) between \( i_1 \) and \( j_1 \). If \( (k_1, i_2) \) do not form a compatible pair, we find the minimum distance move to be made by robot 2 to a point compatible with \( k_1 \). Otherwise, the second robot may stay at the same place. Let \( k_2 \) be the point at which we have the robot 2 when robot 1 is at \( k_1 \). The increase in the cost of the tour due to the insertion of point \( k_1 \) between \( i_1 \) and \( j_1 \) is then \( C_{i_1 k_1} + \max(C_{k_1 j_1}, C_{k_2 j_2}) - \max(C_{i_1 j_1}, C_{i_2 j_2}) \) where \( C_{pq} \) denotes the distance given by max metric between points \( p \) and \( q \). We want to insert \( k_1 \) such that this tour cost increase is minimum. For each singleton, this is a linear time algorithm.

6.5.4 Evaluation of the Algorithm

In Section 6.3 we presented the path planning problem formulation wherein we proposed to divide the problem into two subproblems, namely, the splitting problem and the ordering problem. Thereafter, in the next two sections we provided algorithms for solving the two problems and provided simulation results showing the performance of the individual algorithms on example datasets. In this subsection we attempt to answer the
following question: How good is the overall algorithm? The ideal answer to this question is a theoretical bound on the ratio of the cost of optimal solution to the cost of the solution of this algorithm. However, we currently do not have any such guarantees. An alternative approach, especially useful to practitioners, is to compute the performance gain achieved in using a $K$-robot system over a single robot system. Here we provide simulation results that show the performance gain achieved in using a two-robot system over a single robot system.

The cost of the path for any robot is the sum of the travel cost ($T_t$) to visit the points and the processing cost ($T_p$) of processing the points on the path (which is proportional to the number of points, assuming constant processing time). However, the measurement unit of the travel cost is that of length and the unit of processing cost is time in our problem formulation. So we need a weight factor between the two costs that depends on the relative importance of the two costs. We can define the performance gain obtained in using a $K$-robot system over a single robot system as:

\[
\text{Performance Gain} = w \frac{T_p^{(1)}}{T_p^{(K)}} + (1 - w) \frac{T_t^{(1)}}{T_t^{(K)}}
\]

(6.5)

where $w$ is the weight factor ($1 \geq w \geq 0$) and the superscript denotes the number of robots used.

Table 6.3 shows the performance gain achieved by using a two robot system over a single robot system using the algorithm described in this chapter. We have assumed that $w = 0.5$ or the total travel time and total processing time are equally weighted. The processing cost for the single robot is proportional to the number of points whereas the processing cost of the two robot system (given in the second column of Table 6.3) is proportional to the sum of the number of pairs plus singletons (given in Table 6.1). The ratio of these two costs, i.e., \( \frac{T_p^{(1)}}{T_p^{(K)}} \) is nearly 2. The third column in Table 6.3 gives the travel cost of the two robot system whereas the fourth column gives the travel cost of the single robot system (obtained by using the Concorde implementation of the Lin-Kernighan heuristic on a Quick-Boruvka initial tour [2]). The travel cost of the two robot system is slightly higher than the travel cost of the single robot system. We believe that this is due to a combination of the following facts: (a) we have designed the whole algorithm on first
minimizing the total processing cost while satisfying the constraints and then minimizing the travel cost while keeping the processing cost fixed, (b) we have an implicit assumption that one robot cannot travel while the other is processing, which can make one robot wait even when it can move. However, as the last column demonstrates, we still have a significant gain in the overall performance. In light of the point (b) above, we can also view this gain as a lower bound on the performance gain. Note that the performance gain is also dependent on the value of $w$. If $w \approx 1$, i.e., the processing cost dominates the travel cost, our algorithm performs very well. On the other hand, if $w \approx 0$, i.e., the travel cost dominates the processing cost, the algorithm performs poorly.

Table 6.3: Overall Performance gain achieved by using a 2-robot system over a single robot system. See text for details.

<table>
<thead>
<tr>
<th>Num. of points</th>
<th>Processing Cost</th>
<th>2-Robot Travel Cost (m)</th>
<th>1-Robot Travel Cost (m)</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>135300</td>
<td>67651</td>
<td>103.289</td>
<td>93.535</td>
<td>1.453</td>
</tr>
<tr>
<td>167536</td>
<td>83797</td>
<td>115.462</td>
<td>97.329</td>
<td>1.42</td>
</tr>
<tr>
<td>181758</td>
<td>90892</td>
<td>100.253</td>
<td>86.828</td>
<td>1.43</td>
</tr>
<tr>
<td>198570</td>
<td>99291</td>
<td>140.162</td>
<td>125.365</td>
<td>1.447</td>
</tr>
<tr>
<td>211856</td>
<td>106011</td>
<td>121.911</td>
<td>98.219</td>
<td>1.40</td>
</tr>
</tbody>
</table>

6.6 Extension to $K$-robot systems

In this section we extend our splitting and ordering algorithms for the two robot system to a $K$-robot system. For a $K$-robot system the splitting problem can be set up as partitioning the compatibility graph of the points into the minimum number of cliques of size less than or equal to $K$. This is a modification of the NP-hard clique partitioning problem [41], the difference being the presence of a bound on the maximum size of the clique in our case. Consequently, it is unlikely that there is a polynomial time optimal solution to this problem. Therefore we present here the extension of the greedy algorithm to the $K$-robot case.

6.6.1 Splitting Algorithm

We first define the $K$-robot splitting problem formally. For concreteness, we use our motivating problem in Figure 6.1 to represent the geometric constraints. We assume
that the architecture is such that the $K$ robots are mounted on the gantry in Figure 6.1 and each robot has independent actuation along the $x$-axis. The problem statement for the splitting problem is thus:

**Problem Statement:** Let $S = \{p_j\} = \{(x_j, y_j)\}, j = 1 \ldots N$, be a set of points in $\mathbb{R}^2$. Let $Q = \{Q_i\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_L, y_L)\}, i = 1, \ldots, N_q$, be an ordered set of $L$-tuples, $L \leq K$ such that $Q_i \cap Q_j = \phi$, each point in $S$ is present in a tuple in $Q$, and the points in each element of $Q$ respect the following constraints

$$x_{li} - x_{mi} \geq s_{\text{min}} - 2\Delta, \quad l, m \in \{1, \ldots, L\}, \quad l > m$$

$$\max_{k=1 \ldots L}\{\frac{1}{L} \sum_{k=1}^{K} y_{ki}\} \leq \Delta$$

(6.6)

Find such a $Q$ of minimum cardinality.

