SOLVING GEOMETRIC CONSTRAINT PROBLEMS THROUGH MONTE CARLO OPTIMIZATION

by

Daniel James Perry

A Masters Thesis Submitted to
the Faculty of the University of Utah
in Partial Fulfillment of the Requirements for the degree of

Master of Science

in

Computer Science

School of Computing
University of Utah
August 2008

ABSTRACT

Geometric constraint problems appear in many situations, including CAD systems, robotics, and computational biology. The complexity of these problems inspires the search for efficient solutions. We have developed a method to solve geometric constraint problems in the areas of geometric computation and robot path planning using configuration space subdivision. In this approach the configuration space, or parameter space, is subdivided and conservatively tested to find collision-free regions, which are then numerically searched for specific path solutions. This thesis presents a new more general approach to this last solution search step, using Monte Carlo optimization. In this new search approach, within a single subdivided area of configuration space, space is randomly sampled and then iteratively re-sampled based on importance weighting, until convergence to a solution with an acceptable error. We show that by using Monte Carlo optimization to extend configuration space subdivision we can solve higher dimensional problems more efficiently than configuration space subdivision by itself.

TABLE OF CONTENTS

Chapter 1

Introduction

One area of ongoing research in geometric computation is efficient solution to geometric constraint problems. Geometric constraint problems appear in many areas, including CAD, robotics, and computational biology. Because of their wide application, new approaches to solving geometric constraints are quite useful.

Geometric constraint problems appear often in a CAD system. Consider finding the medial surface of a complex model, or collision detection between moving parts in a machinery design. Constraint problems are also an important part of robotics, especially in the realm of path planning and collision detection. Additionally, computational biology also makes use of solutions to constraint problems, for example in the accurate calculation of collision and surface offset in simulations of molecules. Even computer games require the solution of geometric constraint problems - again consider character collision detection, proximity detection and related problems. Geometric constraint problems are general enough that they appear in many situations - resulting in a wide application of the methods used to find their solutions.

In the area of robotics, one interesting area that uses geometric constraint solutions is robot path planning. A path planning problem normally consists of finding a path

for a robot through a workspace that avoids collision with obstacles. The problem can be made more difficult by having more obstacles to navigate around, and by using a more complex robot.

We have developed a method for finding the solution to such constraint problems using configuration space subdivision. The constraints are sampled in specific configuration parameter ranges, which create hypercubes in n dimensional configuration parameter space. Using the constraint sampling, the hypercubes are conservatively tested as to whether they contain potential solutions. These hypercubes in configuration space are recursively subdivided and tested to narrow down regions of configuration space where solutions will be found. This configuration subdivision, constraint sampling, and testing approach results in regions or hypercubes in configuration space where solutions potentially lie, which then need to be searched for exact solutions. One simple approach to finding the exact solution is to keep subdividing until the size of the hypercubes are smaller than a specified error - then taking the center of the cube to be the solution. Alternatively, if an explicit derivative is available, the approach can use the Newton-Raphson and related methods to reach a specific solution. This process is described more fully in Section 3.3.1.

In this thesis we adapt Monte Carlo optimization as a general type of numerical search. In essence, the Monte Carlo algorithm optimizes the geometric constraint within each of the hypercubes resulting from configuration space subdivision, instead of subdividing down to the answer. The optimization is done by sampling the subspace, weighting the samples according to an error metric, and then iteratively re-sampling around samples with better weights until convergence. The error metric is created from the constraints, which guide the optimization algorithm towards a solution. This approach is described more fully in Section 3.3.2.

Monte Carlo methods in general are useful for solving complicated problems in higher dimensions [27, 23]. We have found this approach is also useful for configuration spaces of higher dimensions, because it only samples and tests the constraint in a random yet guided approach. Because it iteratively samples the constraint, Monte Carlo optimization matches well to the computational requirements of configuration space subdivision, and we have found that they fit well together.

By combining configuration space subdivision and Monte Carlo optimization, we are able to solve path planning problems to a higher precision, and solve constraint problems of a higher dimension than configuration space subdivision alone. This is demonstrated in both geometric constraint problems, and in robot path planning problems.

Chapter 2

Background

2.1 Literature Survey

Solving geometric constraints is an important part of current geometric computation research. Path planning is an important part of robotics research. There has been substantial work done in these related areas, using a variety of approaches. Additionally, there has been considerable work done in Monte Carlo optimization itself. Here work most relevant to this thesis is reviewed, and categorized into similar approaches.

2.1.1 Symbolic Analysis

One approach to solving geometric constraint problems include [16], that uses "degrees of freedom analysis" to solve geometric constraints. That type of analysis describes the constraints to be solved and the initial state of the geometry, and then outlines sequences of rotations and translations to satisfy the constraints, providing a plan for that particular solution symbolically. Once the plan is made, it can then be run numerically on different initial geometries. This provides an intuitive

approach to constraint solving, but requires specific plans to be made for each constraint, rather than an automatic solution through numerical computation as done in other approaches.

Recent work in solving geometric constraints has been done by [8, 15]. In their approach they reduce the problem of solving a geometric constraint to the problem of finding the zero-set of a system of NURBS. By doing so they are able to use the special attributes of NURBS to subdivide the constraint system representation into subproblems, each of which can be tested for potential solutions. This process enables efficient elimination of sub-sections where solutions cannot exist. Because the constraints are represented as NURBS, a Newton-Raphson like method can be used to numerically search for potential solutions with high accuracy.

This approach of reducing a geometric constraint problem to finding the solution of a system of constraint equations represented as NURBS has been applied to multiple problems, such as finding the minimum distance between a point and surface as in [25], or in solving the visibility problem as in [24]. Reducing the problem to the solution of a system of equations represented as NURBS allows for a very robust approach to solutions to constraint and other geometric problems.

One restriction to this approach is that the input geometry must also be represented as NURBS, which precludes its use in other geometry. Additionally, higher dimensional problems reach current hardware limitations, because of the memory-intensive, explicit representation of the constraint equation. This is in contrast to other approaches that only sample the constraint function to test for constraint satisfaction or optimization. Representing the system of equations as NURBS requires additional memory requirements.

2.1.2 Approximation through Sampling

Other approaches, instead of representing the constraint completely, only evaluate or sample the constraint at specific values. Then, using those samples they attempt to satisfactorily find the values at which the constraint is satisfied or optimized.

One approach that uses sampling as opposed to exact representation to solve robot path planning problems is described in [31]. Using approximate cell decomposition of the configuration space they are able to show path existence. The same authors later combine the approximate cell decomposition with a probabilistic road-map planner in [32] with good results.

We are developing a similar approach. Likewise, the constraint is only sampled in order to narrow down areas where potential solutions lie. These areas, which are hypercubes in the configuration space, are then searched for the answer using geometric techniques specific to the problem. This approach allows for some interactive results of problems that cannot be solved using complete representation on modern computers. Additionally this subdivision of configuration space allows for other than spline geometry, such as polygonal models.

Significant research has been done in path planning for robots in general, and in particular for path planning for robots with many degrees of freedom. A valuable introduction to path planning can be found in [5]. Work has not only been done on rigid robot bodies, but also on flexible bodies as in [12].

Because path planning in higher dimensions becomes difficult, some of the only solutions to high dimensional problems have used probabilistic road-map planners (PRM), such as [14]. Additionally [11] achieves good results for high dimension planners by improving a solution incrementally. Another approach that was successful at solving high dimensional path planning problems used a derivative of dynamic

programming, as described in [1]. Rapidly-exploring random trees (RRT) planners have also had success in path planning problems, up to 12 degrees of freedom [18, 17]. RRT planners can have difficulty in narrow passages, in response [30] adapts the RRT approach as a retraction-based planner with success.

2.1.3 Monte Carlo optimization

We use Monte Carlo optimization as the numerical search step in configuration space subdivision. There has been substantial research done on Monte Carlo optimization.

A good introduction to Monte Carlo methods in general are [27, 9, 13]. Monte Carlo methods as well as a thorough treatment of different Monte Carlo optimization techniques are discussed in [21].

The approach we are developing is similar to Monte Carlo optimization "evolution strategy", which is discussed at length and compared to most of the current optimization algorithms in [22, 23].

In developing this approach, we are using strategies from Genetic Algorithms as well. In particular, when re-sampling around a potential solution, one challenge is to determine the proper replacement strategy. In [19, 20] avoidance of replacement errors in cases of multiple solutions is discussed. This is also important in single solution strategies like that presented in this thesis.

An important aspect of our implementation of Monte Carlo optimization is the initial sampling approach. We explore three types of initial sampling, uniformly distributed random sampling, stratified sampling, and Halton-Hammersley sampling. Stratified sampling, also know as jittered sampling, is explained very well in [26], as applied to computer graphics. Halton and Hammersley based sampling are discussed in a straitforward manner in both [2], as used in computer graphics, and [3], as applied

to PRM robot planners.

2.2 State of the Art

It is worth it to emphasize background on those approaches most similar to this thesis. Currently there are many methods proposed for solving constraint problems, and especially path planning problems in robotics.

As mentioned, reducing the problem to the solution of a spline function is a very robust approach to solving this type of problem [8].

One method described in [28] solves motion planning problems by dividing free space in the workspace into star-shaped sections. Each section has a central point known as a guard. By dividing the free space into these star-shaped sections, the guards of the different sections can be connected to form road maps.

By using approximate cell decomposition of the configuration space [31] are able to find solutions to path planning problems. By subdividing the configuration space and using conservative tests, they are able to show that a path does exist. In [32], they extend their approach to include methods adapted from probabilistic road-map planners, for instances where standard subdivision cannot show if a path exists or not. They are able to solve problems with 4 DOF, and gain substantial speedup using the probabilistic road-map planning step.

This is different from the approach described in this thesis, because they use approximate cell decomposition to construct paths. If approximate cell decomposition is not able to find a path then techniques from probabilistic road-map planners are used to check for the potential path existence, warranting further cell subdivision. The approach proposed here on the other hand uses uniform random importance sampling to search each cell in the decomposition for a solution.

The probabilistic road-map planners have had success in motion planning problems of high dimensions, as in [17],[14], [4]. For example, [14] shows solutions to motion planning problems for robots with up to 7 DOF.

Probabilistic road-map planners are very successful in most situations, but still rely on stochastic processes, rather than a more deterministic process. The approach presented here uses configuration space subdivision to provide more determinism, then uses Monte Carlo optimization to search for the actual solutions.

2.3 Robot Configuration Space

This section provides a brief introduction to Robot Configuration Space, due to the very domain specific nature of the topic. For a more in-depth explanation see [5].

2.3.1 Configuration Space and Workspace

Two spaces in robotics will be discussed in this thesis - the robot workspace, and configuration space.

Workspace is the "real world" of the robot - the real environment in which the robot exists and moves.

Configuration space is the parameter space of a robot. A set of configurations or parameters are a way of describing the position of the robot in workspace, by describing what position the individual configurable parts of the robot are in. More formally, a robot R with n dimensions of freedom is configurable using n configuration parameters, which creates an n-dimensional configuration space.

For a simple example, consider a circle-shaped robot that can move in the x and y directions of a 2-dimensional Euclidean plane. The configuration of the robot

in this case is the same as the position of the robot in workspace - specified by the pair x, y. This would be called a 2-dimensional configuration space. Changing the robot into a sphere-shaped robot and allowing it to move in the z-direction as well would result in 3-parameters, and a 3-dimensional configuration space. The 3-dimensional configuration space is still the same as the position of the spherical robot in workspace. The configuration parameters are describing a translation of the robot from its original position. Converting from configuration parameters to workspace position in this case would be very simple. Given any point $[x_0, y_0]^T$ or $[x_0, y_0, z_0]^T$ on the robot at its initial position in workspace, and given the configuration parameters $[x, y]^T$ or $[x, y, z]^T$, the converting function $C_2 : \Re^2 \Rightarrow \Re^2$ or $C_3 : \Re^3 \Rightarrow \Re^3$ could be defined as,

$$P_2([x, y]^T) = [x_0, y_0]^T + [x, y]^T \tag{2.1}$$

or

$$P_3([x, y, z]^T) = [x_0, y_0, z_0]^T + [x, y, z]^T \tag{2.2}$$

The relationship between configuration space and workspace become a little different when different types of robots are considered. For example, consider a one segment, rotational robot arm that is "fastened" to the ground on one end allowing it to rotate a full 360 degrees. It has a 1-dimensional configuration space, but the configuration space and the position of the robot in workspace are no longer the same. The only parameter for the robot is the angle at which the arm will be positioned. If the configuration parameter were 90 degrees, the robot arm would be positioned with a 90 degree angle from the original position. As the configuration parameter ranges from 0 to 360, the robot arm rotates around - the end-effector drawing a circle

with a center at the fastened end, and the radius being the length of the one-segment arm. The configuration space is topologically a circle, it wraps around onto itself. For example, the position at 360 degrees would be the same as at 0 degrees.

Converting from configuration space to workspace is also straightforward in this case. To produce the workspace position of any point on the robot (x_0, y_0) (at its initial position), given the angle configuration parameter, θ, the converting function $T_1 : V \subset \Re \Rightarrow \Re^2$ could be defined as,

$$T_1(\theta) = R(\theta)[x_0, y_0]^T \tag{2.3}$$

where $R(\theta)$ is the 2-dimensional rotation matrix of θ.

A more complex example is created by adding another segment to the arm with the same rotational ability, resulting in a 2-dimensional configuration space, with one parameter controlling the angle of the first segment and the second parameter controlling the angle of the second segment. A value for each parameter dictates the position of the two-segment robot arm in workspace. The position of any pont $[x_0, y_0]^T$ on the first segment of the robot in its initial workspace position, can be defined in its configured position given parameters θ_1, θ_2, using the function $T_2 : V \subset \Re^2 \Rightarrow \Re^2$:

$$T_2(\theta_1, \theta_2) = R(\theta_1)[x_0, y_0]^T \tag{2.4}$$

and for any point $[x_1, y_1]^T$ on the second segment in the initial workspace position:

$$T_2(\theta_1, \theta_2) = R(\theta_1)([x', y']^T + R(\theta_2)([x_1, y_1]^T - [x', y']^T)) \tag{2.5}$$

Where $[x', y']^T$ is the position of the connection between the first and second segments in initial position. Topologically, this configuration space is a torus - both

dimensions wrap around on themeselves. However, for ease in visualization this space is drawn as a limited 2-dimensional plane - a torus cut open and spread out.

Continuing to add segments to the robot arm (and angle parameters to the configuration space) would increase the dimension of the configuration space.

Rotational and translating abilities can be combined in a single robot. A robot that can both translate in the plane and rotate within the plane could have two parameters controlling the translation of the robot, and one parameter controlling the rotation of the robot - resulting in a 3-dimensional configuration space. Any point $[x_0, y_0]$ on the robot in its initial position can be described in a configured position as a function of the configuration parameters. The configured position function $T_3 : V \subset \Re^3 \Rightarrow \Re^2$ could be defined as,

$$T_2([x, y]^T, \theta) = R(\theta)[x_0, y_0]^T + [x, y]^T \tag{2.6}$$

While this is not an exhaustive list of robot types, it gives an idea of the types of robots and configuration spaces considered in this thesis, and how their configuration parameters and workspace positions relate.

Consider a simple example illustrating the use of configuration space to plan a simple robot path. A simple robot that can translate in the plane is used, resulting in a 2-dimensional configuration plane. There are two obstacles in the workspace, and to plan around them the Voronoi diagram is found (which in this simple case is just the bisector line between them). The result is shown in workspace in Figure 2.1, with the corresponding configuration space diagram in Figure 2.2. The green line in this diagram represents the solution to the Voronoi diagram, which also represents the path on which the robot is drawn in the workspace figure. A correlation is not difficult to find between configuration space and workspace - because the configuration space

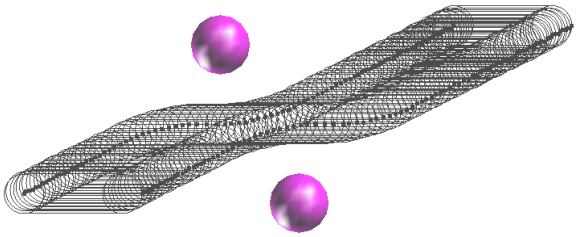

Figure 2.1: Simple translating robot, drawn along the Voronoi diagram of the two obstacles.

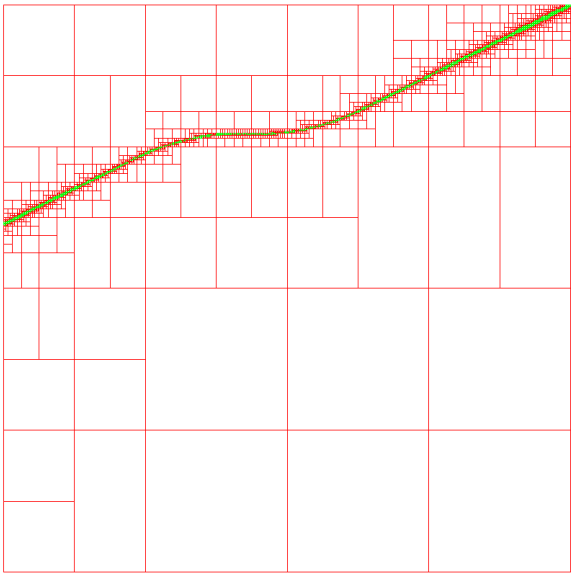

Figure 2.2: The configuration space diagram of Figure 2.1

only contains positional parameters of the robot. In other situations the correlation is not quite as trivial (consider the correlation of the configuration and workspace of the example given in Section 4.4.2).

