Solving Distance Problems with Concave Bodies
Using Simulated Annealing

Juan A. Carretero and Meyer A. Nahon
Space and Subsea Robotics Laboratory
Dept. of Mech. Eng., University of Victoria
Victoria, British Columbia, Canada
e-mail: jcarrete@uvic.ca, mnahon@uvic.ca

On Ma
MacDonald Dettwiler,
Space and Advanced Robotics Ltd.
Brampton, Ontario, Canada
e-mail: oma@mdrobotics.ca

Abstract — Determining the minimum distance between two convex objects is a problem that has been solved using many different approaches. Some methods rely on computational geometry techniques, while others rely on optimization techniques to find the solution. Some fast algorithms sacrifice precision for speed while others are limited in the types of objects that they can handle (e.g. linearly bound objects, quadratically bound objects, etc.).

On the other hand, computing the minimum distance between combinations of convex and concave objects is known to be a more complicated problem. Some methods propose to partition the concave object into convex sub-objects and then solve the convex problem between all possible sub-object combinations. While this method has been shown to work reliably, it adds a large computational expense when the concave objects in the scene are complicated, or when a quadratically bound object is to be linearized.

In this paper an optimization approach is used to solve the concave problem without the need for partitioning the concave object into convex sub-objects. Since the optimization problem is no longer unimodal (i.e. has more than one local minimum point), a global optimization technique is used.

Simulated Annealing is used to solve the concave problem. In order to reduce the computational expense, it is proposed to replace the objects’ geometry by a set of points on the surface of each body. This reduces the problem to a combinatorial problem where the combination of points (one on each body) that minimizes the distance will be the solution. Some examples using this novel method are presented.

1 Introduction

Probably the most popular use of the distance determination algorithms is in robot path planning [1] where the trajectory of a robot through a workspace with objects is planned in order to have a collision free trajectory.

Simulation of physical systems and virtual experimentation are other applications for the minimum distance problem. In many cases the simulation of physical systems requires the use of some form of contact dynamics model where the separation or interference distance between the objects is required [2] (see Figure 1).

![Fig. 1. Examples of the minimum separation distance between convex and concave objects.](attachment:image)

Similarly, CAD / CAM applications, such as mechanical assembly, tool path planning, virtual reality and virtual prototyping require the use of distance determination algorithms or interference identification in order to make the task possible or the virtual models more realistic.

Some of the above applications require the minimum distance problem to be solved at many instants as the scenario changes in time, e.g. a robot moving a payload amongst objects. In other applications, only the fact that the objects are interfering or not is of interest, i.e. no detailed information is needed about separation or interference (e.g. mechanical assembly). On the other hand, in contact dynamics, a detailed knowledge of the interference geometry (interference distance and volume, etc.) is desirable.

In this paper a new optimization based method is described. The proposed method, unlike many others, does not require partitioning the concave object into convex sub-objects. On the other hand, since the objective function to be minimized (i.e., the distance between objects) is no longer unimodal (i.e., has more than a single local minimum point), global optimization techniques are used.

A Simulated Annealing algorithm was designed to solve the concave problem. In order to reduce the run time computational expense, the objects’ geometries are replaced by a set of points on the surface of each body. This reduces the optimization to a combinatorial problem where the combination of points (one on each body) that minimizes the distance will be the solution. Hence, the solution found is an approximation whose accuracy will depend on the coarseness of the surface mesh.

2 Distance Determination

2.1 Minimum Distance for Convex Objects

Interference detection algorithms, where two objects are checked for interference, such as the one described by Maruyama [3], are the predecessors of some of the distance determination algorithms used nowadays. Maruyama’s interference method exhaustively checks the boundaries of every pair of faces for interference. If any two faces interfere, then the objects are said to interfere.

Some analytical methods, e.g., [4], exhaustively compute the distance between edges and vertices of one object and those on the other object. The minimum distance is then determined by comparing all the obtained distances.
2.2 Minimum Distance for Concave Objects

2.2.1 Partitioning Methods

Some works have reported the use of convex partitioning (see Figure 2) of concave bodies to solve the distance determination problem between concave bodies. These methods solve the distance problem for every object pair and return the minimum of all the distances as the solution of the concave problem. That is, once the partition is done, usually off-line, the convex distance problem is solved between all the sub-body pairs. The overall solution will then be obtained as the minimum distance between all pairs.

While these methods work and rely on very fast and robust convex methods, the computational expense becomes a problem when dealing with complicated concave objects. Let us consider the following example: if the minimum distance between two concave objects created from ten convex subobjects is to be solved, then, one hundred convex minimum distance problems will need to be solved.\(^2\)

2.2.2 Non-partitioning Methods

An interference detection based on Boyse’s scheme [4] for convex polyhedra was extended to non-convex polyhedra in [12], where every pair of surfaces is checked for interference. The line of intersection between pairs of planes (containing each of the faces in question) is first obtained. The line is divided in small segments that reach the faces’ limits. If any segment is fully contained within the boundaries of both faces then the two faces are said to interfere except when one of the surfaces is a hole. If the face has a hole (a concavity) it is modeled as a negative entity, thus if a line segment is contained within the boundary of that face there is no interference. This method only returns a boolean result, i.e. whether or not there is interference, but does not quantify the minimum distance.