The constraints on the $x$-coordinates ensure collision avoidance and the constraints on the $y$-coordinates indicate that the robots are constrained to move along the $x$-axis but have a square footprint for processing. The greedy algorithm for the 2-robot case can be extended to the $K$-robot case in a straightforward fashion. The main steps of the algorithm are as follows:

- Sort the points according to their $y$-coordinates and divide the points into bands of height $2\Delta$.
- For each band divide the points into almost equal $K$ zones using $K$ medians of the band. Number the zones 1 to $K$ from left to right.
- Then starting with the leftmost available point in zone 1 at the first position in each tuple, assign a point to the $k$th position in the tuple if there is a point in zone $k$ that satisfies the constraints, where $1 < k \leq K$. If there is more than one such point, choose the one with minimum $x$-coordinate breaking ties with the minimum $y$-coordinate. If there are no such points, then add a virtual point at the $k$th position.
- Repeat until all points are assigned.

The performance of this algorithm on the example datasets, shown in Table 6.4, shows that we obtain a very good splitting of the points among the robots. The datasets
used are the same as that for the two robot case, and \( s_{\text{min}} \) and \( \Delta \) are 96 mm and 8 mm respectively.

**Table 6.4: Greedy Algorithm Results for four-head machine on example data files**

<table>
<thead>
<tr>
<th>Num. of points</th>
<th>Num. of quads</th>
<th>Num. of triples</th>
<th>Num. of pairs</th>
<th>Num. of singles</th>
<th>Total num. of tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>167536</td>
<td>41874</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>41894</td>
</tr>
<tr>
<td>198570</td>
<td>49632</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>49653</td>
</tr>
<tr>
<td>211856</td>
<td>52954</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>52975</td>
</tr>
<tr>
<td>135300</td>
<td>33792</td>
<td>29</td>
<td>8</td>
<td>29</td>
<td>33858</td>
</tr>
<tr>
<td>181758</td>
<td>45398</td>
<td>5</td>
<td>57</td>
<td>37</td>
<td>45497</td>
</tr>
</tbody>
</table>

### 6.6.2 Ordering Algorithm

The set \( Q \) obtained as a result of the splitting problem consists of point tuples of size less than or equal to \( K \). We set up the ordering problem as a multi-dimensional TSP in the \( K \)-tuple space, similar to the two robot case. Thereafter, we use the tour improvement heuristic to improve the tour by interchanging the points in the \( K \)-tuple space. The tuples of size less than or equal to \( K \) are then inserted into the tours of the robots using the cheapest insertion heuristic described previously. The distance between two \( K \)-tuples, say \( i \) and \( j \), is a generalization of the definition for \( K = 2 \) and is defined as

\[
\max \{ |x_{ik} - x_{jk}|, |y_{ik} - y_{jk}| \}_{k=1}^{K} 
\]

(6.7)

where \((x_{ik}, y_{ik}), (x_{jk}, y_{jk})\) are the the coordinates of the points in the \( k \)th positions of the \( K \)-tuples \( i \) and \( j \) respectively. The tour improvement heuristic and the heuristic for inserting tuples of size less than \( K \) are direct extensions of the heuristics for the two robot system. For improving the tour cost while not changing the processing cost, we first identify the self-crossings in the tours of an individual robot and then try to interchange the order of the points assigned to that robot to achieve reduction in travel cost. For incorporating a tuple with \( l \) elements \((l < K)\), we compute the insertion costs in the \( l \)-tuple space. The same concept and formula as used in Section 6.5.3 is used for determining the order of the tuples in the tour.

Table 6.5 shows the processing cost, the travel cost, and the overall performance
gain achieved by using a four robot system over a single robot system. The second and third columns give the processing and travel cost respectively for the four robot system. The definition of performance gain is the same as given in Section 6.5.4. Similar to the two robot case we see here that there is a substantial gain in the processing cost (the ratio of the second column to the first column in Table 6.5 is almost 4 for each case). However, the travel cost is worse in some cases (the ratio of travel cost in the third column and second, fourth and fifth rows of Table 6.5 to the travel cost in forth column and same rows is greater than 1) although we have an overall performance gain in all cases. Note that this performance gain is a conservative estimate and can be thought of as a lower bound on the performance gain that can be achieved.

### Table 6.5: Overall Performance gain achieved by using a 4-robot system over a single robot system

<table>
<thead>
<tr>
<th>Num. of points</th>
<th>Processing Cost</th>
<th>4-Robot Travel Cost (m)</th>
<th>1-Robot Travel Cost (m)</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>135300</td>
<td>33858</td>
<td>31.482</td>
<td>93.535</td>
<td>3.48</td>
</tr>
<tr>
<td>167536</td>
<td>41894</td>
<td>136.691</td>
<td>97.329</td>
<td>2.35</td>
</tr>
<tr>
<td>181758</td>
<td>45497</td>
<td>54.079</td>
<td>86.828</td>
<td>2.8</td>
</tr>
<tr>
<td>198570</td>
<td>49653</td>
<td>159.197</td>
<td>125.365</td>
<td>2.39</td>
</tr>
<tr>
<td>211856</td>
<td>52975</td>
<td>147.153</td>
<td>98.219</td>
<td>2.33</td>
</tr>
</tbody>
</table>

### 6.7 Conclusion

In this chapter we presented algorithms for path planning of a constrained $K$-robot system to cover a set of points in the plane. There are two relative degrees of freedom between the robots and the points, and each robot can process one point at a time within its processing footprint. The processing times for each point are assumed to be identical. We divided the path planning problem into two subproblems of splitting (and assigning) the points to each robot, and then determining an order of processing the points so that the overall tour cost is minimized. We showed that for two robots, the splitting problem can be solved optimally by converting it into a Maximum Cardinality Matching problem on a graph. However, the matching algorithm is too slow for large datasets and we developed a suboptimal $O(N \log N)$ greedy algorithm that exploits the geometric structure of our problem. For the ordering problem we first formulate and solve a TSP in
the pair space (PTSP). We then improve the solution of the PTSP by identifying necessary conditions (self-crossings) for tour improvement on the (PTSP induced) tours of the individual robots. We also give a cheapest insertion heuristic on the pair space to incorporate the singletons in the ordering algorithm. We provide computational results showing the performance gain achieved by a two robot system using our algorithm over a single robot system. Finally, we generalize our splitting and ordering algorithms to $K$-robot systems and provide computational results showing their performance on typical industrial datasets for $K = 4$.

The division of the overall problem into a splitting problem and ordering problem is made to make the problem more tractable. However, this approach will lead to a suboptimal solution. An important question for the future is to obtain theoretical bounds on the suboptimality of this solution procedure. Another future direction is to look at other methods of changing the initial pairing that we obtain from splitting so that we have a better TSP tour. In the next chapter, we explore extensions to systems where the processing time at each point may be different.
CHAPTER 7
Minimum Time Point Assignment for Coverage by Two Constrained Robots

7.1 Introduction

In this chapter we revisit the splitting problem introduced in the previous chapter. Here, we relax the assumption of identical processing time for each point. In the context of laser drilling operation the processing time may differ depending on the radius of the hole (e.g., in drilling by trepanning, where the diameter of the hole to be drilled is larger than that of the laser beam).