2.3.2 Obstacles

Obstacles in workspace are objects with which the robot can potentially collide. Because any position of the robot can be described using configuration parameters, they

can be used to describe positions of the robot that collide with obstacles, also known as invalid configurations and positions. These invalid configurations are represented as a subset of the robot configuration space C, called C_{obs}. Then any configuration outside of C_{obs}, would be considered free space, or $C_{\text{free}} = C/C_{\text{obs}}$.

Consider for example the circular robot example from above, and add to the workspace a circular obstacle. The invalid configurations can be described as any (x, y) pair that position the robot such that it is colliding with the obstacle. Let (x_0, y_0) be the original position of the center of the robot, and r_r be the radius of the robot. Let the circular obstacle be described by a center (x_b, x_b) and radius r_b. Given that it is the only obstacle in workspace, the invalid configurations C_{obs} could be described as all configurations (x, y) satisfying this inequality,

$$\sqrt{((x_0 + x) - x_b)^2 + ((y_0 + y) - y_b)^2} - r_r - r_b \leq 0 \tag{2.7}$$

All valid configurations (x, y) or C_{free} would satisfy the inequality,

$$\sqrt{((x_0 + x) - x_b)^2 + ((y_0 + y) - y_b)^2} - r_r - r_b > 0 \tag{2.8}$$

In other words, any position of the robot that produces a signed distance less than 0 would be invalid. This is a very simple case. As the robots and workspaces become more complex, so does the job of describing both valid and invalid configurations.

In this thesis the configuration space C is decomposed using axis-aligned hypercubes, which results in an inexact separation of C_{obs} and C_{free} - some of the hypercubes will contain both valid and invalid configurations. Consequently any configuration hypercubes containing both valid and invalid configurations are considered part of C_{obs}.

A more complex case of describing C_{obs} is shown in Section 4.4.2.

2.3.3 Usefulness

The abstraction of robot configuration parameters as an n-dimensional space and then dividing it into C_{free} and C_{obs} subspaces is useful in robot path planning.

Abstraction allows a simplification of the assortment of configurations into which a robot can be placed. Even for a 2-dimensional rotational robot it would be difficult to exactly describe all of the potential collisions and free movement positions into which the robot can be placed, using only the workspace. By abstracting configurations into a set, and then identifying the various configurations as representing valid or invalid positions of the robot, it becomes much easier to then know exactly what configurations and resulting positions the robot can and cannot use.

This abstraction also allows the relationships between C_{free} and C_{obs} to become more clear. For example, it may not be entirely apparent from workspace alone that a robot cannot enter a particular target position from a given initial position. However with C_{free} and C_{obs} clearly identified, the possibility of a movement between the two positions can be determined comparatively easily - through an attempt to connect the two positions in configuration space. Configuration space representation also provides a way to establish a path or to prove path non-existance (assuming the configuration space decomposition is sufficiently precise). The abstraction means that a path can be represented as a curve in configuration space, rather than a sweeping of complex geometry through workspace - allowing the mentioned analysis benefits.

Once the C_{free} and C_{obs} have been identified more general methods, not necessarily specific to robotics, can be used in solving the problem. For example, consider again that C_{free} and C_{obs} have been identified and a path between an initial and

final position is sought. The space could be discretized and arranged in a graph, and then more general graph routing algorithms could be used to connect the two positions in configurations space. This could not be done in workspace alone.

Another of these more general tools is visualization. Depicting all positions in which a robot collides with obstacles in workspace would be very messy at best. Assuming the space can be visualized, depicting C_{free} and C_{obs} or subsets of them in configuration space would be much more clear. It is more clear because each point in configuration space corresponds to either a valid or invalid robot position - which is not as easy to see in workspace.

Chapter 3

Approach

3.1 Overview

In this thesis we seek the solution to geometric constraint problems related to motion planning involving robots with higher degrees of freedom. We use a configuration space subdivision approach, with a last solution step using Monte Carlo optimization. While other papers have used configuration space subdivision, and even incorporated some random searching for speed ([31, 32]), the approach presented in this thesis uses explicit Monte Carlo optimization to find solutions. Further, we seek to build generalized Voronoi diagrams to solve the motion planning problem, as in [10].

One of the safe paths of a robot through a workspace with obstacles is the same as the generalized Voronoi diagram (GVD) of the complete workspace and obstacles. Once an accurate generalized Voronoi diagram is found, the cell boundaries can serve as the robot's path to safely navigate the workspace. The Voronoi diagram or safe path can be described using constraints in the configuration space of the robot.

In other words, rather than decomposing the entire configurations space C into C_{obs} and C_{free}, we instead search for the Voronoi diagram in C, which represents in

a very compact sense, a roadmap through C_{free} [5]. The roadmap consisting of all configurations with equal distance to the closest obstacles. A specific path can then be created, following the roadmap, which will avoid the obstacles.

A Voronoi cell consists of all points nearest to a generating point [29]. The generalized Voronoi Diagram extends this definition so that a cell consists of all points nearest to a generating shape [5]. In our case, a diagram is created in configuration space, and the distance metric is defined to be the workspace minimum distance. The k obstacles in workspace, $\vec{O} = (o_1, \ldots, o_k)$, are the generating shapes for each Voronoi cell. A GVD cell consists of all n-dimensional configuration space points $\vec{p} \in \Re^n$ that are nearest to the generating shape. The distance function is the workspace Euclidean distance between the obstacle shape and the robot in a particular configuration, signified as $Q(\vec{p})$.

A particular configuration point $\vec{p}$ belongs to the GVD cell with generating shape o_j only if,

$$d(Q(\vec{p}), o_j) = min_i(d(Q(\vec{p}), o_i)) \tag{3.1}$$

Where $d(\cdot, \cdot)$ is the Euclidean distance between the two objects (the distance between the two closest points on each model). Additionally, the cell boundaries can be defined using this same structure.

For example, if $n = 3$, then all points can be found that belong to the GVD cell wall (surface) between cells generated by o_{i1} and o_{i2}, by solving for all points that satisfy these three equations:

$$d(Q(\vec{p}), o_{i1}) = d(Q(\vec{p}), o_{i2}) \tag{3.2}$$

$$d(Q(\vec{p}), o_{i1}) < min_{l,l\neq i1,i2}(d(Q(\vec{p}), o_l)) \tag{3.3}$$

$$d(Q(\vec{p}), o_{i2}) < min_{l,l\neq i1,i2}(d(Q(\vec{p}), o_l)) \tag{3.4}$$

In the $n = 3$ case the boundary curve where these cell wall surfaces intersect can also be defined, which is the intersection of three GVD cells generated by o_{i1}, o_{i2}, o_{i3}. They would need to satisfy these 5 equations:

$$d(Q(\vec{p}), o_{i1}) = d(Q(\vec{p}), o_{i2}) \tag{3.5}$$

$$d(Q(\vec{p}), o_{i2}) = d(Q(\vec{p}), o_{i3}) \tag{3.6}$$

$$d(Q(\vec{p}), o_{i1}) < min_{l,l\neq i1,i2,i3}(d(Q(\vec{p}), o_l)) \tag{3.7}$$

$$d(Q(\vec{p}), o_{i2}) < min_{l,l\neq i1,i2,i3}(d(Q(\vec{p}), o_l)) \tag{3.8}$$

$$d(Q(\vec{p}), o_{i3}) < min_{l,l\neq i1,i2,i3}(d(Q(\vec{p}), o_l)) \tag{3.9}$$

Continuing with the $n = 3$ case the point where multiple cells come together can also be defined, which the intersection of at least four GVD cells generated by $o_{i1}, o_{i2}, o_{i3}, o_{i4}$. They would need to satisfy these 7 equations:

$$d(Q(\vec{p}), o_{i1}) = d(Q(\vec{p}), o_{i2}) \tag{3.10}$$

$$d(Q(\vec{p}), o_{i2}) = d(Q(\vec{p}), o_{i3}) \tag{3.11}$$

$$d(Q(\vec{p}), o_{i3}) = d(Q(\vec{p}), o_{i4}) \tag{3.12}$$

$$d(Q(\vec{p}), o_{i1}) < min_{l,l\neq i1,i2,i3,i4}(d(Q(\vec{p}), o_l)) \tag{3.13}$$

$$d(Q(\vec{p}), o_{i2}) < min_{l,l\neq i1,i2,i3,i4}(d(Q(\vec{p}), o_l)) \tag{3.14}$$

$$d(Q(\vec{p}), o_{i3}) < min_{l,l\neq i1,i2,i3,i4}(d(Q(\vec{p}), o_l)) \tag{3.15}$$

$$d(Q(\vec{p}), o_{i4}) < min_{l,l\neq i1,i2,i3,i4}(d(Q(\vec{p}), o_l)) \tag{3.16}$$

These specific constraint problems (which while useful in robotics as discussed, are also more generally useful) are solved in Section 4.3.

In problems of higher dimensions, the components of the GVD are made up of hyper-surfaces. To find a large hyper-surface completely using sampling and approximate cell decomposition would be difficult because sufficient point solutions need to be found to accurately represent the surface. In those situations we propose using our approach to find *important points* in the configuration (or configurations of interest) which are normally just points instead of multi-dimensional surfaces.

These important points would be useful for finding the GVD, as well as for describing the the full C_{obs} and C_{free} subspaces. Finding these *important points* often requires over-constraint of the solutions.

One type of over-constraint would be very similar to the above discussion, where adding constraints to the GVD equations reduces the size of the solution set. Above using Equations 3.10-3.16 constraints were added which reduced the solution set from a set of surfaces to a set of points.

Another type of over-constraint in robot path planning less related to the GVD problem, more a sub-problem of describing C_{obs} completely, would be to constrain the number of collisions to be detected. Constraining the number of collisions would produce solutions of special cases of collision, helping to start to form an idea of what the complete C_{obs} subset looks like. Alternately, constraining the type of collision - for example only finding where the tip of the robot collides. A variety of other related constraints can be imposed on the problem, for example constraining the

range of rotation at which collision occurs.

Consider a path planning problem involving a grounded (non-translating) robot arm with four segments and four joints/hinges. All configurations of the robot arm where it collides with obstacles (C_{obs}) could be represented as a surface. While it may be difficult to find the complete surface, our approach would be able to find all configurations where only the tip of the robot collides. In the geometry of the C_{obs}-surface these solutions represent corners and edges where the surface changes. This would be a good starting point in understanding the shape of the surface. A more in depth discussion of this approach can be found in section 4.4.2.

3.2 Algorithm

The algorithm consists of these steps:

1. Using configuration subdivision and conservative geometric tests, find hypercubes in configuration space where solutions potentially lie. (For an example see Figures 3.1, 3.2)

2. Using Monte Carlo Optimization to numerically search each of the hypercubes that potentially contain a solution, for specific solutions.

   (a) Define an error metric based on the geometric constraint.

   (b) Sample the space within the hypercube with N samples. (For an example see Figure 3.3)

   (c) If any sample has an error less than the acceptable ϵ, return that sample as the solution.

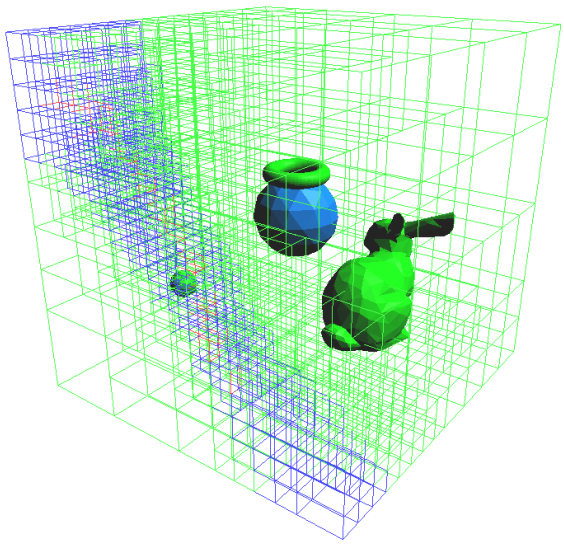

Figure 3.1: Approximate Cell Decomposition. Green are empty cells, blue are potential solution cells, red are actual solution cells.

(d) If all the samples leave the hypercube, then return without a solution (because of conservative testing in Step 1, some hypercubes may not contain solutions).

(e) Else weight each sample according to the error metric.

(f) Re-sample according to weight. (For an example see Figure 3.4)

(g) repeat steps 2c-2f

Note that this approach lends itself very well to parallelization. Step 2 executes multiple times (once for every hypercube result from Step 1), and are all independent executions so that they can potentially execute in parallel. This is beneficial because configuration space subdivision is also easily parallelizable, assuming shared memory between parallel executions.

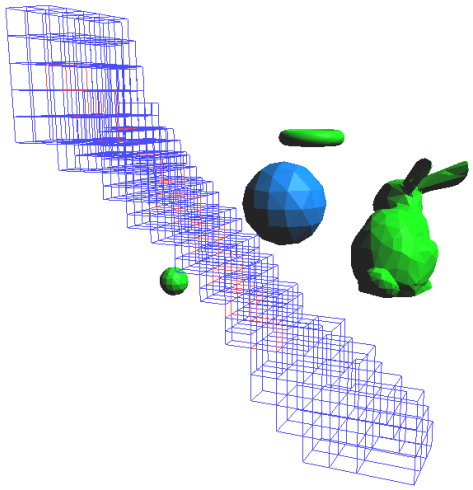

Figure 3.2: Resulting 3-cubes where solutions potentially lie.

3.3 Implementation Details

Here some of the steps of the algorithm are discussed in more detail, as well as implementation details.

3.3.1 Configuration Space Decomposition

(Algorithm Step 1)

Our implementation follows the general cell decomposition approach, which while more completely described in [33, 7, 6], will be reviewed briefly here.

First, a relevant subspace of an n-dimensional parameter space is described by a containing n-dimensional axis-aligned hypercube. This parameter space hypercube is then recursively subdivided into two subspaces or cells, the division occuring on the dimensions with the largest size. At each subdivision step the boundaries are conservatively tested for the possibility of a solution existing within the cell and

subsequent sub-cells. Conservatively testing means that the preference is to err by retaining too many cells, rather than prematurely trimming a cell.

The actual cell testing is more specific to the problem. Essentially, the configuration parameters at the edge of the cell are converted into real world positions. Additionally, the potential range of real world positions are considered (from one hypercube vertex configuration to another hypercube vertex configuration). From these positions and ranges it is determined whether a solution to the constraint could potentially lie within the range of configurations.

If there is no potential for a solution to lie within the cell, subdivision of that cell stops (essentially trimming that area of configuration parameter space from being considered for further searching). If a solution can potentially exist, subdivision continues to the predefined minimum cell size.

In our implementation, a subdivision only results in two subspaces, so that the entire subdivision can be arranged into a binary tree.

See Figures 3.1 and 3.2 for an example.

Figure 3.1 shows the complete decomposition. The green cells are those that were tested and could not possibly contain a solution, the blue cells are those that passed the conservative test and can contain a solution, and the red cells are those numerically searched for a specific solution by the Monte Carlo optimization step and actually contain a solution.

Figure 3.2 shows the same scenario as Figure 3.1, but without the green cells.

3.3.2 Monte Carlo optimization

Error Metric

(Algorithm Step 2a)

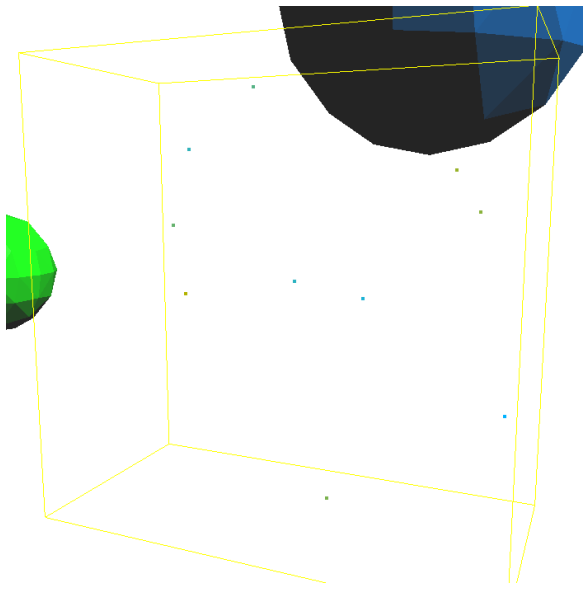

Figure 3.3: Initial sampling of one of the cubes.

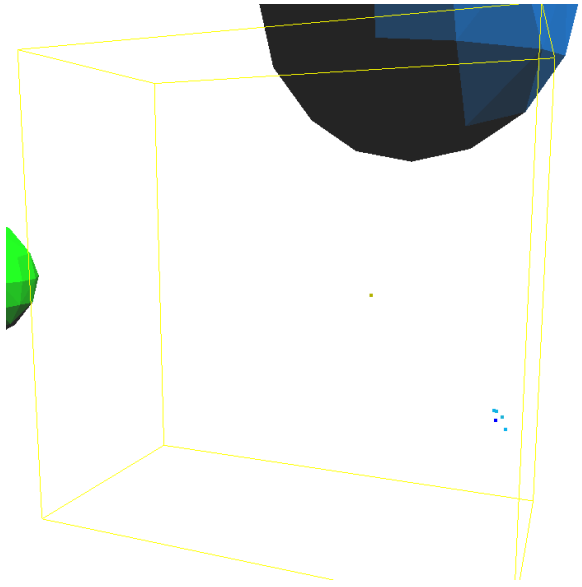

Figure 3.4: After a few of iterations the samples cluster around an answer.

An error metric is defined that maps a point in configuration space to a scalar value indicating how well it satisfies the geometric constraint. In all of our examples, configuration space is always a manifold in $\Re^m$, so that the error metric can be described by a mapping function $f : \Re^m \Rightarrow \Re$, where m is the dimension of configuration space.