The solution for the splitting problem presented in the previous chapter is not optimal when the processing times are different. In this chapter we show that when the processing times of the points are different, the splitting problem can be formulated as a maximum weighted matching problem (MWM) on a suitably constructed weighted graph. Thus, the splitting problem can be solved in $O(N^3)$ time in general [67, 82]. However, this is not suitable for large datasets (in our applications, approximately $10^5$ points). Therefore we present a greedy algorithm that takes $O(N^2)$ time. We prove that the ratio of the processing time obtained with our greedy algorithm to that of the optimal processing time is within a factor of $3/2$. We also provide a simple example to show that this is a tight bound. Finally, for the example setup shown in Figure 6.1, we provide computational results showing that the greedy algorithm gives solutions very close to the optimal solution on typical industrial datasets.

This chapter is organized as follows: In Section 7.2, we briefly summarize the relevant literature. In Section 7.3, we provide solution algorithms and results for the 2–robot point splitting problem. Finally, in Section 7.4, we present our conclusions and future work. A preliminary version of this chapter appeared in [18].
7.2 Related Literature

The problem of assigning points to two geometrically constrained robots is not widely studied in the literature. A related problem that is similar to the problem we address is the minimum makespan scheduling problem [88]. There are various versions of the minimum makespan scheduling problem [88]. The problem closest to our interest is minimum makespan scheduling with precedence constraints. This is a NP-hard problem, even for two machines. However, there are two important distinctions between the minimum makespan problem with precedence constraints and our minimum time assignment problem with geometric constraints.

- Minimum makespan scheduling problem has precedence constraints, which means that if two jobs are constrained, then one cannot be started before the other is finished. On the other hand, we have a geometric constraint that only states that two jobs cannot be done simultaneously.

- In the basic version of the minimum makespan scheduling we can assign a new job to a machine when it is finished with the currently assigned job. However here we can start a new job only when both the jobs are finished. This is because our problem occurs as a subproblem of a more general problem where the two robots need to move from one point to another before processing them and also maintain the geometric feasibility conditions, for example, no-collision conditions.

Although the minimum makespan problem with precedence constraints is NP-hard, our problem has a polynomial time solution. We convert the problem to a maximum weighted matching on a suitably constructed graph. Maximum weighted matching is a well studied problem and can be solved in $O(N^3)$ time (Edmonds [36]) where $N$ is the number of vertices of the graph. There are also algorithms that are slightly faster (e.g., Micali and Vazirani’s $O(\sqrt{|V||E|})$ algorithm [74]).

7.3 Splitting Problem

In this section we provide solution algorithms for the splitting problem. The splitting problem consists of assigning the set of points to each robot so as to minimize the
total processing time while respecting the geometric constraints. A solution to this problem ensures maximum parallelization of the processing operations. To be concrete, we will use the geometric constraints for the problem shown in Figure 6.1. However, the results are valid for constraints specified by other types of metrics and in higher dimensional spaces. We call a pair of points a compatible pair (of points) if they can be processed together while respecting the geometric constraints. Any two compatible pairs are called a disjoint compatible pair if the points belonging to the two pairs are distinct. In a compatible pair of points, the point having the larger processing time is called the dominant point and the point having a smaller processing time is called the non-dominant point.

The formal statement for this problem is:

**Problem Statement:** Let $S = \{p_i\} = \{(x_i, y_i)\}$, $i = 1, 2, \ldots, N$, be a set of points in $\mathbb{R}^2$ with the processing time of point $p_i$ being $t_i$. Let $P$ be a set of ordered subsets of $S$ of size less than or equal to 2 that partitions $S$, i.e., $P = \{(p_i, p_j)\} \cup \{(p_k, \ast)\} \cup \{(*) , p_l\}$, $i, j, k, l \in \{1, 2, \ldots, N\}, i \neq j \neq k \neq l$, where $\ast$ denotes a virtual point and any pair $(p_i, p_j) \in P$ respects geometric constraints of the form $f(x_i, y_i, x_j, y_j) \leq 0$. For the system shown in Figure 6.1 the constraints are the following:

\[
\begin{align*}
|x_i - x_j| & \geq s_{\text{min}} - 2\Delta \\
|y_i - y_j| & \leq 2\Delta
\end{align*}
\]

(7.1)

where $s_{\text{min}}$ is the minimum allowable distance between the two robots and $2\Delta \times 2\Delta$ is the processing footprint of each robot. Find such a $P$ that minimizes the cost given by

\[
C = \sum_{\text{all pairs}} \max(t_i, t_j) + \sum_{\text{all singletons}} t_k
\]

(7.2)

The ordered pair $(p_i, p_j)$ denotes that $p_i$ is assigned to robot 1 and $p_j$ to robot 2. Moreover $(p_k, \ast)$ denotes that $p_k$ is a singleton assigned to robot 1, while $(\ast, p_l)$ denotes that $p_l$ is a singleton assigned to robot 2. In Equation 7.1, the constraint on the $x-$coordinates ensure collision avoidance between the robots. The constraint on the $y-$coordinates indicate that the robots are constrained to move along the $x-$axis but have a square footprint for processing.
7.3.1 Optimal Algorithm for Splitting

The cost of any partition of \( S \) given by Equation 7.2 can be rewritten as follows:

\[
C = \sum_{\text{all pairs}} \max(t_i, t_j) + \sum_{\text{all singletons}} t_k, \quad (i \neq j \neq k)
\]

\[
= \sum_{\text{all pairs}} (t_i + t_j - \min(t_i, t_j)) + \sum_{\text{all singletons}} t_k
\]

\[
= \sum_{\text{all points}} t_k - \sum_{\text{all pairs}} \min(t_i, t_j), \quad (i \neq j)
\]

Since the sum of the processing times of all points is constant, the problem of minimizing \( C \) is equivalent to maximizing \( C_1 = \sum_{\text{all pairs}} \min(t_i, t_j) \), i.e., finding a subset of compatible pairs of points such that the sum of the processing times of the non-dominant points (in each pair) is maximized. The solution to this problem is given by the maximum weight matching on a weighted graph. The construction of the graph is as follows: Let \( G = (V, E) \) be a weighted graph, with the set of vertices \( V = \{1, 2, \ldots, N\} \) corresponding to the points. An edge \((i, j)\) exists between two vertices if the corresponding points \( p_i, p_j \) are compatible, i.e., they satisfy Equation 7.1. The weight \( w_{ij} \) on each edge is given by \( \min(t_i, t_j) \). Thus a MWM on this graph gives a set of disjoint compatible pairs where the sum of the processing times of non-dominant points is maximized.

The MWM problem on a graph is a well known combinatorial optimization problem and can be solved in \( O(N^3) \) time (Edmonds [36]). However, there are applications where \( N \) can be quite large, and consequently, the matching algorithm is not practical. For example, \( N \) can be of the order of \( 10^5 \) in the laser drilling application of Figure 6.1. Hence, we provide a \( O(N^2) \) time greedy algorithm that gives a suboptimal solution with the worst case approximation ratio of 3/2. Note that there are deterministic constant factor approximation algorithms in the weighted matching literature that have running time linear in the number of edges, i.e., \( O(N^2) \) in the worst case, (see [113], and references therein). The simplest such algorithm has a worst case approximation ratio of 2 which implies that the ratio of the value of \( C_1 \) in the optimal matching to that in the obtained matching is upper bounded by 2. For our problem it means that the ratio of the sum of the processing times of the non-dominant points in the optimal solution to that of the greedy solution is less than or equal to 2. This implies that the cost given by Equation 7.2
is within a factor of $3/2$ of the optimal cost. However, we can construct simple examples on datasets of 4 points where our proposed algorithm performs better than this graph matching approximation algorithm. There are more sophisticated $O(N^2)$ time algorithms that have an approximation ratio arbitrarily close to $4/3$ for our problem. However, the implementation of such algorithms are more complicated and it is not clear whether those algorithms have been implemented in practice. Therefore, we present a simple greedy algorithm, not requiring explicit construction of the compatibility graph, whose total cost is provably within $3/2$ of the optimal cost.

### 7.3.2 Greedy Algorithm

For our problem, the trivial algorithm where each point is processed individually takes time at most twice that of the optimal solution for the two robots (if there were $k$ robots this bound would be $k$). Thus any useful algorithm should have a theoretical worst case bound less than 2. When there are no constraints between the points the optimal algorithm to pair up the points is to sort the points in the order of decreasing processing times and pair up consecutive points (starting from the top) in this sorted list. When there are constraints on the points that can be paired, this algorithm is not optimal. However, we prove that it is within a factor of $3/2$ of the optimal solution. The basic algorithm is:

1. Sort the points in order of decreasing processing times.

2. Start with the point having largest processing time. Pair each point with a compatible point (that is not already paired) that has the largest processing time. If there is more than one compatible point with same (largest) processing time choose one at random. If there is no compatible point available make the point a singleton.

The cost of this algorithm is determined by the second step above. A straightforward implementation compares each point to every other point in the worst case and thus the algorithm has a worst case running time of $O(N^2)$.

**Lemma 7.3.1** Let $T_G$ be the total processing time when the points are paired according to the greedy algorithm and $T_O$ be the optimal processing time. Then $T_G/T_O \leq 3/2$. 
Proof: Let the set \( P_G \) be the solution of the greedy algorithm and \( P_O \) be the solution of the optimal algorithm. The ratio of the greedy cost to optimal cost can be written as

\[
\frac{T_G}{T_O} = \frac{\sum_{P_G} \max(t_i, t_j) + \sum_{P_G} t_k}{\sum_{P_O} \max(t_i, t_j) + \sum_{P_O} t_k}
\] (7.3)

Let \( P'_G \subseteq P_G \) and \( P'_O \subseteq P_O \) be subsets of the two solutions that contain the same points. We partition the whole solution set into such partial subsets. Let \((p_i, p_j) \in P'_G\) be a pair in the greedy solution (w.l.o.g. assume \(t_i\) is the greatest element in that set; we will explain later why we do not consider that a singleton can be the greatest element). Then, it suffices to consider the three forms for the sets \( P'_G \) and \( P'_O \) shown in Figure 7.1. For brevity, we use the indices of the points to denote the points, for example, \((p_i, p_j)\) is denoted by \((i, j)\).
1. We have a sequence in the optimal pairing given by \(j, (i, k_1), (k_2, k_3), \ldots, (k_{m-1}, k_m),\)
\(k\) \((m \text{ odd})\) with the corresponding pairing in the greedy solution given by \((i, j), (k_1, k_2),\)
\(\ldots, (k_m, k)\), i.e., there are two singletons in optimal solution not present in greedy solution.

2. We have a sequence in the optimal pairing given by \(j, (i, k_1), (k_2, k_3), \ldots, (k_m, k)\)
with the corresponding pairing in the greedy solution given by \((i, j), (k_1, k_2), \ldots, (k_{m-1}, k_m), k\), \(k\) \((m \text{ even})\).

3. We have a sequence in the optimal pairing given by \((i, k_1), (k_2, k_3), \ldots, (k_m, k),\)
\((j, l_1), (l_2, l_3), \ldots, (l_m, l)\) with the corresponding pairing in the greedy solution given
by \((i, j), (k_1, k_2), \ldots, (k_{m-1}, k_m), k, (l_1, l_2), \ldots, (l_{m-1}, l_m), l\), i.e., there are two
singletons in the greedy solution that are not present in the optimal solution.

Note that we can have at most two singletons in \(P'_G\) that may not be a singleton in \(P'_O\).
So, either we can reach all singletons in the greedy set by this construction or they are
present as singletons in the optimal solution also. The first case is of interest to us where,
if the singleton had the highest cost it would be paired up (by the nature of our greedy
algorithm) because it has a feasible partner. Thus the singleton in \(P'_G\) cannot have the
highest cost. We now show that for each case above the partial greedy cost \(T'_G\) is within
a factor of \(3/2\) of the partial optimal cost \(T'_O\).

\textit{Case 1:} The ratio of partial greedy cost to optimal cost is

\[
\frac{T'_G}{T'_O} = \frac{t_i + \sum_{u=1}^{(m-1)/2} \max(t_{2u-1}, t_{2u}) + \max(t_{m}, t_k)}{t_i + t_j + t_k + \sum_{u=1}^{(m-1)/2} \max(t_{2u}, t_{2u+1})}
\]

If we assume \(t_{k_1} \geq t_{k_2}, t_{k_3} \geq t_{k_4}, \ldots, t_{k_m} \geq t_k\),

\[
\frac{T'_G}{T'_O} = \frac{t_i + t_{k_1} + \sum_{u=1}^{(m-1)/2} t_{2u+1}}{t_i + t_j + t_k + \sum_{u=1}^{(m-1)/2} \max(t_{2u}, t_{2u+1})} \tag{7.4}
\]

As \(t_j \leq t_{k_1}\) \((\text{otherwise } P'_G \text{ would have the pair } (i, k_1))\) and \(\max(t_{k_2u}, t_{k_{2u+1}}) \geq t_{k_{2u+1}}\),
all the terms in the numerator and denominator of Equation 7.4 are identical, with \(t_k\) as an
extra term in the denominator implying that \(T'_G/T'_O \leq 1\), i.e., the greedy cost is less than
the optimal cost. Therefore the processing times of the greedy pair cannot be of the above
form. Similarly, we can show that \( t_{k_1} \leq t_{k_2}, t_{k_3} \leq t_{k_4}, \ldots, t_{k_m} \leq t_k \) is not possible. Thus, the sequence in \( P'_G \) will have alternating subsequences where the first point dominates and then the second point dominates and so forth.