In our examples the error function is defined by converting from configuration space to robot position in workspace, and then measuring error of that configuration to the defined constraint. The error is based on the constraint problem being solved, thus guiding the samples toward the constraint solution.

The error metric is used to customize the general approach to individual problems. The error metric acts as an *artificial gradient*, guiding the sampling procedure towards the solution of the constraint. Another viewpoint, is that the error function is being optimized by Monte Carlo optimization.

As an example, consider the case of finding the generalized Voronoi diagram of three co-planar points in 3-space. The constraint can be described in a straight forward way - find all positions equidistant to the obstacles. Consider the diagram in Figure 3.5, the error of point p (red square) could be described by the difference between the distances to each of the three Voronoi points (center points of the 3 spheres). More specifically, if the center point of each sphere is $\vec{c_1}, \vec{c_2}, \vec{c_3}$, then the Euclidean distance ($d(x, y) = \sqrt{\sum_i (y_i - x_i)^2}$) from any point on that plane, $\vec{p}$, to each sphere center, would be

$$d_i = d(\vec{p}, \vec{c_i}). \tag{3.17}$$

The constraint could be defined as

$$\sum_i |d_i - d_{i+1}| < \epsilon, \tag{3.18}$$

where ϵ is the acceptable error.

We use the multiplicative inverse of the error as the weight function, to guide re-sampling towards the solution. To give an idea of how this weight behaves, the weight function, or multiplicative inverse error function,

$$\frac{1}{\sum_i |d_i - d_{i+1}|}, \tag{3.19}$$

is plotted as a height-field shown in Figure 3.6. The (x, y) position on the green graph is the position on the plane (created by the three sphere centers), and the height of the graph is the weight, or inverse of the error. As a potential solution approaches the center of the three Voronoi points, the error approaches 0, so that the inverse becomes very large (for purposes of visualizing, the graph was cut off at a large value, which is why the top is visible).

By continually re-sampling points whose weight is the greatest, samples are continually found with smaller error, until the desired error threshold is obtained.

Other examples of potential constraint error functions could be considered. For example: the constraint problem involved in finding configurations where the robot collides with the obstacles. The error metric from that constraint could be simply the distance from the robot to the closest obstacle.

$$d(\vec{p}, O) < \epsilon, \tag{3.20}$$

where O is an obstacle. The solution is approached as that value goes to zero by the samples. Similarly the weight function will increase as the configuration nears an

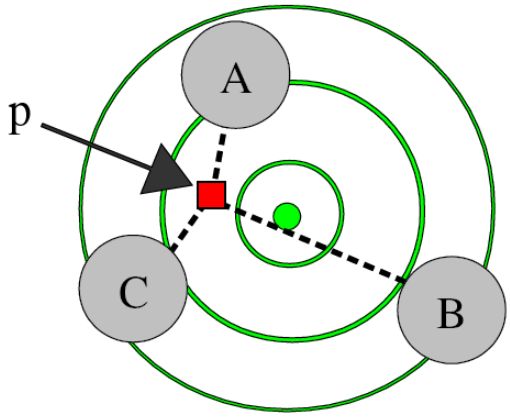

Figure 3.5: Overhead diagram of the error metric.

obstacle. A variety of other constraint problems can be described in a similar fashion.

Initial Sampling

(Algorithm Step 2b)

The initial sampling is an important part of Monte Carlo optimization. The initial sampling affects both the quality of the solution and the speed of convergence on a solution. If the initial sampling is not sufficiently covering the target space, it is not always easy to migrate towards the solution. If more iteration steps are required to reach the solution, due to not having an initial sample near the solution, it results in a longer runtime.

After running a number of experiments to determine what type of initial sampling would suit our system best - between uniform random sampling, stratified sampling and Halton sampling - we decided to use stratified sampling (also known as jittered sampling). The details of the experiment are found in Section 4.2.

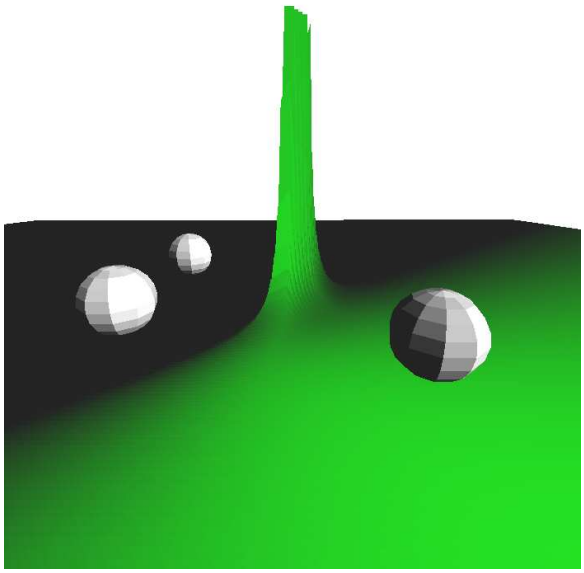

Figure 3.6: The inverse error metric height-field for finding the center point of the three sphere centers.

Determining if a solution exists

(Algorithm Step 2d)

Using conservative testing during the configuration subdivision means that some of the cells will not contain the solutions to the geometric constraint. This means that if the solution lies outside the cell the solution points will migrate outside the cell. The cell is trimmed when this occurs (since that solution will be found in its own cell).

The test for a sample to be outside the cell is done by finding the distance from each sample to the center of the cell, and then checking that this distance is less than the size of the cell.

One interesting question is how many of the samples are allowed to leave the hypercube before concluding that there is no result within the hypercube. It may seem tempting to terminate the search early when a majority of the the points leave

the hypercube. Through experimentation we have found that, though terminating early on this kind of condition can save computation time, it is safer to require all samples to leave the cell before terminating the search. If the solution lies near the boundary of the hypercube, a majority of the samples can leave the hypercube for a time, and then re-enter. In fact we have found that a better approach is to give a small cushion or buffer zone outside the hypercube, which the samples have to pass before being considered outside the hypercube.

In the resulting approach, each sample is checked to see if it lies without the hypercube (and potential buffer zone) and then terminate the search only when all samples have completely left the hypercube, a maximum iteration count has been reached, or the solution is found.

Re-sampling

(Algorithm Steps 2e and 2f)

The re-sampling portion of the algorithm was designed to favor re-sampling of points with smaller error (better weight). To this end, the weight of each point is assigned to be the multiplicative inverse of its error, which is later normalized against all the other samples.

Once the samples are weighted appropriately, the space is re-sampled using the weights. The weighting system is designed so that the samples with higher weights (which corresponds to lower errors), are selected to be re-sampled more often.

An example distribution over 10 sample points is shown in Figure 3.7. That Figure is a number line showing the cumulative weight of each sample, where the sum of all weights is equal to 1.0. The weight of the an individual sample is represented by the distance between itself and the previous sample. By picking a uniformly distributed

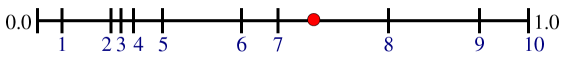

Figure 3.7: An example of cumulative weight values for 10 samples. Each vertical line represents the cumulative weight of the samples up to that weight. The weight of each sample is the distance between itself and the previous weight.

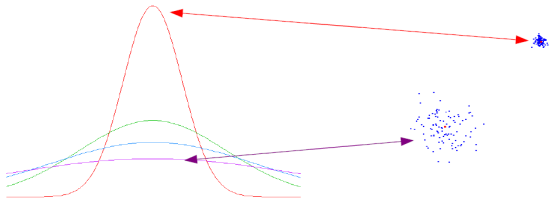

Figure 3.8: Example Gaussian curves with standard deviation or $\sigma = 1, 0.7, 0.5, 0.2$ from bottom to top. Changing the σ in the Gaussian spread affects the convergence rate of the Monte Carlo optimization step. The lower the σ, the closer new samples will be to the original sample overall. The higher the σ, the further they will be from the original sample.

random number between 0.0 and 1.0, and then selecting the sample it "lands on", samples with larger weights are selected more often. For example, a random sample where the red dot is positioned would select sample 8.

This approach results in the samples being selected to be re-sampled according to the inverse error metric. The samples with lower errors will have a better chance of being re-sampled and replace other samples that have a higher error. The samples with a higher error (and therefore, lower weight) will fall out of the sample set over multiple iterations.

Re-sampling is done by adding a Gaussian distributed random vector to the original point. If the new sample has better weight (lower error) than the original, it replaces it. Otherwise, the original is kept.

The convergence rate can be modified by varying the standard deviation, or σ, of the Gaussian distribution. As depicted in Figure 3.8, a larger σ leads to larger

Table 3.1: Standard deviation depends on the error size.

Error Range	Stand. Dev.
$(2.0, \infty)$	10×10^{-1}
$(1.0, 2.0]$	10×10^{-2}
$(0.1, 1.0]$	10×10^{-3}
$(-\infty, 0.1]$	10×10^{-4}

random vectors, which allows more exploration. A smaller σ leads to smaller random vectors, which helps converge faster near the solution (smaller steps do not over step the solution as easily). Our approach is to change the σ according to the error of the sample, for smaller errors, a smaller σ is selected - this allows more exploration more when far from the solution, and faster convergence when near the solution.

In our implementation the standard deviation would step down at specific error intervals. The standard deviation for each error range is listed in Table 3.1.

Chapter 4

Results

4.1 Overview

Through experimental results, we show that by using Monte Carlo optimization for the final solution step in configuration space subdivision, we can solve to a higher precision, as well as solve higher dimensional problems than subdivision by itself.

Using subdivision by itself to find a solution can require an exponential amount of memory, and comparable increases in time. By extending subdivision with Monte Carlo optimization, subdivision terminates at a lower precision, and the Monte Carlo approach is used to "refine" the low precision solution to a more precise solution. This requires much less memory and we have shown through experimentation that solutions we were unable to solve with configuration space subdivision alone can be solved with this combined approach.

In order to recover a surface or line solution from the point solutions found, the configuration subdivision hierarchy must be stored. This is the cause of most of the memory usage in configuration space subdivision. If only point solutions were of interest in the end (no reconstruction of surfaces or curves necessary), the method

could be adapted to trim the tree structure as it went, moderating the memory usage. Note that in the area of path planning instead of reconstructing surfaces, the tree organization would aid in constructing a smooth path from an initial position to a final position.

This section is broken into three major categories. First, the results to our initial sampling experiment are presented. Afterwards, the results of using the complete algorithm on multiple different problems are presented in two sections "General Geometric Constraint Problems" and "Robot Path Planning Problems".

The results to a few general geometric constraint problems are shown in order to give an idea that the approach can be used in situations other than robot path planning. This also begins to give an idea of the speed and memory advantages of using Monte Carlo optimization with configuration space subdivision.

Next, the results to various robot path planning problems are presented. The main results of our experimentation is that extending configuration space subdivision with Monte Carlo optimization allows the solution of path planning problems of higher precision, and constraint problems in higher dimension and precision than subdivision by itself.

4.2 Initial Sampling Experimentation

A very important part of Monte Carlo optimization is the initial sampling. The initial sampling affects both the quality of the solution and the speed of convergence on a solution. If the initial sampling does not sufficiently cover the target space, it is not always easy to migrate toward the solution. If more iteration steps are required to reach the solution, due to not having an initial sample near the solution, a longer runtime results or the solution may not be found at all.

To determine what sort of initial sampling would be most appropriate, we ran our implementation of Monte Carlo optimization several times on a simple constraint problem adapted for this purpose. By evaluating different values for the number of samples and the type of initial sampling distribution with respect to a desired precision, one approach was found that performed better than the others.

4.2.1 Experiment Setup

The simplified problem is setup to consider different situations that the Monte Carlo optimization will encounter, while extending our implementation of configuration space decomposition. In our decomposition system, a subdivision causes the largest dimension to divide in half, so that at any time some of the dimensions of the configuration hypercube may be half of the other dimensions. To try to recreate this in an experimental situation, two n-dimensional hypercubes are created, the first hypercube, H_1, is centered about a point $\vec{P} = [p_1, \ldots, p_n]^T \in \Re^n$, with distance to each face $\vec{L} = [l_1, \ldots, l_n]^T \in \Re^n$, where $l_i = k, \forall i \in [1, n]$, k a constant. A second n-dimensional hypercube, H_2, is centered about a point $\vec{Q} = [q_1, \ldots, q_n]^T \in \Re^n$, with distance to each of the faces $\vec{M} = [m_1, \ldots, m_{n/2-1}, \frac{1}{2}m_{n/2}, \frac{1}{2}m_{n/2+1}, \ldots, \frac{1}{2}m_n]^T \in \Re^n$, where $m_i = k, \forall i \in [1, n]$, k a constant. In other words, while the first hypercube has all faces equidistant from its center point, the second has the first half set to the same constant, while the second half are set to half of that constant. In the results below we set $\vec{P}, \vec{Q}$ to be the origin, and $l_i = m_i = 5$.

The Monte Carlo optimization step may run into solutions that are difficult to find. To try to capture this occurence, we create three optimization problems to solve. The first problem is to find the center of the cube; second, to find a point very close to the corner of the cube (which can be difficult to find without leaving the hypercube);

and finally, to find a random point in the cube - this will represent situations where the solution may not be in the corner or in the center, but somewhere else within the hypercube. The constraint for each of these problems is that the solution be equal to the target point, so the error function will simply be the distance from the specified point. More specifically, for each problem, where the target point is $\vec{T} = [t_1, \ldots, t_n]^T \in \Re^n$, and the current guess is $\vec{G} = [g_1, \ldots, g_n]^T \in \Re^n$, $\text{error}(\vec{G}) = \sqrt{\sum_{i=1}^{n} (t_i - g_i)^2}$.

Varying numbers of samples, precision, and type of initial sampling are used to try to solve each of these problems.

The goal of the experiment is to find which of three initial sampling techniques will perform the best with the smallest number of samples (since fewer samples require less compute time in the end). It would also be beneficial to gain some intuition as to whether one type of sampling technique works best for this type of problem. The three investigated here are

(1) Halton Point Sampling

Halton point sampling (closely related to Hammersley points) utilizes a deterministic sequence generator to create what can appear to be a sequence of random points, but in reality is a well spaced, deterministic sequence of points. The basic idea of the sequence generator can be understood from a simple example. A positive integer k has a binary representation

$$k = b_0 b_1 \ldots b_n = b_0(2^0) + b_1(2^1) + \ldots + b_s(2^s). \tag{4.1}$$

If the components of the binary representation are reversed, including the binary point, and then re-interpreted as a decimal value, the result $k' = b_s b_{s-1} \ldots b_0$

has the value

$$k' = b_s(2^{-1}) + b_{s-1}(2^{-2}) \ldots + b_0(2^{-(s+1)}). \tag{4.2}$$

This operation can be refered to as $G(\cdot)_2$, so that $k' = G(k)_2$.

Because any positive number k can be decomposed and represented by any prime number p in the same way as by 2, the same operation can be defined using any prime base p. This more general operation can be referred to as $G(\cdot)_p$.

It is important to note that if the domain of $G(\cdot)_p$ is all positive integers, the range will remain $(0, 1)$.

Finally, given a point $\vec{R} = [r_1, r_2, \ldots, r_n]^T$ a function can be defined

$$H(\vec{R}) = [G(r_1)_{p1}, G(r_2)_{p2}, \ldots, G(r_n)_{pn}]^T \tag{4.3}$$

such that $p1, p2, \ldots, pn$ are distinct prime numbers.

To generate a Halton point set, a sequence of points whose components are all positive integers is each in turn operated on by $H(\cdot)$, resulting in a set of points within the range $(0, 1)^n$, the n-dimensional unit hyper-cube. In our implementation $H(\cdot)$ is used on the set of S n-dimensional points, $[1, \ldots, 1]^T, [2, \ldots, 2]^T, \ldots, [S, \ldots, S]^T$. Resulting in the Halton points $H([1, \ldots, 1]^T), H([2, \ldots, 2]^T), \ldots, H([S, \ldots, S]^T)$.

For a more detailed discussion and implementation example see [2].

Our motivation for trying a Halton set was the idea that an even spread of points throughout space would aid in finding a potential solution faster.

(2) Stratified Sampling

Stratified sampling in general statistical terms refers to sampling by first arranging the sampling population into relatively homogeneous subgroups, called strata, before sampling. This is done in hopes to gain representation from each of the different strata in the final sampling.

Uniform regular stratified sampling in computer graphics (also commonly called jittered sampling), and related fields - including our approach - follows this same idea. The sampling population is the entire space, and the strata or subgroups into which it is divided are distinct uniform-size subspaces.

A simple example illustrates. Assume the sampling is of a 2-dimensional space with $S = s^2$ samples. One approach to stratified sampling is to divide the space into s^2 equally sized subregions. Then s^2 samples are selected by finding a random point within each of the subregions. See [26] for more details.

To extend this approach to n-dimensional space, s^n uniform subregions are created. Then s^n samples are selected by finding a uniformly random point within each of the s^n subregions.

In our implementation, given a number of samples to be taken S, set $S' = \lfloor S^{1/n} \rfloor$. Then if $b = S - (S')^n \neq 0$, the remaining b samples are taken as uniform random variables over the whole space.

This approach was also implemented and tested in hopes that a more equal spread of initial sampling would aid in finding the answer faster, and with less samples.

(3) Uniform Random Distribution Sampling

This approach is the most straightforwad of the three. To select S samples from an n dimensional space, S n-dimensional points are created where each

component is a uniformaly distributed random number. More specifically,

$$\vec{P} = [p_1, \ldots, p_n]^T \tag{4.4}$$

where $p_1, \ldots, p_n$ are each uniformly distributed random numbers.

Uniform random sampling is taken as the base case. The experiment was created to determine whether a different initial distribution of points would be more effective.