Now, assume that the greedy pairs satisfy \( t_{k_1} \leq t_{k_2}, t_{k_3} \leq t_{k_4}, \ldots, t_{k_{l-1}} \leq t_{k_l} \), for some \( l < m \) and \( t_{k_{l+1}} \geq t_{k_{l+2}} \), i.e., the pairs have their second point dominating up to the pair \((k_{l-1}, k_l)\). We do not assume anything about rest of the points. We develop the proof in two steps. We first show that the ratio of the two partial costs up to this pair is less than \( 3/2 \). We then argue that if there are more such changes in the position of the dominant points in the greedy pairs the ratio of partial costs up to those pairs will also be less than \( 3/2 \). Thus we can extend this scheme to the whole set \( P'_G \) and say that the cost \( T^{p'}_G/T^p_O \leq 3/2 \). We start with the first step. The ratio of the costs up to the point where there is the first change in dominating point is

\[
r = \frac{t_i + t_{k_2} + \cdots + t_{k_{l-2}} + t_{k_l} + t_{k_{l+1}}}{t_i + t_j + \cdots + \max(t_{k_{l-2}}, t_{k_{l-1}}) + \max(t_{k_l}, t_{k_{l+1}})}
\]

Without loss of generality assume \( t_{k_l} \geq t_{k_{l+1}} \) (proof with \( t_{k_l} \leq t_{k_{l+1}} \) is analogous). Then \( t_{k_{l-1}} \geq t_{k_{l+1}} \), otherwise \((k_l, k_{l+1})\) would be in \( P'_G \) instead of \((k_{l-1}, k_l)\). Using \( \max(t_{k_{2u}}, t_{k_{2u+1}}) \geq t_{k_{2u}} \) for all \( u < (l - 2)/2 \) in the denominator and \( C = \sum_{u=1}^{(l-3)/2} t_{k_{2u}} \), we have

\[
r \leq \frac{t_i + t_{k_{l-2}} + t_{k_l} + t_{k_{l+1}} + C}{t_i + t_j + \max(t_{k_{l-2}}, t_{k_{l-1}}) + t_{k_l} + C}
\]

\[
\leq \frac{t_{k_{l-2}} + t_{k_l} + t_{k_{l+1}}}{\max(t_{k_{l-2}}, t_{k_{l-1}}) + t_{k_l}}
\]

\[
\leq \frac{t_{k_{l-1}} + t_{k_l} + t_{k_{l+1}}}{t_{k_{l-1}} + t_{k_l}} \quad \text{assume w.l.o.g } t_{k_{l-1}} \geq t_{k_{l-2}}
\]

\[
= 1 + \frac{t_{k_{l+1}}}{(t_{k_{l-1}} + t_{k_l})} \leq \frac{3}{2} \quad (\because t_{k_{l+1}} \leq t_{k_{l-1}} \leq t_{k_l})
\]

If \( t_{k_{l-1}} \leq t_{k_{l-2}} \), we could replicate the same arguments substituting \( t_{k_{l-2}} \) in place of \( t_{k_{l-1}} \). When the sequence in \( P'_G \) changes from a second point dominating pair to a first point dominating pair there is only slight increase in the cost of the denominator and consequently the bounds are satisfied. However, when the sequence again changes to a second point dominating pair, we can write the partial sum similar to the above derivation.
Let the change be at \( k_n \), i.e., \( t_{k_n+1} \geq t_{k_n+2} \). The second line of equation 7.5 is then

\[
\frac{t_{k_i-2} + t_{k_i} + t_{k_{i+1}} + t_{k_n-2} + t_{k_n} + t_{k_{n+1}}}{\max(t_{k_i-2}, t_{k_i-1}) + t_{k_i} + \max(t_{k_n-2}, t_{k_n-1}) + t_{k_n}}
\]

with \( t_{k_{n+1}} \leq t_{k_{n-1}} \leq t_{k_n} \). The next steps are analogous to the steps in equation 7.5. We can proceed likewise until the whole set \( P'_G \) is covered and the approximation ratio is thus less than 3/2.

**Case 2:** Assume that \( t_{k_1} \leq t_{k_2}, t_{k_3} \leq t_{k_4}, \ldots, t_{k_{m-1}} \leq t_{k_m} \) in \( P'_G \). w.l.o.g; if this were not true we could have proceeded as described in case 1 above. We also have \( t_k \leq t_{k_{m-1}} \leq t_{k_m} \), otherwise \( P'_G \) would have the pair \((k_m, k)\) instead of \( k_{m-1}, k_m \). The ratio of the cost of \( P'_G \) to \( P'_O \) is less than 3/2 as shown below. The second step is obtained by using the assumption above in the numerator and using the fact that \( \max(a, b) \geq a, \forall a, b > 0 \) in the denominator. The fourth step is obtained from the third step using the fact that if \( a/b > 1 \) and \( c/d < 1 \) then \((a + c)/(b + d) < a/b\).

\[
\frac{t_i + \sum_{u=1}^{m/2} \max(t_{k_{2u-1}}, t_{k_{2u}}) + t_k}{t_i + \sum_{u=1}^{(m-2)/2} \max(t_{k_{2u}}, t_{k_{2u+1}}) + \max(t_k, t_{k_m}) + t_j} \leq \frac{t_i + \sum_{u=1}^{m/2} t_{k_{2u}} + t_k}{t_i + \sum_{u=1}^{m/2} t_{k_{2u}} + t_j} \leq \frac{t_i + t_{k_m} + t_k + C}{t_i + t_{k_m} + t_j + C} \quad (C = \sum_{u=1}^{m/2-1} t_{k_{2u}}) \leq \frac{t_i + t_{k_m} + t_k}{t_i + t_{k_m}} = 1 + \frac{t_k}{t_i + t_{k_m}} \leq \frac{3}{2} \quad (\therefore t_k \leq t_{k_m}, t_k \leq t_i)
\]

**Case 3:** Here, it may be possible that \((k, l)\) form a pair in the greedy solution (see Figure 7.1 bottom). However, for worst case analysis, it suffices to look at the case when \( k \) and \( l \) are singletons. Here again we assume that \( t_{k_1} \leq t_{k_2}, t_{k_3} \leq t_{k_4}, \ldots, t_{k_{m-1}} \leq t_{k_m}\), \( t_{l_1} \leq t_{l_2}, t_{l_3} \leq t_{l_4}, \ldots, t_{l_{n-1}} \leq t_{l_n} \) without any loss of generality. We also have \( t_k \leq t_{k_{m-1}} \) and \( t_l \leq t_{l_{n-1}} \) because \((k_{m-1}, k_m)\) and \((l_{n-1}, l_n)\) are the greedy pairs. The partial costs
$T^p_O$ and $T^p_G$ are then

\[ T^p_G = t_i + \sum_{u=1}^{m/2} \max(t_{k2u-1}, t_{k2u}) + \sum_{u=1}^{n/2} \max(t_{l2u-1}, t_{l2u}) + t_k + t_l \]

\[ + \sum_{u=1}^{m/2} t_{k2u} + \sum_{u=1}^{n/2} t_{l2u} + t_k + t_l \]

\[ T^p_O = t_i + \sum_{u=1}^{(m-2)/2} \max(t_{k2u}, t_{k2u+1}) + \max(t_{k_m}, t_k) \]

\[ + \max(t_j, t_l) + \sum_{u=1}^{(n-2)/2} \max(t_{l2u}, t_{l2u+1}) + \max(t_{n}, t_l) \]

\[ \geq t_i + \sum_{u=1}^{m/2} t_{k2u} + \sum_{u=1}^{n/2} t_{l2u} + t_j \]