While we find no clear winner overall, for our target numbers of samples (from 10 to 30), we conclude from the experiment that (2) Stratified Sampling has the best results, and use that type of sampling in our solutions where possible. This agrees somewhat with our initial intuition - that a more equally spread sampling would perform better than a purely uniformly random distribution. Although because Halton sampling is a little more uniform (deterministic, rather than random in its distribution), we had originally thought that it would fare the best.

The results are presented next, accompanied by a discussion of how those results influence our decisions on what to use in the Monte Carlo optimization.

4.2.2 Experiment Results

Two main criteria are examined from each type of sampling.

(A) Percent of successfully finding the result

(number of successes/number of trials)

(B) Average number of iterations for a successful solution

(total number of iterations / number of successes)

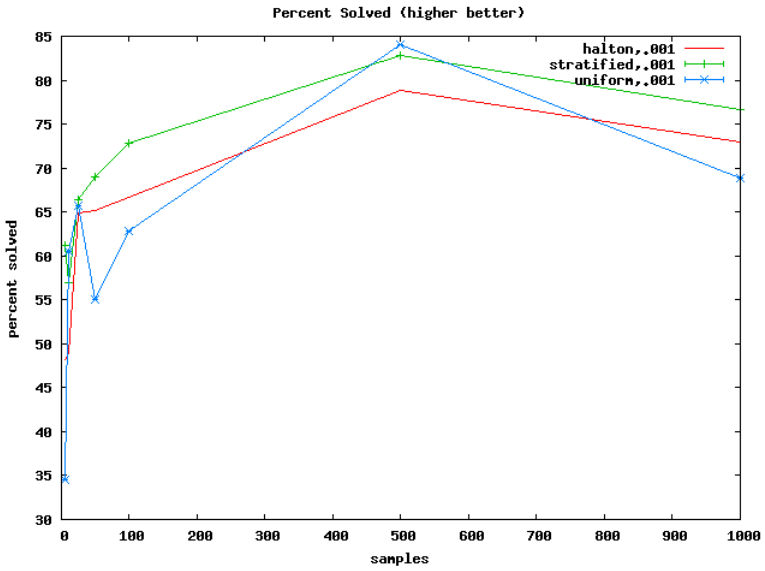

Figure 4.1: Overall percentage of sampling test problems solved correctly at .001 precision.

We maintain that criterion (A) is more important to our approach than (B), because the solution should be found even if it takes a few more iterations on average.

4.2.3 Overall

Looking at the overall results for (A) percentage of successful solutions, shown in Figure 4.1 and Table 4.1, it appears that the Stratified sampling approach, while not always the best, does the best in most of the sample sizes. Similarly overall in (B) the average number of iterations for a successful solution, shown in Figure 4.2 and Table 4.2, it appears stratified sampling uses the least average number of iterations.

4.2.4 Sample Sizes under 100

As stated previously, we are more interested in finding the best approach for solutions using smaller numbers of samples - preferably between 10 and 30 - consequently the

Table 4.1: Overall percentage of sampling test problems solved correctly at .001 precision.

samples	Halton	Stratified	Uniform
5	48.16	61.16	34.50
10	48.83	57.00	60.50
25	64.83	66.33	65.66
50	65.16	69.00	55.00
100	66.66	72.83	62.83
500	78.83	82.83	84.00
1000	73.00	76.66	68.83

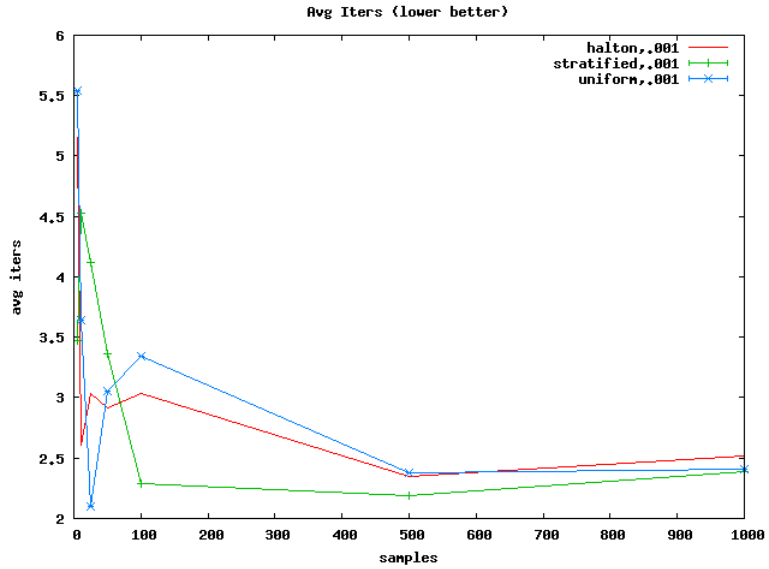

Figure 4.2: Overall average iterations to find a correct solution at .001 precision.

Table 4.2: Overall average iterations to find a correct solution at .001 precision.

samples	Halton	Stratified	Uniform
5	5.15	3.47	5.54
10	2.61	4.52	3.64
25	3.03	4.11	2.09
50	2.91	3.36	3.05
100	3.03	2.28	3.34
500	2.35	2.18	2.37
1000	2.51	2.38	2.40

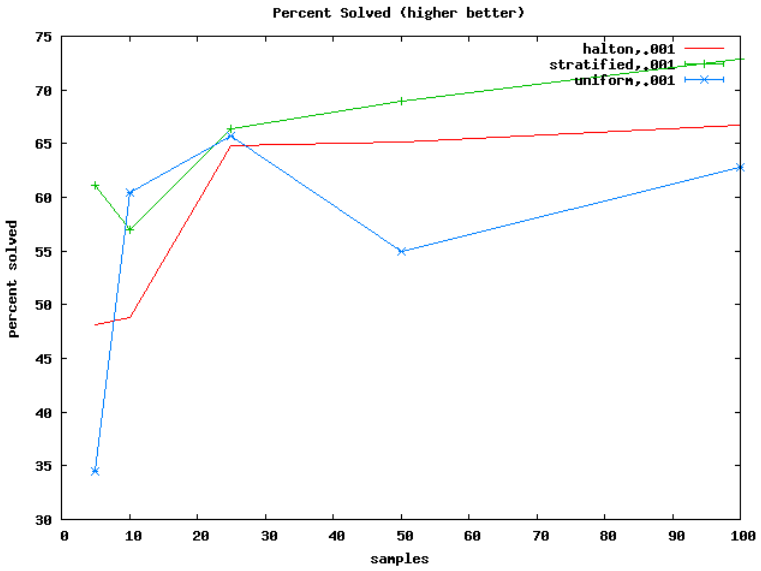

Figure 4.3: Overall percentage of sampling test problems up to 100 samples solved correctly at .001 precision.

results between 0 and 100 samples carry more importance. Shown in Figure 4.3 are the same results for criterion (A) from Figure 4.1, except only showing up to 100 samples. As shown the stratified solution performs the best with any number of samples over about 20. And between 0 and 20 it comes in second. In contrast, Figure 4.4 shows that stratified sampling actually performs the worst in terms of average number of iterations, criterion (B), until a sample size of about 60. While this is not encouraging, we maintain that stratified sampling is still the best choice overall, because as stated previously criterion (A) is more important to our approach.

4.2.5 Sample Size 25

By examining the results for a specific sample size of 25, the selection of stratified sampling is supportable, but not a clear leader in the scoring. As can be seen in Figure 4.5 and Table 4.3, stratified sampling does not fare the best. While it does

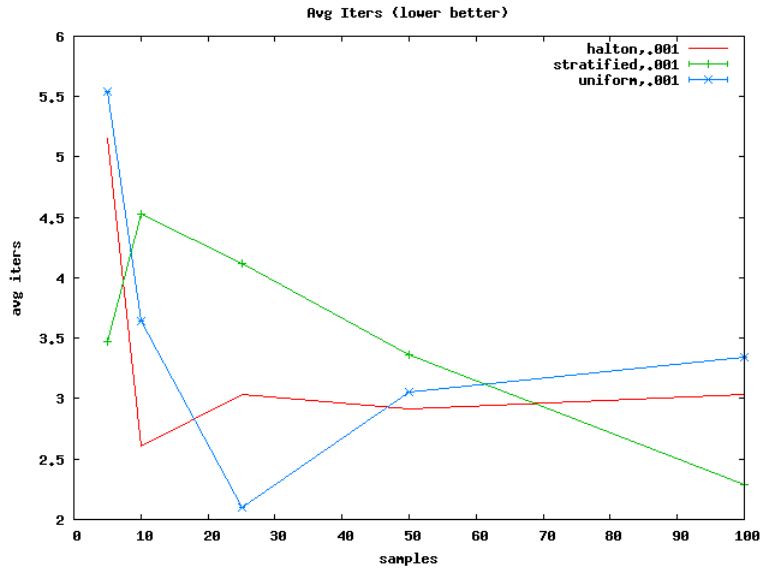

Figure 4.4: Overall average iterations to find a correct solution up to 100 samples at .001 precision.

not score the worst on every dimension, it only becomes the best choice in the 8 dimensional problem. The results reveal that with a sample size of 25, stratified sampling performs averagely compared to the other approaches. However, as shown in Figure 4.6 and Table 4.4, in average iterations, stratified sampling does the best in every dimension.

The 25 sample results differ somewhat from the overall results examined above. While the stratified sampling performed very well in most of the cases for criterion (A), in the 25-sample case it does not do quite as well. For criterion (B) it does better. We maintain the choice of stratified sampling, because overall it shows the most promise, and while the specific 25 sample case reveals some potential problems, neither of the other candidates show as strong a position in all cases considered.

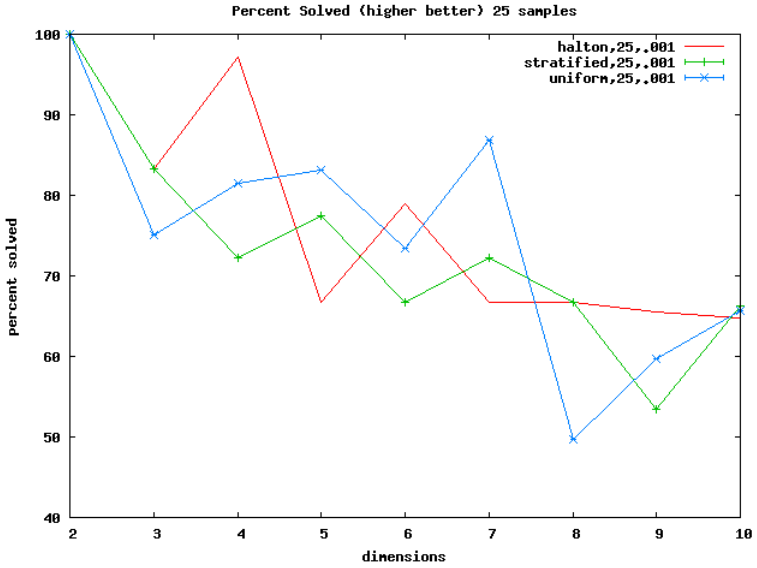

Figure 4.5: Percentage of problems solved correctly with sample size 25 and .001 precision.

Table 4.3: Percentage of problems solved correctly with sample size 25 and .001 precision.

dimension	Halton	Stratified	Uniform
2	100.00	100.00	100.00
3	83.33	83.33	75.00
4	97.16	72.16	81.50
5	66.66	77.50	83.16
6	79.00	66.66	73.50
7	66.66	72.16	86.83
8	66.66	66.66	49.66
9	65.50	53.50	59.66
10	64.83	66.33	65.66

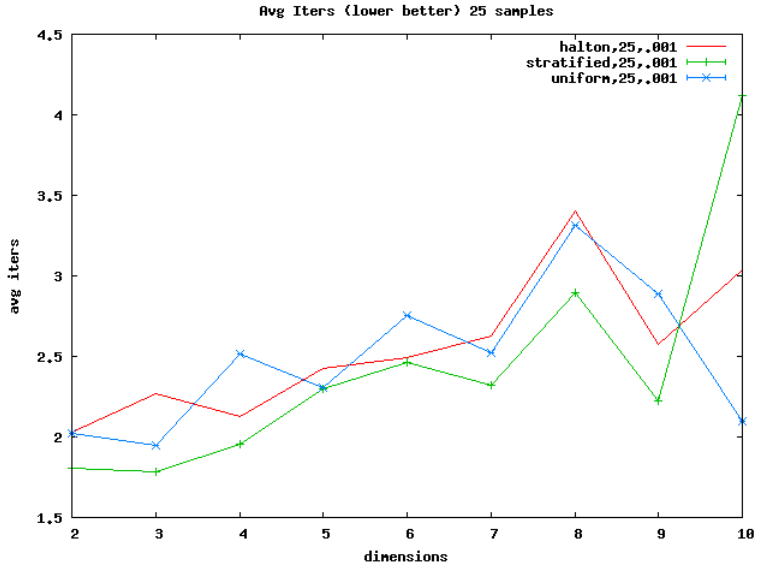

Figure 4.6: Average iterations to find a correct solution with sample size 25 and .001 precision.

Table 4.4: Average iterations to find a correct solution with sample size 25 and .001 precision.

dimension	Halton	Stratified	Uniform
2	2.02	1.80	2.02
3	2.27	1.78	1.94
4	2.12	1.95	2.51
5	2.42	2.29	2.30
6	2.48	2.46	2.75
7	2.63	2.32	2.52
8	3.40	2.89	3.31
9	2.57	2.22	2.88
10	3.03	4.11	2.09

4.3 General Geometric Constraints Problems

The constraint problems presented mainly center around finding the Generalized Voronoi diagram of multiple objects. First, the cell boundary surfaces are found, and then through over-constraint, the cell edge curves, and finally the Voronoi vertices are found.

4.3.1 Bisector Surfaces

The trimmed bisector surfaces of four models are found at once, shown in Figures 4.7 and 4.8. This solution is the same as the generalized Voronoi diagram of the four objects. In fact, these surfaces are the cell boundaries of the generalized Voronoi diagram. The constraint can be described as equidistantance to the two closest objects, and the solution as the collection of points that satisfy the constraint.

More precisely, a solution is saught for all points, $\vec{P} = [p_1, p_2, p_3]^T$, such that

$$d(\vec{P}, \vec{B^1}) = d(\vec{P}, \vec{B_i^2}) \tag{4.5}$$

Where $d(\vec{X}, \vec{Y}) = \sqrt{\sum_i (x_i - y_i)^2}$ or the Euclidean distance, $\vec{B^1} = [b_1^1, b_2^1, b_3^1]^T$ is the closest point on one of the two nearest models, and $\vec{B^2} = [b_1^2, b_2^2, b_3^2]^T$ is the closest point on the other of the two nearest models.

The solution to this constraint using configuration space subdivision by itself is shown in Figure 4.8, and using Monte Carlo optimization in Figure 4.7. The runtime details are listed in Table 4.5. The method of using configuration space subdivision by itself is indicated by "CS", and using Monte Carlo optimization to extend the configuration space subdivision is indicated by "MC". Both solutions are done using an error threshold of 10×10^{-2}. Of note is that using the Monte Carlo optimization

Table 4.5: Timing and memory usage comparison for finding the bisector surfaces of 4 objects shown in Figures 4.7 and 4.8.

method	time(sec)	memory	CS precision	MC precision
CS	76.363	133 MB	10×10^{-2}	N/A
MC	2.125	23 MB	10×10^{-1}	10×10^{-2}

approach the constraint is solved in 2% of the time and using less than 20% of the space it takes without it. In contrast, consider Figure 4.10, which is the solution with the same configuration subdivision precision of 10×10^{-1}, but without the solution improvement by Monte Carlo optimization. The error is large enough that the surfaces more resemble volumes. Figure 4.8 shows the subdivision-only solution, but with error equal to the 10×10^{-2} - the precision of the MC optimization solution in Figure 4.7. This solution almost looks like a solid surface, because the subdivision cells become as small as the specified error tolerance leading to solution points which are close enough together that they resemble a surface.

Monte Carlo optimization is able to solve the problem much faster and use much less memory because using the MC optimization to reach the solution configuration space subdivision can terminate much earlier. Figure 4.9 shows the solution using Monte Carlo optimization with the configuration space subdivision cells visible. In essence, a solution is found with the same precision but with a more sparse distribution of solution points. This sparsity can be adjusted as needed. Interpolating between the solutions found can fill in the gaps.

While solutions here are shown as collections of individual points, the surfaces or curves they form can be reconstructed from these points.

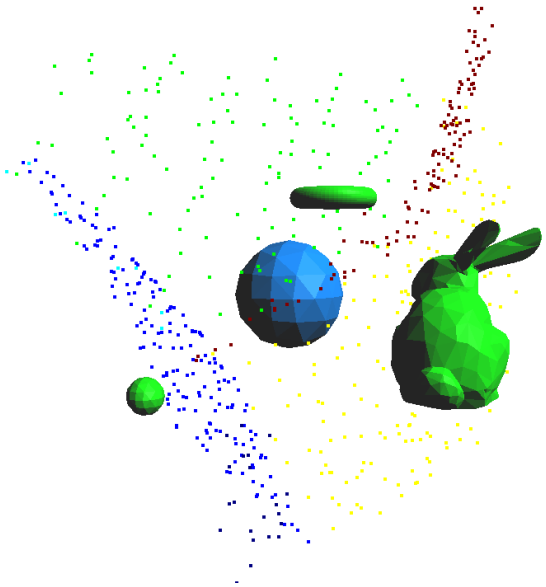

Figure 4.7: Bisector surfaces of four models, found using Monte Carlo optimization.