The ratio of the partial costs is then

\[ \frac{T^p_G}{T^p_O} \leq \frac{t_i + \sum_{u=1}^{m/2} t_{k2u} + \sum_{u=1}^{n/2} t_{l2u} + t_k + t_l}{t_i + \sum_{u=1}^{m/2} t_{k2u} + \sum_{u=1}^{n/2} t_{l2u} + t_j} \]

\[ = \frac{t_{k_m} + t_{k_{m-1}} + t_n + t_{l_{n-1}} + t_k + t_l + C}{t_{k_m} + t_{k_{m-1}} + t_n + t_{l_{n-1}} + t_j + C} \]

\[ (\text{where } C = t_i + \sum_{u=1}^{m/2} t_{k2u} + \sum_{u=1}^{n/2} t_{l2u}) \]

\[ \leq \frac{t_{k_m} + t_{k_{m-1}} + t_n + t_{l_{n-1}} + t_k + t_l}{t_{k_m} + t_{k_{m-1}} + t_n + t_{l_{n-1}}} \]

\[ \leq 1 + \frac{t_k + t_l}{t_{k_m} + t_{k_{m-1}} + t_n + t_{l_{n-1}}} \leq \frac{3}{2} \]

Using $t_k \leq t_{k_{m-1}} \leq t_{k_m}$, $t_l \leq t_{l_{n-1}} \leq t_{l_n}$

The greedy solution set can be represented as an union of disjoint sets of the forms described by the 3 cases above and each partial sum is less than $3/2$ times the corresponding optimal partial cost. Therefore, we have $T_G \leq 3/2 T_O$. ■

The above bound is a tight bound. We give an example on 4 points where the cost of the greedy algorithm is $3/2$ times the cost of the optimal algorithm. Let the 4 points \{i, j, k, l\} have unit processing times and the set of compatible points be \{(i, j), (i, k), (i, l), (j, k)\}. The greedy algorithm may pick the pair (i, j) since all the pairs have equal
processing times. Thus, the greedy solution is \( \{(i, j), k, l\} \) with cost 3. The optimal cost is 2 with solution \( \{(i, l), (j, k)\} \).

In the preceding discussion we gave algorithms to split the points among the two robots. However, we did not make an explicit assignment of the points to the robots. If the geometric constraints are such that one robot is always constrained to be on the left of the other (as in Figure 6.1) we can always assign the point in the pair with smaller \( x \)-coordinate to one robot and its partner to the other. The ordering problem can then be set up as a multi-dimensional TSP in the pair space [17]. However, if the geometric constraints are different, for example, the constraints are based on maintaining a minimum (or maximum) Euclidean inter-robot distance, then the above scheme may not be appropriate. In this case, we need to solve the assignment and ordering problems simultaneously. Let \((p_i, p_j)\) and \((p_k, p_l)\) be two pairs and the robots have to move from one pair to the other. The question we need to answer here is: Does the robot at \( p_i \) or the one at \( p_j \) move to \( p_k \)? If \( p_i \) moves to \( p_k \) and \( p_j \) moves to \( p_l \) the travel cost is \( d_1 = \max\{d(p_i, p_k), d(p_j, p_l)\} \). On the other hand if \( p_i \) moves to \( p_l \) and \( p_j \) moves to \( p_k \) the travel cost is \( d_2 = \max\{d(p_i, p_l), d(p_j, p_k)\} \). The robots should move so that the cost incurred is minimum of \( d_1 \) and \( d_2 \). Thus, we can define the cost of travel between two pairs as the minimum of \( d_1 \) and \( d_2 \). The cost metric defined above satisfies the triangle inequality. The TSP tour on the pairs using this cost metric will give the order in which the pairs of points should be traversed and hence also the assignment of points to the robots as they travel. An approach for solving the multidimensional TSP problem is discussed in [17].

### 7.3.3 Computational Results

We now provide computational results showing the performance of the greedy algorithm on example industrial data sets. However, the processing time assigned to each point is randomly generated in the range 0.01 to 0.1 seconds. We implemented the greedy algorithm in C++. The results of the splitting problem using both the greedy algorithm and optimal (matching) algorithm along with the corresponding running times are shown in Table 7.1. The value of the parameters used for obtaining the results are \( \Delta = 8 \) mm, \( s_{\text{min}} = 96 \) mm. To compare our greedy algorithm results against the optimal results, we
assumed all the processing times to be equal (one) and used an implementation of Edmond’s algorithm available in the Boost Graph Library [99] to solve the MCM problem (which is equivalent to the MWM in this case). Table 7.1 compares the processing times of the greedy solution (second column) to that of the optimal solution (fourth column). It can be seen from Table 7.1 that the greedy algorithm is always within a factor of 1.5 of the optimal solution (see last column) and is much faster than the optimal algorithm (compare third and fifth columns). For the larger datasets, the MCM implementation does not return results within a reasonable time. So we have provided the greedy algorithm results for them. Table 7.2 shows the results for the greedy algorithm where the processing times of individual points vary from 0.01 to 0.1 seconds. The last column of Table 7.2 gives an upper bound on the ratio of the greedy cost to optimal cost (because the sum of all the point processing times is always less than or equal to twice the optimal cost). Table 7.2 (last column) shows that the greedy algorithm performs quite well for the large datasets and takes less than 1 minute (third column). The second dataset gives the worst result.

Table 7.1: Performance Comparison of Greedy and Matching algorithms for Splitting, with unit processing time for each point.

<table>
<thead>
<tr>
<th>Number of points</th>
<th>Greedy Algorithm</th>
<th>Matching Algorithm</th>
<th>Ratio $T_G/T_O$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Processing Time ($T_G$) (sec)</td>
<td>Running Time (sec)</td>
<td>Processing Time ($T_O$) (sec)</td>
</tr>
<tr>
<td>1396</td>
<td>788</td>
<td>1</td>
<td>700</td>
</tr>
<tr>
<td>11109</td>
<td>8537</td>
<td>8</td>
<td>6972</td>
</tr>
<tr>
<td>27810</td>
<td>16569</td>
<td>15</td>
<td>13905</td>
</tr>
</tbody>
</table>

Table 7.2: Performance of Greedy Algorithm on large datasets.