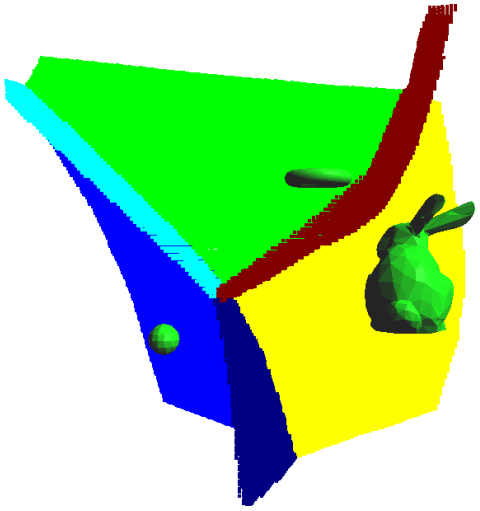

Figure 4.8: Bisector surfaces of four models, using only configuration space subdivision. The points are close enough that they appear to be a solid surface.

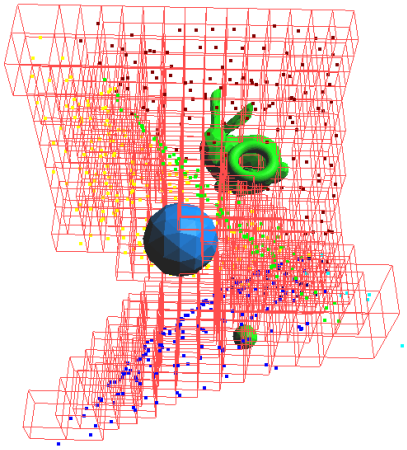

Figure 4.9: Same as Figure 4.7, detailing the size of volumes from which the solutions are found.

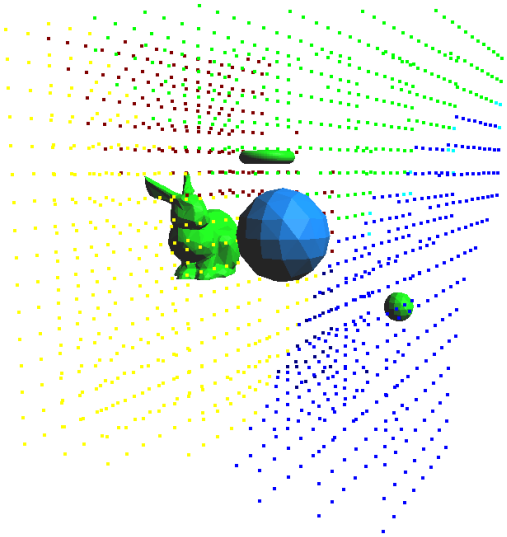

Figure 4.10: In contrast to Figure 4.7, consider the same solution at precision 10×10^{-1}, but without the solution improvement by Monte Carlo optimization. Because the error is so large, the surfaces look more like volumes.

4.3.2 Intersection of three bisectors

The system is also able to solve for the curves representing the Voronoi cell edges or where two or more cell boundaries intersect. This is shown in Figure 4.11.

These edges can be found for all four models at once, as shown in Figures 4.12 and 4.13.

This constraint can be described in a manner similar to Equation 4.5. Again, a solution is saught for all points $\vec{P}$ such that

$$d(\vec{P}, \vec{B^1}) = d(\vec{P}, \vec{B^2}) = d(\vec{P}, \vec{B^3}) \tag{4.6}$$

Where $\vec{B^3} = [b_1^3, b_2^3, b_3^3]^T$ is the closest point on one of the three closest models, and b_i^1 and b_i^2 are defined similarly as in Equation 4.5 and where $\vec{B^1}, \vec{B^2}, \vec{B^3}$ are all on different models.

This problem adds a constraint to the previous problem, so that the constraint is now equidistantance to the closest three obstacles.

This problem is solved using subdivision by itself in Figure 4.13, and then with MC optimization in Figure 4.12. The timing and memory usage for each solution are presented in Table 4.6. As the table results indicate, a solution to the problem using MC optimization is found in 1.86% the time without it. The table also shows the much smaller memory footprint, only requiring 20 MB instead of 79 MB.

As explained above, these gains are because MC optimization allows termination of the subdivision process earlier.

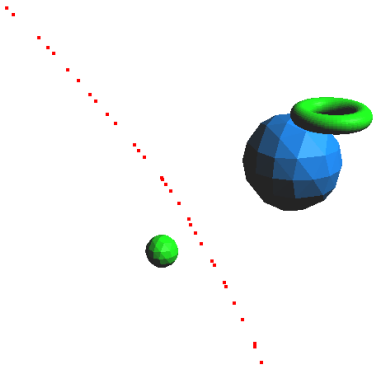

Figure 4.11: Intersection of three bisector surfaces, or bisector edges.

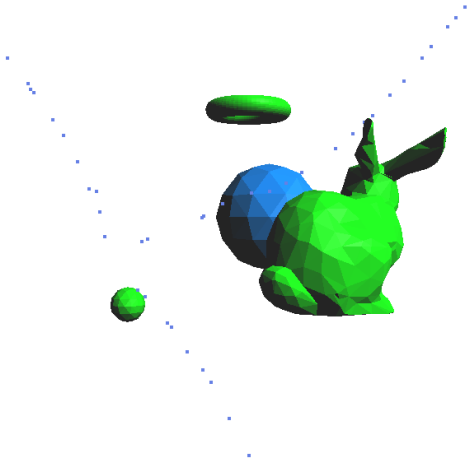

Figure 4.12: Bisector curves for four models, found using Monte Carlo optimization.

Table 4.6: Timing and memory usage comparison for finding the bisector curves of 4 objects shown in Figures 4.12 and 4.13.

method	time(sec)	memory	CS precision	MC precision
CS	54.344	79 MB	10×10^{-3}	N/A
MC	1.011	20 MB	10×10^{-1}	10×10^{-3}

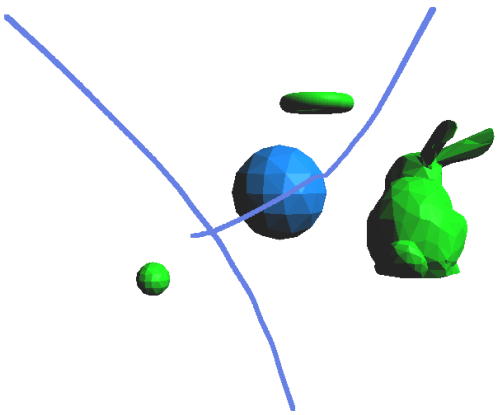

Figure 4.13: Bisector curves for four models using only configuration space subdivision. The individual points are so close together that they appear to be a solid line.

4.3.3 Intersection of six bisectors

From the previous examples and images, it can be seen that there is a single point at which all these curves intersect, which is also the point where all six bisecting surfaces intersect. It would also be considered a vertex in the generalized Voronoi diagram.

The solution is shown in Figure 4.14. This constraint would be described as equidistantance to the four closest objects, thus increasing the number of constraints. More precisely a solution is saught for the point $\vec{P}$, such that

$$d(\vec{P}, \vec{B^1}) = d(\vec{P}, \vec{B^2}) = d(\vec{P}, \vec{B^3}) = d(\vec{P}, \vec{B^4}) \tag{4.7}$$

Where the b_i^1, b_i^2, b_i^3 components are defined the same as Equations 4.5,4.6, and $\vec{B^4} = [b_1^4, b_2^4, b_3^4]^T$ is the closest point on one of the 4 closest models, where $\vec{B^1}, \vec{B^2}, \vec{B^3}, \vec{B^4}$ are all on different models.

The constraint is solved using both methods again. The results for using Monte Carlo optimization are shown in Figure 4.14, while the results for subdivision by itself are in Figure 4.15. The point in Figure 4.15 may look a little larger than normal, this is because up close a collection of configuration volumes that satisfy this constraint within an error ϵ are visible, as shown in Figure 4.16.

The runtime and memory usage are shown in Table 4.7. Again, MC optimization is able to terminate subdivision earlier and solve to a point solution. Without the MC step, subdivision must continue further. The difference is not as drastic as in other cases, but the runtime using MC optimization is still only 20.4% of the runtime without it - which is still a substantial gain. An additional column is shown displaying only the data portion of memory usage. Because the memory usage difference is so small, this closer look is needed to note any difference. The similarity in memory usage is largely due to the minimal configuration space subdivision in both cases,

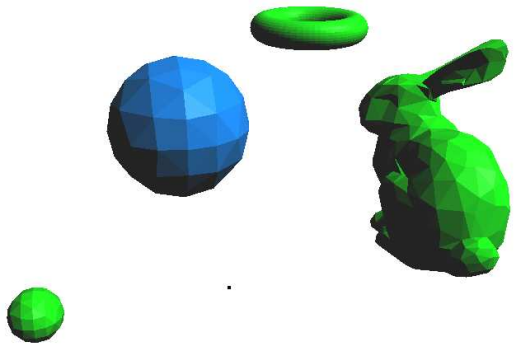

Figure 4.14: Intersection point of six bisectors solved using MC optimization.

Table 4.7: Timing and memory usage comparison for finding the intersection point of all the bisector surfaces, shown in Figures 4.14 and 4.15.

method	time(sec)	memory	data only mem	CS precision	MC precision
CS	0.299	17 MB	8664 KB	10×10^{-3}	N/A
MC	0.061	17 MB	8660 KB	10×10^{-1}	10×10^{-3}

caused by the over-constraint of the problem. Potential areas are "trimmed" from the final solution early because of the strict constraints on the solution. Figure 4.17 shows all the cells in the configuration space subdivision only case, to emphasize that most cells are trimmed (green) at a very large size. This causes a fast runtime for both cases, as well as a very small memory overhead.

4.3.4 Over Constraint

Sections 4.3.2 and 4.3.3 show that to reduce the size of the solution, a constraint can be added to the problem being solved. In order to move from a surface solution to a

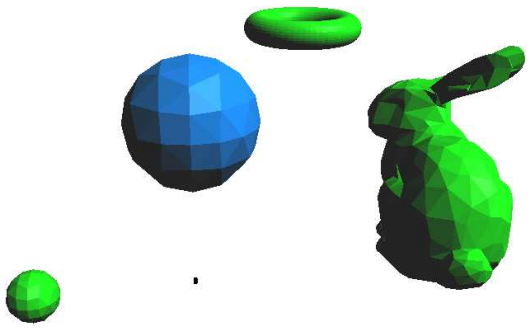

Figure 4.15: Intersection point of six bisectors solved only using configuration space subdivision.

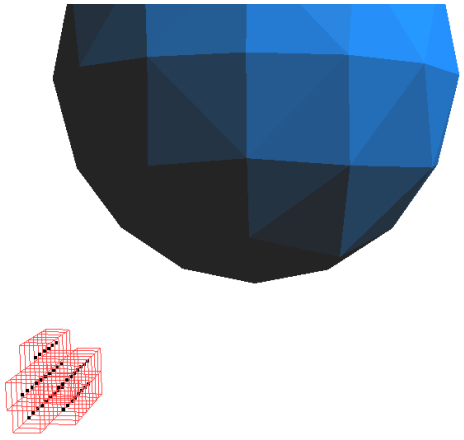

Figure 4.16: Close up view of the solution in Figure 4.15 shows that there is actually a collection of solutions.

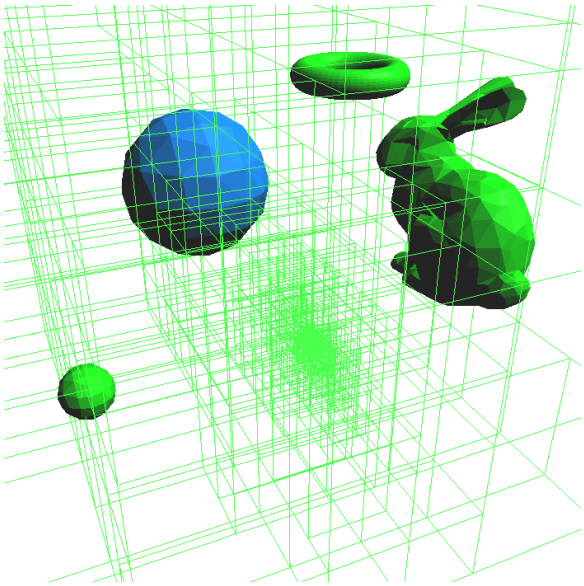

Figure 4.17: The same solution as in Figure 4.15, but with all cells shown depicting how the majority of cells are trimmed early (green) due to the very tight constraint for this problem. This results in a fast runtime and low memory usage, even for the subdivision-only solution.

curve solution, one constraint

$$d(\vec{P}, \text{obj1}) = d(\vec{P}, \text{obj2}), \tag{4.8}$$

was increased to two constraints,

$$d(\vec{P}, \text{obj1}) = d(\vec{P}, \text{obj2}) \tag{4.9}$$

and

$$d(\vec{P}, \text{obj2}) = d(\vec{P}, \text{obj3}) \tag{4.10}$$

Likewise in order to move from a curve solution to a single point solution, two constraints were increased to three constraints.

By adding constraints to the problem, also termed over-constraining the solution,

the number of points that need to be found for the solution is reduced, which is desirable in higher dimensional problems where solutions become hyper-surfaces.

More generally, by over-constraining some problems the most important points, or points of interest, of the solution can be found. Having the points of interest of the final solution can give an idea of the complete solution, and may possibly be used for the final solution. We take this position with the solution to the higher dimensional path planning problems as well - by over-constraining to find important points which may be used to give an idea of the complete solution.

4.4 Robot Path Planning Problems

4.4.1 Generalized Voronoi Diagram

This section discusses one approach to robot path planning and how to solve it using configuration space subdivision, and then improves that solution using Monte Carlo optimization.

The ultimate goal of path planning is to find a path through workspace that avoids all obstacles, from some beginning point to some ending point. One approach to this is to construct the Generalized Voronoi Diagram of the entire configuration space, and then by following the Voronoi cell edges, the robot will be at a position with the same distance between it and the closest n obstacles.

In essence, if all the configurations of the robot, which place it in the generalized Voronoi diagram of the configuration space, can be found they can be used to plan from an initial starting point to the end point. This approach assumes the initial and end points are either in the Voronoi diagram or can be connected to it trivialy.

Specific Setup

The path planning problem consists of using a translating (in 2 dimensions) rotating (on one axis) robot, resulting in a 3 dimensional configuration space. There are 3 obstacles placed in the workspace, which need to be planned around.

First, the generalized Voronoi diagram of the configuration space is found. Once the configuration space Voronoi diagram is found, those results can be used to plan a path through the workspace.

Results

The generalized Voronoi diagram is found using configuration space subdivision by itself, to the indicated precision. The result is shown in Figure 4.18, and the runtime statistics are in Table 4.8.

In order to reach the desired precision, the configuration space subdivision must subdivide until the individual hypercubes are no larger than the precisions specified (in this way it guarantees that the solution is to that precision). Because each level of the configuration subdivision is stored, the memory footprint can also increase dramatically (essentially storing a very large binary tree). The specified precision requires 1.6 GB of memory.

By using Monte Carlo optimization, the subdivision is stopped at a more reasonable precision, and then use Monte Carlo optimization to find a more precise answer. The first pair of results listed (one CS and one MC) show a faster run time using MC optimization, as well as a smaller memory footprint - see Table 4.8. In the second pair of results, MC optimization takes a little longer than subdivision, but takes much less memory to compute. The result of the MC optimization solution is shown in Figure 4.19. These results are more important if the problem needs to be solved on a computer with less than 1 GB of memory. Configuration space subdivision by itself ran out of memory at a precision of 10×10^{-3}, this is represented in the table

as taking an infinite amount of time and memory.

Additionally, the last pair of results show another advantage of using MC optimization, to find a solution with an error less than 10×10^{-7}. Using only configuration space subdivision, it would have been very difficult to find a solution to the problem with that precision.

Figures 4.18 and 4.19 are actually the projections of the generalized Voronoi diagram in 3-dimensional configuration space to the 2-dimensional workspace. This is why the robot arms are not aligned completely with the Voronoi cell boundaries - they would in configuration space, but as a projection of that solution into the 2-dimensional workspace they appear differently. Additionally, because the error margin is only 10×10^{-2} (in the configuration space only case), this allows for some visual error showing in Figure 4.18. The positions shown are only a selection of all solutions found, in order to make the individual robots more visible.

Once the decomposition is complete and the configurations of the robot that make up the Voronoi diagram are found, an actual robot path can be found through these solutions. The result is shown in Figure 4.20. This path was created by constructing a graph out of the solution cells, and then using Dijktra's algorithm to find the shortest path between the inital and final position. This results in a path that is always on the Voronoi diagram, but that is optimized as the shortest path. Variations on this approach could be taken, for example weighting the edges of the graph by their distance from the obstacles in order to find the path furthest from all obstacles (the "safest" path).

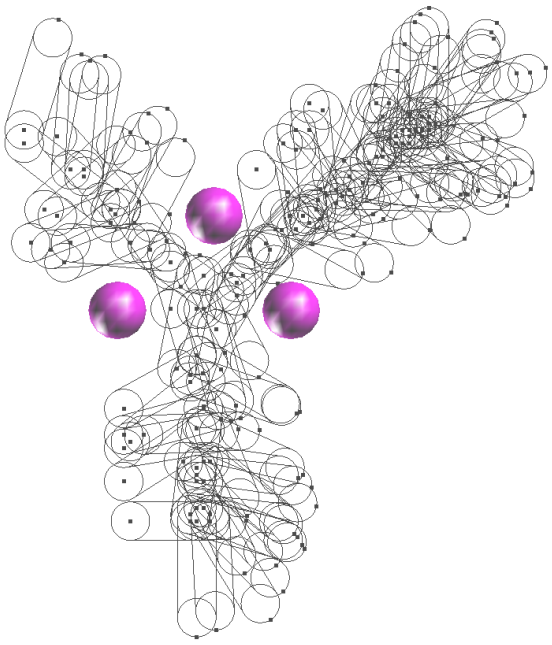

Figure 4.18: Robot drawn on the generalized Voronoi diagram, done with configuration space subdivision by itself.