<table>
<thead>
<tr>
<th>Number of points</th>
<th>Processing Time ($T_G$) (sec)</th>
<th>Running Time (sec)</th>
<th>Sum of all Point Processing Times ($T$) (sec)</th>
<th>Ratio $2T_G/T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1396</td>
<td>37.66</td>
<td>1</td>
<td>72.05</td>
<td>1.05</td>
</tr>
<tr>
<td>11109</td>
<td>407.47</td>
<td>8</td>
<td>554.82</td>
<td>1.47</td>
</tr>
<tr>
<td>27810</td>
<td>726.34</td>
<td>15</td>
<td>1390.99</td>
<td>1.04</td>
</tr>
<tr>
<td>135300</td>
<td>3435.78</td>
<td>25</td>
<td>6782.24</td>
<td>1.01</td>
</tr>
<tr>
<td>167536</td>
<td>4236.38</td>
<td>30</td>
<td>8388.49</td>
<td>1.01</td>
</tr>
<tr>
<td>198570</td>
<td>5009.33</td>
<td>40</td>
<td>9919.32</td>
<td>1.01</td>
</tr>
<tr>
<td>211856</td>
<td>5336.82</td>
<td>47</td>
<td>10564.50</td>
<td>1.01</td>
</tr>
</tbody>
</table>
However, in this point set there are at least 2835 points that have to be singletons even in the optimal solution (we can see this from the MCM solution). So the bound on the approximation ratio for this data set is not tight and the actual approximation ratio may be better. All the running times are obtained on an IBM T43p Thinkpad with a 1.8 GHz processor and 1 GB RAM.

### 7.4 Conclusion

In this chapter we considered the point assignment for processing problem for a constrained two-robot system covering a point set with non-identical point processing times. Since each robot can process one point at a time, the geometric (e.g., collision avoidance) constraints and kinematic constraints between the robots result in restrictions on points that can be simultaneously processed. We showed that the assignment problem can be solved optimally by converting it into a maximum weighted matching problem on a suitably defined graph. However, this $O(N^3)$ matching algorithm is too slow for large datasets and so we presented an $O(N^2)$ greedy algorithm and proved that it is guaranteed to return a solution within $3/2$ of the optimal solution. Note that our approach can handle fairly general geometric constraints between the robots, and is not restricted to planar point sets.

Future work includes computational comparison of our greedy algorithm solutions with approximation algorithm solutions for weighted matching, in terms of both solution quality and running time. The extension of the algorithms proposed here for $K > 2$ robots is also important. Relaxing the assumption that both robots are simultaneously stationary (when one or both robots are processing points) and obtaining algorithms that have constant factor approximation guarantees for the overall path planning problem (of point splitting and point ordering) is another important extension.
CHAPTER 8
Summary and Future Work

In this thesis, we presented optimization-based tools and algorithms for incorporating geometric constraints in two classes of robot motion planning problems: (a) State to State motion planning in the presence of geometric, kinematic, and dynamic constraints. (b) Multiple robot point set coverage in the presence of geometric constraints. For state to state motion planning our approach for handling geometric constraints is philosophically opposite to the conventional approach. We directly use the contact information from the dynamic simulation algorithm to obtain a collision-free input. On the other hand, traditional methods discard any input that do not satisfy the geometric contact constraints. Our simulation-based method is applicable to both motion planning problems with and without contact. However, the quality of the solutions depend on the accuracy of the dynamic simulation, which is not good for problems with intermittent contact. As a step towards enhancing the utility of our motion planning algorithm for problems with intermittent contact, we studied two key sources of error that arise in contact dynamics simulation and provided solutions for them. The geometric constraints that we studied in the first part of the thesis always arise from the physical requirement of non-penetration between objects. However, there are another class of problems in multi-robot systems where, in addition to the non-penetration requirements, geometric constraints arise may also from task requirements. In the second part of the thesis we studied such a problem, namely, minimum-time point set coverage by multiple robots. In this case, before solving the point-to-point motion planning problem we need to assign the points to the robots and specify an order in which the points should be visited (i.e., find a tour for each robot). The geometric constraints need to be considered not only during the point-to-point motion planning stage but also while finding a tour for each robot. We provided solutions for the problem of assigning the points to the robots and specifying the order of visiting the points while ensuring that the geometric constraints are satisfied.

In Chapter 3, we presented a dynamic simulation-based kinodynamic motion planning algorithm. Our algorithm relies on the following insight: the (virtual) contact force
that is obtained from a complementarity-based dynamic simulation algorithm can be used to obtain a collision-free input for a robot moving among obstacles. Since our algorithm uses the contact information to find a collision free input (given any input to the dynamic simulation algorithm), it is easy to find collision-free inputs from a state within a narrow passage in the configuration space. Thus, for solving narrow passage problems, our algorithm has advantages over all current sampling based algorithms that only integrate the Newton-Euler equations while building up the graph representing the free space. We provided some simple examples to illustrate our algorithm and restricted our attention to problems where collision avoidance is the key requirement. The safety distance for avoiding collisions is a user specified parameter. In forming the discrete-time dynamics model, we have used a linear approximation of the distance function. A possible future direction of research is to relate the safety distance to the error in distance function approximation and to the dynamic simulation time-step. Systematic evaluation of our planner with existing randomized planners is another area of future research.

In Chapter 4, we presented an interior point algorithm for distance computation between objects described by intersections of convex implicit surfaces (including polyhedra). This chapter was motivated by the relative sparsity of collision detection algorithms for non-polyhedral objects in the current literature. The theoretical time-complexity of our algorithm is $O(n^{1.5})$ for objects described by intersections of self-concordant functions (that includes planes and quadrics), where $n$ is the total number of surfaces describing the two objects. This is the first algorithm for quadrics with a polynomial running time guarantee. The practical running time behavior is linear in the number of constraints for all the classes of implicit surfaces that we have studied. The speed at which distance computations can be performed enables real-time dynamic simulations and haptic interactions at 1 KHz rates. An important additional advantage of our collision detection algorithm is that it provides a uniform framework for distance computation between convex objects described as arbitrary intersections of polyhedra, quadrics, or any convex implicit surface. Furthermore, within this framework we can handle global affine deformations of implicit surface objects, and index change deformations of superquadrics (or hyperquadrics) without significant computational overhead. Finally, we showed that continuous collision detection for linearly translating implicit surface objects can be performed by solving two
related convex optimization problems. For polyhedra and quadrics, we established that the computational complexity of this continuous collision detection problem is $O(n^{1.5})$. There are several directions for future work. One direction is to explore alternative interior point algorithms (LOQO [111] and IPOPT [115], for example) to test their performance on the minimum distance problem. Performing warm starts, where a good initial estimate for the solution is available, can potentially improve the running time when there is coherence. This may be best achieved by a combination of interior point methods and sequential quadratic programming approaches. An important future project is to extend this approach to nonconvex objects, modeled as unions of convex shapes and incorporate it in a hierarchical framework. Longer term directions for future research include tracking closest points continuously for haptics applications, and extending this approach to performing continuous collision detection with both rotational and translational motion. The approach can be extended to compute the translational penetration depth in a given motion direction for two intersecting objects.