Table 4.8: Runtime statistics for finding the generalized Voronoi diagram.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	246.17	1.6 GB	10×10^{-2}	N/A	1123058
MC	138.09	722 MB	20×10^{-2}	10×10^{-2}	271568
CS	243.15	1.6 GB	90×10^{-3}	N/A	1123058
MC	348.23	248 MB	80×10^{-2}	90×10^{-3}	10359
CS	∞	∞	10×10^{-3}	N/A	0
MC	1404.88	805 MB	80×10^{-2}	10×10^{-3}	7065
MC	1548.98	905 MB	80×10^{-2}	10×10^{-4}	6284
MC	1582.00	915 MB	80×10^{-2}	10×10^{-5}	6233
MC	1633.04	938 MB	80×10^{-2}	10×10^{-6}	6230
MC	1855.18	1.0 GB	80×10^{-2}	10×10^{-7}	6223

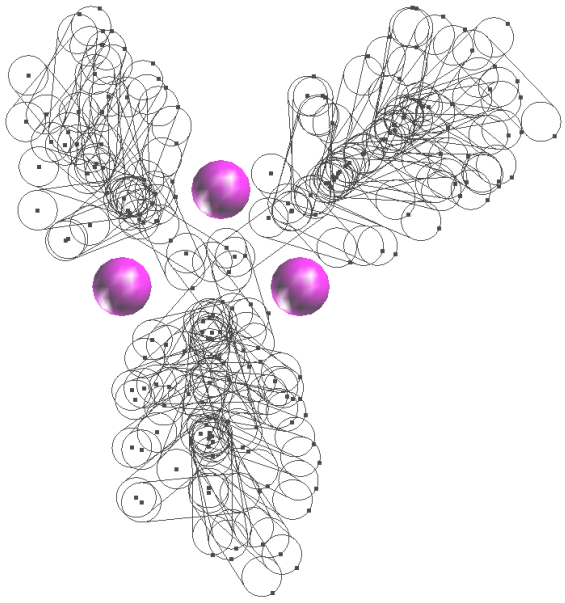

Figure 4.19: Robot drawn on the generalized Voronoi diagram, done with Monte Carlo optimization.

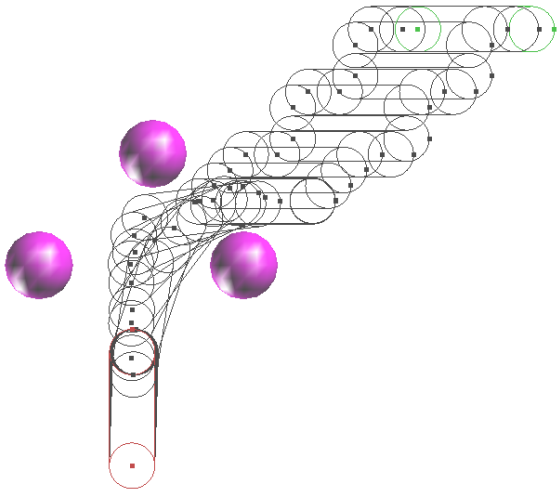

Figure 4.20: After creating the generalized Voronoi diagram using MC optimization, a start (green) position is connected to a finish position (red) to create a path through the workspace.

4.4.2 Important Point Solutions

How Important Points Can be used

As discussed in Section 2.3, in robot path planning configuration space can be thought of as being partitioned into distinct areas (not necessarily contiguous), free space or C_{free}, where the robot can move freely, and collision space or C_{obs}, where the robot will collide with an obstacle. Once these spaces are accurately described, especially the boundaries between the free and collision spaces, path planning can be done in a pretty straight forward manner - move from start point to end point using only the free configuration space.

In higher dimensional configuration space, finding where the collision space and free space are can be very challenging. Configuration space decomposition attempts to decompose the high dimensional configuration space, but will quickly run out of memory on modern computers.

We propose one starting point at which to approach this problem. Using configuration space subdivision and Monte Carlo optimization, "important points" can be found in the configuration space. These are points in space where changes in the free or collision space occur. We call them *important points* instead of *critical points* to emphasize that they enclose a more broad description than traditional use of the term "critical point".

For example, consider a 2-dimensional problem involving a grounded, 2-link rotational robot arm. The solution is found for all configurations in the configuration space for which the 2-link arm collides with an obstacle. The solution is shown in Figure 4.21 drawn in workspace, and in Figure 4.22 drawn in configuration space. Because it is only a 2-dimensional configuration space a solution can be found for the entire collision-freespace boundary, indicated by the green boundary. To clarify, the

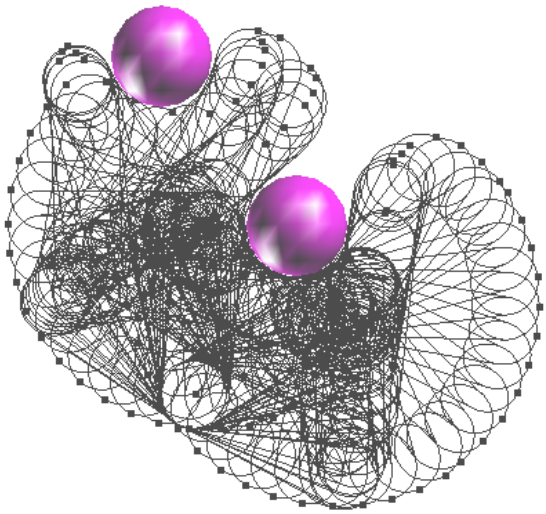

Figure 4.21: Showing a sampling of all configurations for which the 2-link robot collides, drawn in workspace.

green boundary is all the points in configuration space where the distance between the robot and obstacles becomes zero - which indicates the boundary between free and collision space.

Viewing the solution for the important points of this problem shows how they fit into the complete picture. First, the solution for everywhere the robot collides with the obstacles twice is shown. The solution drawn in workspace is shown in Figure 4.23, and in configuration space in Figure 4.24. Comparing Figures 4.22 and 4.24 reveals that they match up quite well, and the green area in Figure 4.24 are points in the complete solution where the shape of the boundary changes.

The following solution solves for all positions where the robot collides with only the very tip of the end-effector. A solution is shown in Figure 4.25 drawn in workspace, and in Figure 4.26 drawn in configuration space. By comparing these results to the complete solution in Figure 4.22, some correlation becomes apparent. These solution points seem to correlate with some of the ridges in the complete solution.

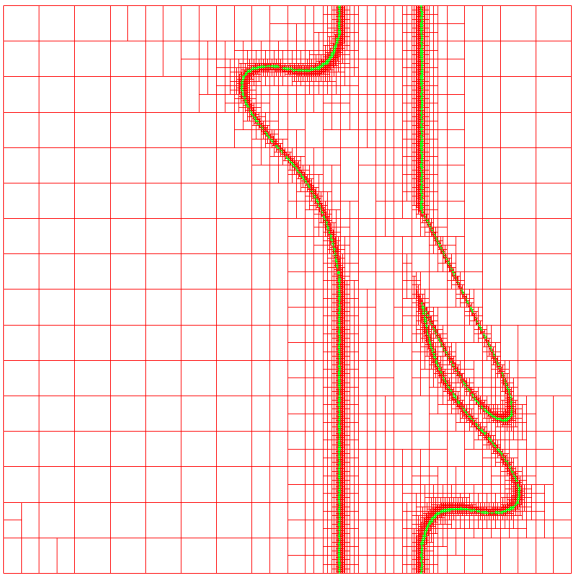

Figure 4.22: All configurations (to a precision of 10×10^{-3}) for which the 2-link robot collides, drawn in configuration space.

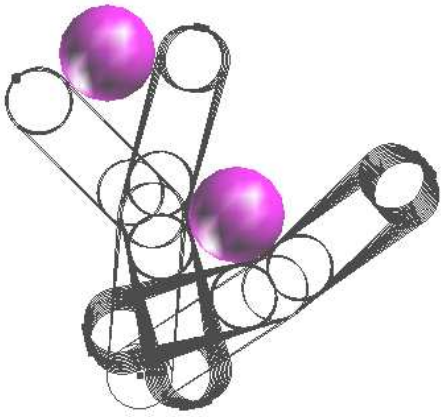

Figure 4.23: Showing a sampling of all configurations for which the 2-link robot collides twice, drawn in workspace.

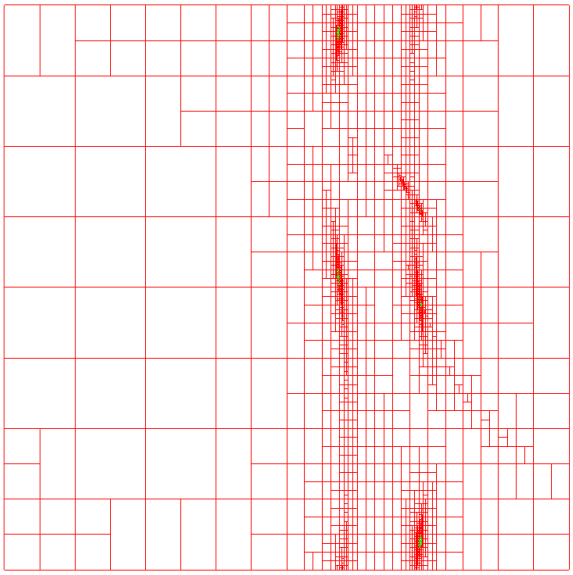

Figure 4.24: All configurations (to a precision of 10×10^{-3}) for which the 2-link robot collides twice, drawn in configuration space.

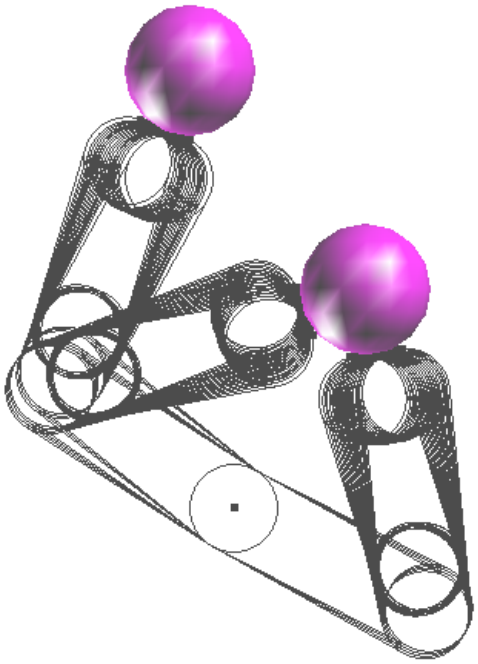

Figure 4.25: Showing a sampling of all configurations for which the 2-link robot collides with only its tip, drawn in workspace.

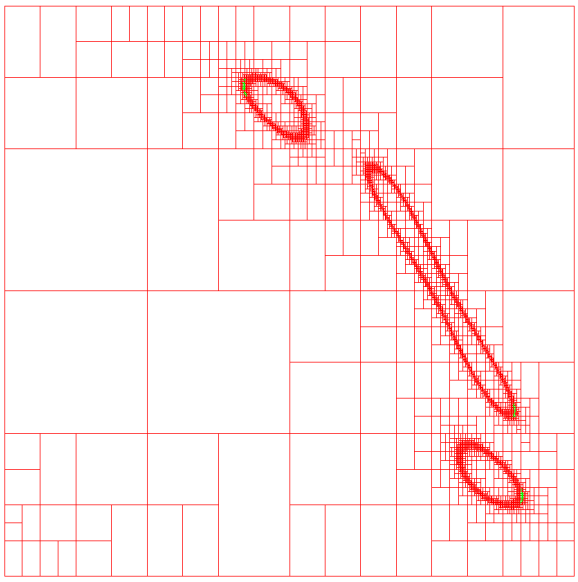

Figure 4.26: All configurations (to a precision of 10×10^{-3}) for which the 2-link robot collides with only its tip, drawn in configuration space (only the green areas are solutions).

In cases where complete solution is difficult to find, even by just solving for these two types of important points, an idea of the shape of the complete solution begins to become apparent.

In this vein of thought, while it may not be possible to represent the entire collision hyper-surface of a high dimensional problem using configuration space subdivision and Monte Carlo optimization, it would still seem useful to detect these important points. By identifying these important points, an idea of the shape of the complete hyper-surface can be gained.

A discussion of a few of these important points and success in finding them is presented, particularly how Monte Carlo optimization has performed.

Over-constrained Voronoi solutions

One set of important points that are more helpful to understand the generalized Voronoi diagram in a high dimension, rather than the complete C_{obs} and C_{free} boundary - although useful in both cases - are over-constraints of the Voronoi constraint problem itself.

In this example, another constraint is added to the Voronoi solution presented in the previous section. The constraint before was that the distance to the two nearest obstacles be equal. To over-constrain that is increased to three, so that the distance to the three nearest obstacles must be equal. The result is shown in Figure 4.27, solved using only subdivision. The solution using Monte Carlo optimization is shown in Figure 4.28, and the runtime statistics for both are listed in Table 4.9.

The results indicate that with any error tolerance larger than 10×10^{-3}, using subdivision by itself is actually better - it uses less memory in most cases and does it faster in all of them. However, if the solution needs to have an error tolerance less than 10×10^{-3}, only the MC optimization approach can find an answer. In fact, to display the advantage in finding solutions of high precision, Table 4.9 displays results with solution precision up to 10×10^{-7} - this would be very difficult to achieve with subdivision alone.

One of the reasons motivating the search for the important points in the Voronoi diagram, are for situations where higher dimensional problems needs to be solved. In these situations, these important points of the Voronoi diagram can be found - indicating areas in configuration space that are equidistant from three or more obstacles - and then used to either reconstruct the entire Voronoi diagram, or simply aid in constructing the complete picture of the C_{obs} and C_{free} boundaries.

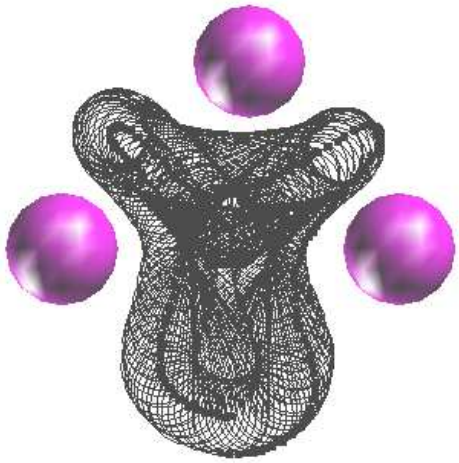

Figure 4.27: 1 link translating and rotating (3-dimensional configuration space) robot, Voronoi center point. 10×10^{-3} precision using only subdivision.

Table 4.9: Runtime statistics for finding the generalized Voronoi diagram center point.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	15.01	60 MB	10×10^{-2}	N/A	14767
MC	18.01	954 MB	20×10^{-2}	10×10^{-2}	8060
MC	21.01	39 MB	30×10^{-2}	10×10^{-2}	2972
CS	63.04	335 MB	10×10^{-3}	N/A	124407
MC	∞	∞	10×10^{-2}	10×10^{-3}	0
MC	2086.77	1.1 GB	20×10^{-2}	10×10^{-3}	2708
MC	459.29	260 MB	30×10^{-2}	10×10^{-3}	964
CS	∞	∞	10×10^{-4}	N/A	0
MC	∞	∞	10×10^{-2}	10×10^{-4}	0
MC	2455.62	1.2 GB	20×10^{-2}	10×10^{-4}	2107
MC	522.34	292 MB	30×10^{-2}	10×10^{-4}	886
MC	567.36	308 MB	30×10^{-2}	10×10^{-5}	880
MC	1119.71	580 MB	30×10^{-2}	10×10^{-6}	157
MC	1174.45	606 MB	30×10^{-2}	10×10^{-7}	2

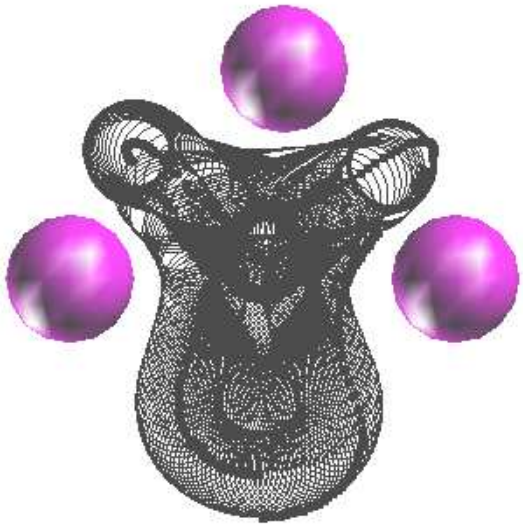

Figure 4.28: 1 link translating and rotating (3-dimensional configuration space) robot, Voronoi important point. 10×10^{-5} precision using MC optimization.

Tip Collision Detection

Tip collision is a collision involving nothing but the very tip of the end-effector of the robot and any obstacle. If C_{obs} is a surface, the tip collision would represent a boundary tip on that surface - an area in configuration space at which the collision space changes from describing where there is no collision to where there is exactly one tip of collision. This appears as a point in lower dimensions, but because of the potential multiplicity of arrangements made possible with more segments in a robot arm, this can become more than a point solution. See Figures 4.25 and 4.26 for an example of this in 2-dimensions.

3-link, non-translating, rotational robot with 1 tip collision

One problem we are able to solve and improve upon with MC optimization is the single tip collision of a 3-link, non-translating robot.

Results are presented from two subsets of the same problem. First the rotation

freedom of the robot is restricted to 180 degrees for each joint. Then a solution is found when the joints are allowed to rotate the full 360 degrees.

This contrast presents an appreciation for the size of the 360 degree problem. Additionally, these results allow easier comparison with later results for a 5-link robot which was only solved by limiting to 180 degree rotation freedom (360 was too large a problem).

First results are presented that only allow 180 degrees of rotation.

This problem is represented in 3-dimensional configuration space. The problem is first solved using only configuration space subdivision, as shown in Figure 4.29. While this is not a necessarily hard problem for only configuration subdivision to do, using Monte Carlo optimization demonstrates a speedup. The solution is shown in Figure 4.30. The computation time and memory usage for each solution are shown in Table 4.10. As the table shows, configuration subdivision was able to solve the problem in 81.05 seconds, while with the Monte Carlo optimization extension it was able to solve it in only 51.03 seconds (using a CS precision of 20-e2). Additionally, alone CS was able to solve the problem using 203MB, while with MC the solution only required 16MB. While either memory size could be easily handled by most modern computers, it is important to note how much using MC reduced the memory footprint - 7.8% of the original size.

Next results are presented where the joints are given the full 360 degrees of rotation freedom.

The problem is first solved using subdivision by itself, shown in Figure 4.31. The results of using Monte Carlo optimization are shown in Figure 4.32. The runtime statistics are presented in Table 4.11. As the table shows, MC optimization finds the solution slower, but saves a lot in memory usage, only using 11.9% of the subdivision

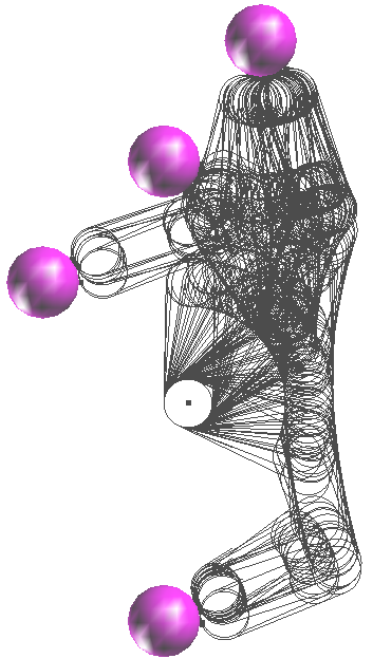

Figure 4.29: Non-translating 3-link robot rotational drawn in configurations at which it collides with obstacles only with the tip of its end effector, found with only configuration decomposition. Joints have 180 degrees of rotation freedom.

Table 4.10: Runtime statistics for finding the tip-collision solutions of a 3-link non-translating robot, where joints have 180 degrees of rotation freedom.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	81.05	203MB	10×10^{-3}	N/A	5743
MC	102.07	24 MB	10×10^{-2}	10×10^{-3}	14
MC	51.03	16 MB	20×10^{-2}	10×10^{-3}	3
MC	72.05	19 MB	30×10^{-2}	10×10^{-3}	9
CS	∞	∞	10×10^{-4}	N/A	0
MC	108.07	25 MB	10×10^{-2}	10×10^{-4}	8
MC	51.03	16 MB	20×10^{-2}	10×10^{-4}	3
MC	81.05	21 MB	30×10^{-2}	10×10^{-4}	1

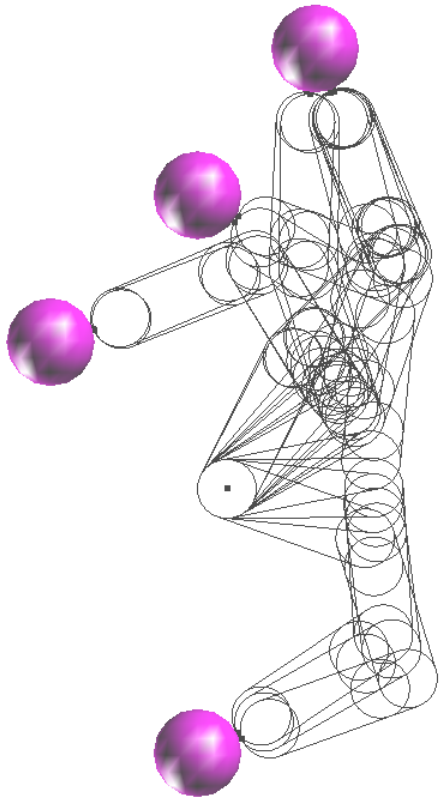

Figure 4.30: Same as Figure 4.29 but using Monte Carlo optimization.

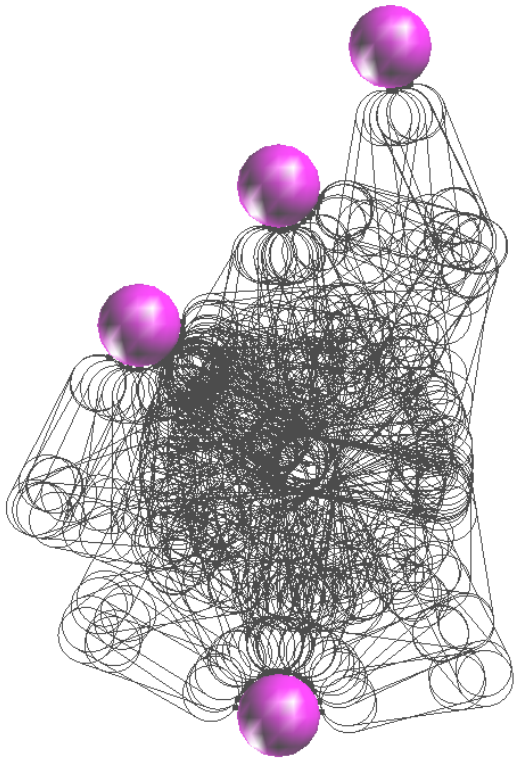

Figure 4.31: Non-translating 3-link robot drawn in configurations at which it collides with obstacles only with the tip of its end effector, found with only configuration decomposition. Joints have 360 degrees of rotation freedom.

only solution. This would be critical if the target machine only had under 1 GB of memory.

Additionally, when the target precision is 10×10^{-4}, the subdivision only approach runs out of memory, while the solution using MC optimization only requires a few hundred MB of memory. It is important to note that using MC optimization provides a way for solving to a precision unreachable with subdivision alone - thus extending subdivision in an important way.

4-link, non-translating, rotational robot with 1 tip collision

Solutions to the tip collision problem with 4-link rotational robot, resulting in a

Table 4.11: Runtime statistics for finding the tip-collision solutions of a 3-link non-translating robot where joints have 360 degrees of freedom (Figures 4.31 and 4.32).

method	time(sec)	memory	CS precision	MC precision	solutions
CS	930.61	1.9 GB	10×10^{-3}	N/A	71807
MC	1383.91	232 MB	10×10^{-2}	10×10^{-3}	152
MC	1915.58	112 MB	20×10^{-2}	10×10^{-3}	35
MC	1290.85	209 MB	30×10^{-2}	10×10^{-3}	50
CS	∞	∞	10×10^{-4}	N/A	0
MC	1506.98	252 MB	10×10^{-2}	10×10^{-4}	61
MC	672.44	114 MB	20×10^{-2}	10×10^{-4}	18
MC	1326.87	217 MB	30×10^{-2}	10×10^{-4}	12

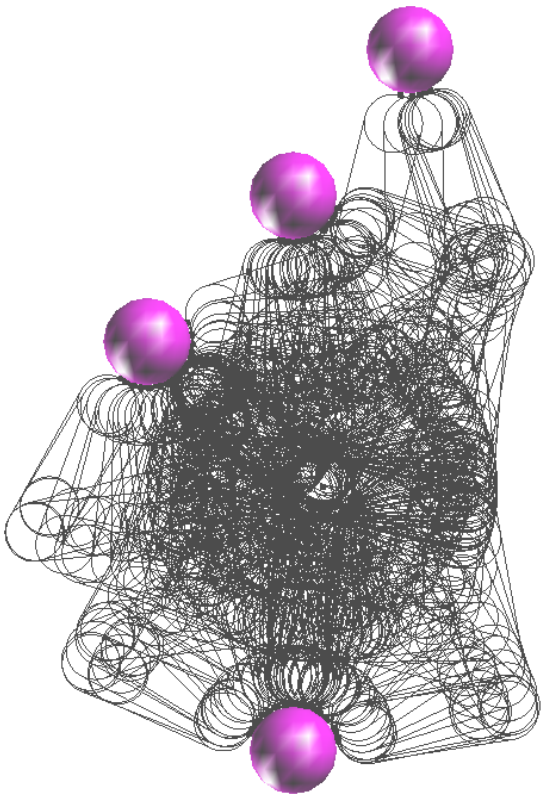

Figure 4.32: Non-translating 3-link robot drawn in configurations at which it collides with obstacles only with the tip of its end effector, found with MC Optimization. Joints have 360 degrees of rotation freedom.

Table 4.12: Runtime statistics for finding the tip-collision solutions of a 4-link non-translating robot with 180 degrees of rotational freedom.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	∞	∞	10×10^{-3}	N/A	180
CS	6.00	20 MB	10×10^{-2}	N/A	389
MC	1059.67	169 MB	10×10^{-2}	10×10^{-3}	105
MC	237.15	43 MB	20×10^{-2}	10×10^{-3}	23
MC	441.28	73 MB	30×10^{-2}	10×10^{-3}	25
CS	∞	∞	10×10^{-4}	N/A	0
MC	1122.70	180 MB	10×10^{-2}	10×10^{-4}	71
MC	279.18	49 MB	20×10^{-2}	10×10^{-4}	2
MC	480.31	79 MB	30×10^{-2}	10×10^{-4}	10

4-dimensional configuration space to search.

Again, the solution to two related subproblems is presented, one where the joints have 180 degrees of freedom and one where they have 360 degrees of freedom.

First a solution where joints have 180 degrees of freedom.

Even with only 180 degrees of rotation freedom, a solution cannot be found using only configuration space subdivision with error 10×10^{-3}, due to insufficient memory. However, it can be found with error 10×10^{-2} - but with that large of an error the results do not look very correct, as depicted in Figure 4.33. A solution using Monte Carlo optimization is presented in Figure 4.34.

In other words, Monte Carlo optimization is able to solve a higher dimensional problem than configuration space subdivision can by itself. While the configuration space by itself can find a solution, it is such a lose error bound that the result is not very usable (Figure 4.33). These results are presented in Table 4.12.

Next a solution to the same problem but with 360 degree rotational freedom is presented. Figure 4.35 shows the result, and Table 4.13 details the runtime statistics. This solution takes long enough that it probably would not be useful for most robotics

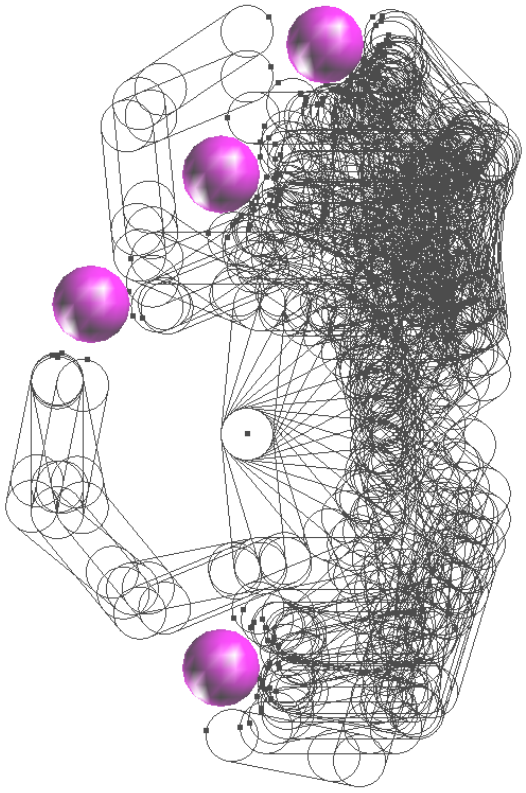

Figure 4.33: Non-translating 4-link robot drawn in configurations at which it collides with an obstacle with the tip of its end effector, found using only configuration space subdivision. 180 degrees of rotational freedom. The results have such a loose error tolerance (10×10^{-2}), that they appear like garbage results.

Table 4.13: Runtime statistics for finding the tip-collision solutions of a 4-link non-translating robot with 360 degrees of rotational freedom.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	∞	∞	10×10^{-3}	N/A	0
CS	∞	∞	10×10^{-2}	N/A	0
MC	∞	∞	10×10^{-2}	10×10^{-3}	
MC	10824.83	1.5 GB	20×10^{-2}	10×10^{-3}	328
MC	∞	∞	30×10^{-2}	10×10^{-3}	0
CS	∞	∞	10×10^{-4}	N/A	
MC	∞	∞	10×10^{-2}	10×10^{-4}	
MC	11404.16	1.6 GB	20×10^{-2}	10×10^{-4}	76
MC	∞	∞	30×10^{-2}	10×10^{-4}	0

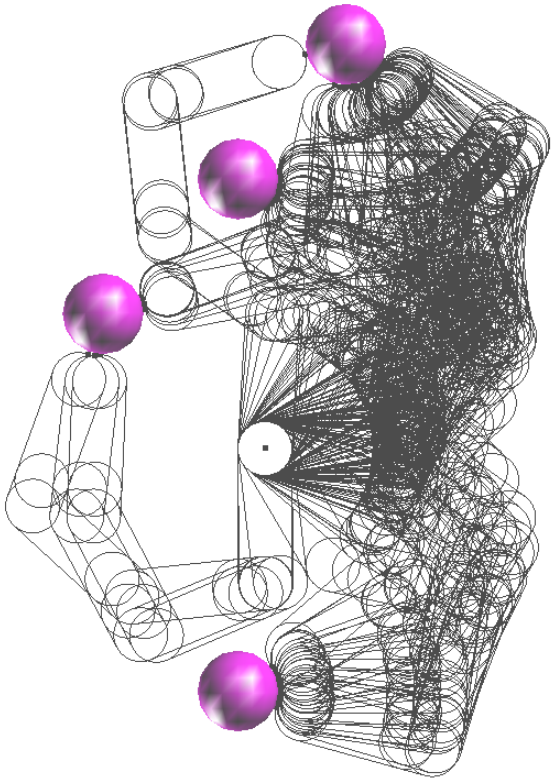

Figure 4.34: Non-translating 4-link robot drawn in configurations at which it collides with an obstacle with the tip of its end effector, found using MC optimization. 180 degrees of rotational freedom.

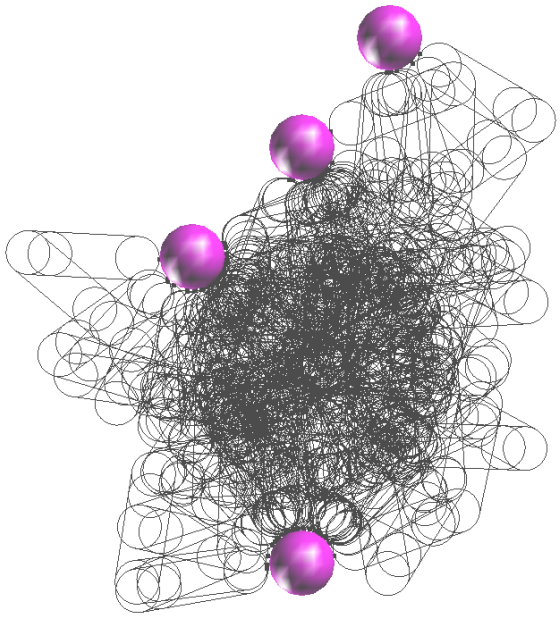

Figure 4.35: Non-translating 4-link robot drawn in configurations at which it collides with an obstacle with the tip of its end effector, found using MC optimization. 360 degrees of rotational freedom.

Table 4.14: Runtime statistics for finding the tip-collision solutions of a 5-link non-translating robot where joints have 180 degrees of freedom (Figure 4.36).

method	time(sec)	memory	CS precision	MC precision	solutions
CS	∞	∞	10×10^{-3}	N/A	0
CS	114.08	214 MB	10×10^{-2}	N/A	4168
MC	20317.69	1.8 GB	10×10^{-2}	10×10^{-3}	1116
MC	1524.99	195 MB	20×10^{-2}	10×10^{-3}	121
MC	2992.96	362 MB	30×10^{-2}	10×10^{-3}	136
CS	∞	∞	10×10^{-4}	N/A	0
MC	15078.58	1.9 GB	10×10^{-2}	10×10^{-4}	809
MC	1774.15	220 MB	20×10^{-2}	10×10^{-4}	29
MC	3344.17	527 MB	30×10^{-2}	10×10^{-4}	41

applications (as do some of the other run times), however, it is meant to emphasize that the MC optimization approach is able to solve a problem in a dimension that could not be solved using configuration space subdivision by itself. This is one of the main points of the thesis, that Monte Carlo optimization can help in speed, precision, and in problem size.

5-link, non-translating, rotational robot with 1 tip collision

The solution to a 5-link, non-translating, rotational robot with 1 tip collision problem is shown in Figure 4.36 and the runtime statistics are listed in Table 4.14. As the table indicates configuration subdivision by itself is unable to solve this problem to a usable precision. Additionally, these are only results where each joint has 180 degrees of freedom - solutions were not found using 360 degress of rotation due to memory constraints. As with the 4-dimensional problem, while the timings are not very encouraging for real-time use, they more importantly bring across the point that MC optimization is able to extend subdivision to solve higher dimensional problems with a reasonable precision.