In principle, the motion planning method of Chapter 3 also applies to planning problems in manipulation and grasping, where the robot may be in intermittent contact with other objects. However, as illustrated in Section 2.3, the approximations made in current dynamic simulation algorithms can lead to wrong simulation results in the presence of contacts. In Chapter 5, we developed a geometrically-implicit time-stepper that integrates collision detection within the dynamic simulation time-step and works for implicit-surface objects. This helps in removing two key sources of error in multibody dynamics simulation: (a) the use of polyhedral representations of smooth bodies, (b) the decoupling of collision detection from dynamic simulation. For formulating our geometrically-implicit time-stepper, we wrote the contact constraints as a constraint between the contact force and a distance function dependent on the closest points on the objects. The closest points satisfy a set of algebraic constraints that are obtained from the KKT conditions of the minimum distance problem. Our time-stepper is the first geometrically-implicit time-stepper that does not assume that the distance between the objects is given in closed form. We presented our time-stepper for both rigid bodies as well as locally-compliant bodies. There are several directions for future work in this area. An important future direction is to address the question of existence and uniqueness of solutions of the NCP we formulate.
More extensive numerical experimentation, and comparison of these solutions with solutions obtained when the closest distance is computed through a function call is another future direction. Numerical studies to precisely quantify the trade-offs between the computation speed and physical accuracy of simulations for different object representations (e.g., polyhedral, implicit, spline), friction approximations, and the choice between geometrically explicit or implicit methods would be an useful extension. Although we have restricted our discussion to convex objects, we believe that this framework can be extended to non-convex objects described as unions of convex objects as well as parametric surfaces.

In Chapter 6 and 7 of this thesis, we developed path planning algorithms for multiple robots that are required to cover a point set in minimum time while satisfying given geometric constraints. In the point set coverage tasks that we considered, the robots are required to spend some time at each point for task completion and the number of points is large (on the order of $10^5$). Our problem is motivated by an industrial laser drilling system and can be modeled as a $K-$TSP with geometric constraints. We presented a heuristic to solve the problem by dividing it into two subproblems: (1) Splitting problem: Assign the points to the $K$ robots subject to geometric constraints, such that the total processing time is minimized. (2) Ordering problem: Find an order of processing the split points by formulating and solving a multi-dimensional Traveling Salesman Problem (TSP) in the $K$-tuple space with an appropriately defined metric to minimize the total travel cost.

In Chapter 6, we developed algorithms for solving the splitting and ordering problem assuming that the processing time at each point is identical. For two robots we showed that the splitting problem can be optimally solved by converting it to a maximum cardinality matching problem (MCM). However algorithms to solve MCMs take $O(N^3)$ time (where $N$ is the number of points) and are too slow for the large datasets that we consider. Therefore, for the laser drilling system that we consider in the thesis, we developed a $O(N \log N)$ time greedy algorithm by exploiting the geometric structure of the machine. We solved the ordering problem by formulating and solving a multi-dimensional TSP in the pair space (PTSP). We improved the PTSP solution by identifying and removing self-crossings in the tours of the individual robots (that are induced by the PTSP tour). This is equivalent to reducing the ordering cost by changing the partners of
the pairs obtained from the splitting problem while keeping the processing cost constant. We generalized our algorithms to $K$-robot systems and provided computational results showing the performance of 2-robot and 4-robot systems on typical industrial datasets. In Chapter 7, we developed solutions for the splitting problem for cases where the processing times are different. For $K = 2$ robots, we provided an optimal solution by converting the splitting problem into a maximum weighted matching (MWM) problem. Since the MWM algorithm also takes $O(N^2)$ time and is too slow for large datasets, we presented a greedy algorithm that has a running time of $O(N^2)$. We proved that the ratio of the total processing time obtained using this greedy algorithm to the optimal processing time is always less than 1.5 and the bound is tight. We provided computational results on industrial datasets that demonstrate that the performance of the greedy algorithm is much better in practice. The two-step approach of solving the geometrically constrained $K$-TSP by dividing it into a splitting problem and ordering problem makes the problem tractable. However, this approach can lead to a suboptimal solution. An important question for the future is to obtain theoretical bounds on the worst-case performance of this solution procedure. More generally, obtaining approximation algorithms for the geometrically constrained $K$-TSP is a challenging problem for the future. Another area of future work is to generalize the greedy splitting algorithm (for non-identical processing times) to a $K$-robot system and give a worst case performance bound for the algorithm.
APPENDIX A

Optimal Control Formulation for Minimum Time Multiple Robot Point Set Coverage

We present here the mixed integer optimal control formulation of the minimum time multiple robot point set coverage problem. Although we do not use this general problem formulation in our chapter, we include it here for completeness. Let there be $K$ robots that have to visit a point set $S = \{p_i\}$, where they have to perform some task that takes time $\tau_i, i = 1, \ldots, N$. Let $q_k(t)$ be the state of the $k$th robot at time $t$ and $\dot{q}_k = f_k(q_k, u_k)$ be the state update equation, where $u_k$ is the time dependent control input. Let

$$\omega_{ijk} = \begin{cases} 
1 & \text{if robot } k \text{ visits point } i \text{ at time } t_j, \\
0 & \text{otherwise}
\end{cases} \quad (A.1)$$
where \( t_j, j = 1, \ldots, N \), are the (unknown) times when the robots reach a point in \( S \). The mixed integer optimal control problem is given by

Minimize \( t_f \)

s.t.:

\[ \dot{q}_k = f_k(q_k, u_k), \quad k = 1, \ldots, K \] (A.2)

\[ \sum_{j=1}^{N} \sum_{k=1}^{K} \omega_{ijk} q_k(t_j) = p_i, \quad i = 1, \ldots, N \] (A.3)

\[ \sum_{j=1}^{N} \sum_{k=1}^{K} \omega_{ijk} q_k(t_j + \alpha \tau_i) = p_i, \] \( \forall \alpha, \quad 0 < \alpha \leq 1 \), \( i = 1, \ldots, N \) (A.4)

\[ \sum_{j=1}^{N} \sum_{k=1}^{K} \omega_{ijk} = 1, \quad i = 1, \ldots, N \] (A.5)

\[ \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{K} \omega_{ijk} = N \] (A.6)

\[ \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{K} \omega_{ijk} \leq K, \quad j = 1, \ldots, N \] (A.7)

\[ h(q_1, q_2, \ldots, q_K) \leq 0 \] (A.9)

\[ g(u_1, u_2, \ldots, u_K) \leq 0 \] (A.10)

\[ t_j \leq t_f, \quad j = 1, \ldots, N \] (A.11)

where \( h(.) \) and \( g(.) \) are vector-valued vector functions representing constraints on the states (e.g., geometric constraints like inter-robot collision avoidance) and control inputs respectively. In addition, we can also include any constraints on the initial and final states of each robot. Equation A.2 states that the state evolution of the robots should obey the dynamics constraints. Equation A.3 states that for each point \( p_i \) there is exactly one robot \( k \) that visits the point at some time, say \( t_j \), and Equation A.4 states that the robot has to stay at the point \( i \) for time \( \tau_i \) to complete the task. Equation A.5 states that each point should be visited only once and Equation A.6 implies that a robot can only be at one
particular point at a time. Equation A.7 states that the total number of points that have to covered by all the robots is $N$ whereas Equation A.8 implies that at a given time $t_j$, at most $K$ points can be processed.