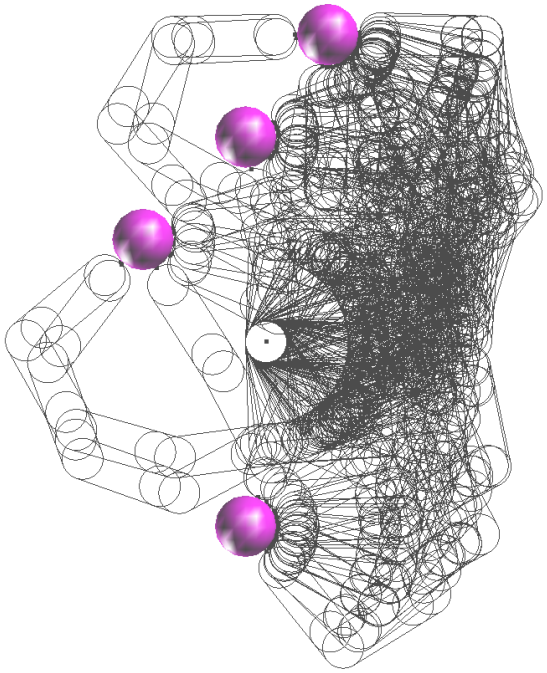

Figure 4.36: Non-translating 5-link robot drawn in configurations at which it collides with an obstacle with the tip of its end effector, found using MC optimization.

Table 4.15: Runtime statistics for finding the 2-collision points of a 2-link non-translating robot.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	3.00	1 MB	10×10^{-2}	N/A	33
CS	3.00	1 MB	10×10^{-3}	N/A	30
MC	54.04	65 MB	10×10^{-2}	10×10^{-3}	1

Multi-Collision Detection

A double collision point in configuration space was discussed in Section 4.4.2. This section discusses success in finding multiple-collision points.

We experienced limited success using Monte Carlo optimization to improve upon solutions to this important point. These results are presented, as well as some discussion as to why this may be. Additionally some results using only configuration space subdivision are presented - to complete the motivation for future work involving important points. This also reveals some shortcomings of Monte Carlo optimization.

2-link, non-translating, rotational robot with 2 collision points

The solution to the constraint involving the double-collision of a two link robot with obstacles using only configuration subdivision is presented in Figure 4.23 in Section 4.4.2. Here, Figure 4.37 presents the solution solved using MC optimization. As Table 4.15 presents, the configuration subdivision only approach solves the problem in less time and surprisingly even less memory.

A discussion of some of the reasons for this shortcoming is presented at the end of this section. In our opinion the results are mostly due to an error metric that is difficult to optimize using Monte Carlo optimization.

3-link, non-translating, rotational robot with 4 collision points

Here the solutions to the 3-link rotational robot colliding at 4 points is presented. Figure 4.38 depicts the solution found using configuration space subdivision by itself.

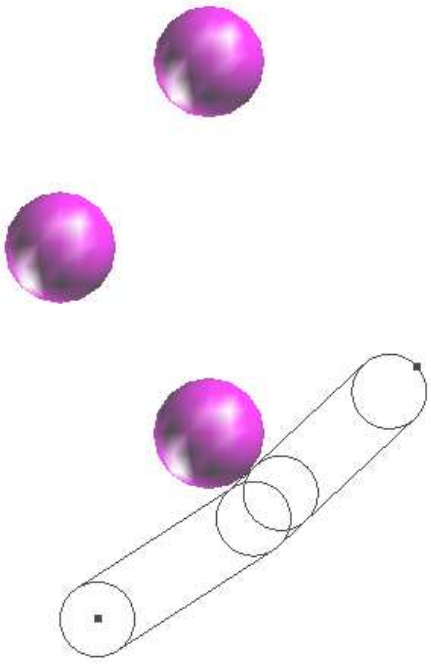

Figure 4.37: Non-translating 2-link robot drawn in configurations at which it collide with obstacles 2 times, solved using MC optimization.

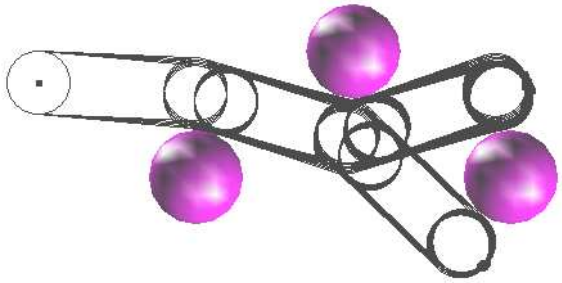

Figure 4.38: Non-translating 3-link robot drawn in configurations at which it collide with obstacles 4 times, solved using only subdivision.

Table 4.16: Runtime statistics for finding the 4-collision points of a 3-link non-translating robot.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	6.00	27 MB	10×10^{-3}	N/A	89
CS	6.00	27 MB	90×10^{-4}	N/A	89
MC	468.29	425 MB	10×10^{-3}	90×10^{-4}	2
CS	6.00	36 MB	10×10^{-4}	N/A	4

Figure 4.39 is the solution found using MC optimization. As with the 2-link, 2-collision case, Table 4.16 presents results showing that the solution using configuration space subdivision is faster and uses less memory.

This is another case displaying the shortcomings of the MC optimization step.

4-link, non-translating, rotational robot with 6 collision points

One solution found using configuration space subdivision by itself was the 6-collisions of a 4-link, non-translating robot.

To fully represent the collision and free space in 4-dimensional configuration space using configuration space subdivision would be difficult using current hardware. However, using configuration space subdivision, these important points can be found.

These points are detectable because it is easy to tell early on whether a very large hypercube of configuration space, a configuration space volume, contains a configura-

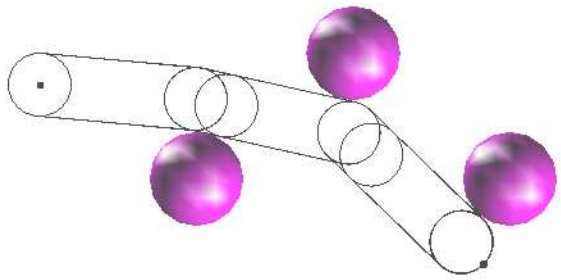

Figure 4.39: Non-translating 3-link robot drawn in configurations at which it collide with obstacles 4 times, solved using MC optimization.

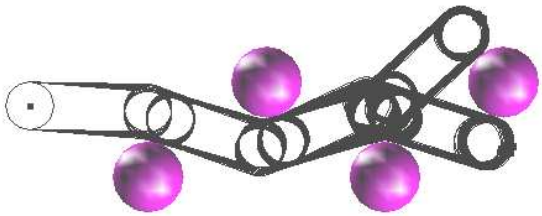

Figure 4.40: Non-translating 4-link robot drawn in configurations at which it collide with obstacles 6 times

tion that will satisfy that geometric constraint. This is in part because the constraint is so precise. With enough early trimming, the entire configuration space is able to be fully searched to find the points where this 6-collision occurs.

Figure 4.40 shows a solution to this problem using configuration space subdivision alone. Table 4.17 shows the runtime results of that process. No interesting results were found using MC optimization for this case - again due to the shortcomings discussed at the end of this section.

Discussion on Monte Carlo Optimization Shortcomings

These results reveal a problem for which MC optimization did not improve the subdivision approach in some way. Here some ideas are proposed as to why this

Table 4.17: Runtime statistics for finding the 6-collision points of a 4-link non-translating robot.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	306.19	1.7 GB	10×10^{-3}	N/A	1021

Table 4.18: Runtime statistics for finding the 2-collision points of a 2-link non-translating robot.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	3.00	1 MB	10×10^{-3}	N/A	30
CS	3.00	1 MB	10×10^{-4}	N/A	24
MC[1]	9.01	17 MB	10×10^{-3}	10×10^{-4}	26

shortcoming occurs, with some supporting test runs and results.

The error metric, which provides the guidance of the sampling towards the solution, plays a vital role in the MC optimization method. We believe the problems experienced in solving for problems involving multiple collisions is in part due to the difficulty of describing the error for this problem with a single scalar value. Considering the robot in workspace, the distance to the nearest n obstacles (where the next n collisions will take place) describes the error, but moving closer to one collision may bring the robot further from the other. An improvement towards contact at one point with an obstacle can increase the distance to another collision point. In other words, the error metric may not provide a very direct guide to the solution, so that maximum iterations are reached and no solution is found. This makes it difficult for Monte Carlo optimization to optimize the error.

As an illustration of this situation, we change the Monte Carlo optimization error metric to only be the distance to the closest obstacle for the 2-link robot, 2 collision example. In other words, the configuration subdivision step will solve for 2 collisions,

[1]error changed to consider only the closest obstacle distance, which is not correct for a multi-collision problem.

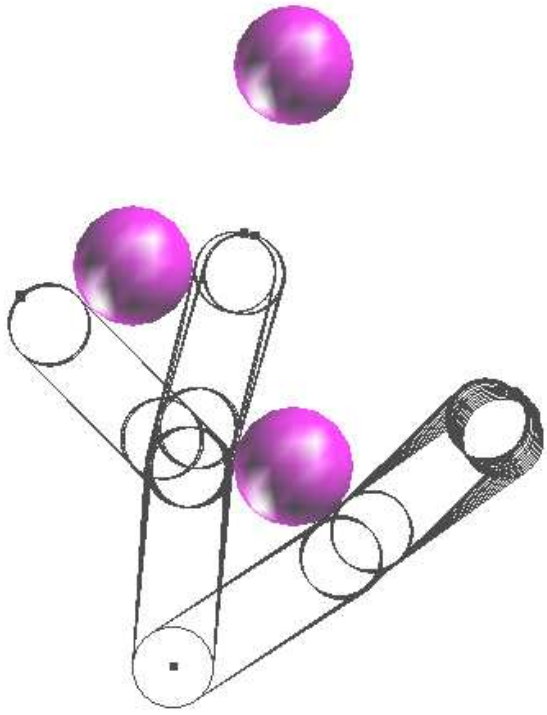

Figure 4.41: Non-translating 2-link robot drawn in configurations at which it collide with obstacles 2 times, found using MC optimization where MC error was only the closest collision distance.

Table 4.19: Runtime statistics for finding the 4-collision points of a 3-link non-translating robot.

method	time(sec)	memory	CS precision	MC precision	solutions
CS	6.00	27 MB	10×10^{-3}	N/A	89
CS	6.00	36 MB	10×10^{-4}	N/A	4
MC[1]	9.01	34 MB	10×10^{-3}	10×10^{-4}	88

and when it then uses Monte Carlo optimization to find a more precise solution, the error will only be for 1 collision. While the solution will not be entirely correct, it will illustrate the exact same setup, but with a different error metric - one that is very directly leading to a solution. The results are shown in Table 4.18, and drawn in workspace in Figure 4.41. The table shows that the runtime is now on par with the CS solution timings. This data gives some support that in the same situation with an error metric that clearly leads to a solution, the runtime is faster.

To provide a similar example, we use the same approach in the 3-link, 4 collision detection problem. We modify the error metric to only include the closest obstacle distance. The resulting runtime statistics are presented in Table 4.19. Again, with the simpler error construction, the runtime is much more similar to the configuration subdivision approach.

This data suggests that, at least in part, a poor error metric may be to blame for the poor results in multi-collision detection using MC optimization. In an attempt to remedy this problem, we have tried a few different approaches but with limited success. We leave further exploration to future work. In particular, exploring how to define the error in a different way, one more conducive to MC optimization, would be an ideal solution.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we have presented a novel approach to finding the final solution in configuration space subdivision by adapting Monte Carlo optimization. We have shown that this new approach allows us to solve constraint problems to a higher degree as well as a higher precision than configuration space subdivision can alone.

The solution of several general constraint problems in 3-dimensions have been shown using the combination of configuration space subdivision and Monte Carlo optimization. The results have shown that using Monte Carlo optimization is able to solve the same problem faster and with less memory than with subdivision by itself.

We have also deomonstrated its use in path planning a 3-dimensional configuration space problem by solving for the gernalized Voronoi diagram in configuration space. This has shown cases where the Monte Carlo optimization has performed better, and in every case required less memory. We have shown that using MC optimization can acheive much better precision than is possible with only configuration space subdivision.

Additionally, we have explored some "important point" examples, showing both successes and shortcomings of the approach. For the over-constrained Voronoi problem we have shown how for smaller precisions the configuration subdivision alone actually outperforms MC optimization, while for higher precision MC optimization performs better. In fact, in order to acheive any precision better than 10×10^{-4}, only MC optimization can find the solution. For the tip-collision problem, we have shown how Monte Carlo optimization has helped to solve constraint problems up to 5-dimensions, while with only subdivision can only solve up to 3 dimensional constraint problems with a usable error. And with multiple collision problems, we found an important shortcoming of MC optimization - that the error metric must be very directly guiding the samples to the solution. If there is not a clear direction for the samples to follow in the error metric, it will be difficult to solve.

Finally, we have presented some beginning steps into analysis of higher-dimensional path planning problems using the important points of path planning related constraint problems. Our contribution has only been to present how over-constraint of a problem can lead to these important points, and we propose future work be done in making use of these points to solve higher dimensional constraint and path planning problems.

In all we have shown how adapting Monte Carlo optimization to work in conjunction with configuration space subdivision can both improve the precision of configuration space subdivision, and in some cases, increase the dimension of problem it can solve.

5.2 Future Work

Most of the effort in this thesis has centered on adapting Monte Carlo optimization to improve upon the configuration space subdivision approach. The area where we propose future work explore most is expanding the use of important points of higher dimensional constraint problems, especially in robot path planning. As we have shown in this thesis, by over-constraining a problem, the size of the solution can be reduced, in some cases reducing to point solutions. Over-constraint is not a new idea, but we have demonstrated how some of these over-constraint solutions participate in the larger, less-constrained solution. If these important points can then be used to reconstruct the complete solution using an alternate method, it could be useful in path planning and other related geometric constraint problems of higher dimensions.

More closely tied to the adaption of Monte Carlo optimization to work with configuration space subdivision, we also propose future work exploring the solution of more constraint problems using this method, and particularly in formulating error metrics for problems like multiple collision, where the trivial error metric does not produce desired results.

Bibliography

[1] J. Barraquand and P. Ferbach. Path planning through variational dynamic programming. 1994.

[2] Ronen. Barzel. *Graphics tools : The jgt editors' choice.* A K Peters, Wellesley, Mass., 2005.

[3] J.-D. Boissonnat. Algorithmic foundations of robotics v. Springer tracts in advanced robotics, v. 7, Berlin; New York, 2004. Springer.

[4] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and System, 2002*, volume 3, pages 2383–2388, Piscataway, NJ, USA, 2002. IEEE.

[5] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents).* The MIT Press, June 2005.

[6] Bruce R. Donald. Motion planning with six degrees of freedom. Technical report, Cambridge, MA, USA, 1984.

[7] Bruce R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353, 1987.

[8] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 1–10, New York, NY, USA, 2001. ACM Press.

[9] George S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag New York, Inc., 1996.

[10] M. Foskey, M. Garber, M.C. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and System, 2001*, volume 1, pages 55–60, Piscataway, NJ, USA, 2001. IEEE.

[11] R. Geraerts and M. H. Overmars. On improving the clearance for robots in high-dimensional configuration spaces. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005).*, pages 679–684, Piscataway, NJ, USA, 2005. IEEE.

[12] L. J. Guibas, C. Holleman, and L.E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. (IROS '99).*, pages 254–259, Piscataway, NJ, USA, 1999. IEEE.

[13] John H. Halton. A retrospective and prospective survey of the monte carlo method. *SIAM Review*, 12(1).

[14] L.E Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, August 1996.

[15] Myung-Soo Kim, Gershon Elber, and Joon-Kyung Seong. Geometric computations in parameter space. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 27–32, New York, NY, USA, 2005. ACM Press.

[16] Glenn A. Kramer. *Solving Geometric Constraint Systems: A Case Study in Kinematics*. The MIT Press, 1992.

[17] S LaValle and J K Jr. Rapidly-exploring random trees: Progress and prospects. In *In 2000 Workshop on the Algorithmic Foundations of Robotics*, 2000.

[18] S M LaValle. Rapid-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[19] S. W. Mahfoud. Crowding and preselection revisited. In *Parallel problem solving from nature 2*, Amsterdam. North-Holland.

[20] Samir W. Mahfoud. Simple analytical models of genetic algorithms for multimodal function optimization. Technical Report IlliGAL Report No 93001, Department of General Engineering, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801-2996, 1993.

[21] Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., 1981.

[22] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Ltd., 1977.

[23] Hans-Paul Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, Inc., 1995.

[24] J.K. Seong, G. Elber, and E. Cohen. Simultaneous precise solutions to the visibility problem of sculptured models. In *Geometric Modeling and Processing (GMP)*, pages 451–464, 2006.

[25] Joon-Kyung Seong, David E Johnson, and Elaine Cohen. A higher dimensional formulation for robust and interactive distance queries. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 197–205, New York, NY, USA, 2006. ACM Press.

[26] Peter Shirley. *Physically based lighting calculations for computer graphics*. PhD Thesis. Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[27] Ilya M. Sobol'. *A Primer for the Monte Carlo Method*. CRC Press, Inc., 1994.

[28] Gokul Varadhan and Dinesh Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[29] Eric W. Weisstein. Voronoi diagram. *MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/VoronoiDiagram.html*.

[30] L Zhang and D Manocha. An efficient retraction-based rrt planner. In *IEEE International Conference on Robotics and Automation (ICRA), 2008*, 2008.

[31] Liangjun Zhang, Young J. Kim, and Dinesh Manocha. A simple path non-existence algorithm using c-obstacle query. 2006.

[32] Liangjun Zhang, Young J. Kim, and Dinesh Manocha. A hybrid approach for complete motion planning. 2007.

[33] D.J. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. In *IEEE Transactions on Robotics and Automation*, volume 7, pages 9–20, Piscataway, NJ, USA, 1990. IEEE.