Other works have concentrated on the decomposition-free solution of the distance problem for non-convex bodies. The method described in [13], which is similar in concept to the work by Canny and Donald [14] [15], computes the distance between all possible vertex-plane and edge-edge pairs and returns the minimum value as a result. Pruning strategies are used to reduce the computational expense of computing the distance between all pairs.

The advantages of the decomposition-free approaches include i) the minimization of the number of object pairs to be tested since the addition of extra features (surfaces, edges, vertices, etc.) is avoided and ii) the solution of the problem in a single run rather than solving the distance problem amongst all pairs of sub-objects. Both advantages can potentially reduce the number of computations needed to solve the concave problem.

3 Solving the Distance Problem

As discussed earlier, one of the methods used to solve the minimum distance problem between convex objects is formulated as a constrained optimization problem. The method locates one point inside or on the surface of each body. Then, using the Cartesian positions of each of these two points as search variables, the Euclidean distance between the two points is minimized subject to a set of inequality constraints (defining the geometry of each body). This problem can be expressed as follows [10]:

\[
\begin{align*}
\text{minimize} & : \sqrt{(p_1 - p_2)^T(p_1 - p_2)} \\
\text{subject to} & : g_j(p_i) \leq 0 \text{ for } j = 1..k \\
& : g_j(p_2) \leq 0 \text{ for } j = k + 1..n_{\text{constr}}
\end{align*}
\]

where \(p_i = [x_i \ y_i \ z_i]^T\) corresponds to the Cartesian coordinates of the point on the \(i\)-th body, \(g_j(p_i) \leq 0\) are the low level geometry primitives defining the geometry of both bodies and \(n_{\text{constr}}\) is the total number of constraints.

\(^1\)A point inside the body is one that satisfies all the constraints that define the geometry of the object.

\(^2\)The number of convex subobjects can be very large, particularly for quadratic objects that need to linearized before they are partitioned into convex pieces.
Both, \( p_i \) and \( g_j \) must be expressed in the same coordinate frame.

In the convex case, the previous problem has a unique solution (\textit{i.e.}, it is unimodal) and can be solved using quadratic programming techniques \cite{10} when the constraints are linear. On the other hand, when at least one of the objects is concave \(^3\), the objective function may no longer be unimodal, \textit{i.e.}, more than one local minimum solution could exist.

### 3.1 Global Optimization

If a function having more than one local minimum (\textit{a.k.a.} multimodal function) is to be optimized using an algorithm based on gradient search, the solution obtained may or may not be the global minimum. When the start point for a gradient based method is close to a local minimum, the algorithm will quickly move toward it and eventually converge there, \textit{i.e.} the algorithm is trapped in the local minimum. This means that if a different start point is used, a different solution may be found. Figure 3 illustrates this with a single variable function. Often, a multi start search is used to bolster the confidence in the minimum value obtained using gradient-based approaches (see for instance [16]).

As an alternative to gradient-based approaches, a variety of different \textit{global} optimization methods have been proposed in the literature. Amongst the most popular techniques, one can find the Genetic Algorithms (GA) and Simulated Annealing (SA). Both these global optimization algorithms have been investigated at this time, but this paper only focuses on the SA implementation.

### 3.2 Simulated Annealing

Simulated Annealing is a global optimization technique which is based on an analogy of the cooling process of melted metals \cite{17}.

The basic SA algorithm is described in point form as follows \cite{18}:

1. Randomly create an initial guess (a pair of points, one on each object)
2. Evaluate objective function at the initial guess
3. Randomly generate a new pair of points that satisfies the constraints and evaluate the objective function
4. If the objective function at the new pair is improved, increase the iteration number (in SA parlance: decrease the temperature) and go to step 3 unless the algorithm has converged or the maximum iteration number is exceeded.
5. Generate a normally distributed random number and compare it to a Boltzmann probability function (one that decreases as the iteration number increases). If it is smaller accept the new point pair and return to step 3, else keep old pair of points and return to step 3.

The most important aspect of the SA process is the \textit{cooling schedule}: \textit{i.e.} the expression defining the Boltzmann probability function (step number 5 described above). A highly exploratory algorithm has a very slow cooling schedule accepting points with poor objective function even after many iterations have passed. Once enough iterations have passed, only a few moves that worsen the objective function are accepted. By the end of the SA process, only moves that improve the objective function are accepted. This randomness allows the search to escape from local minima.

### 3.3 Combinatorial Optimization Method

It is important to notice that every time a point is generated it has to be checked for feasibility, \textit{i.e.}, to ensure the point is inside or on the surface of a body. This entails evaluating all the constraints at that particular trial point.

For example, if one had two simple objects defined by 20 constraints each, the number of multiplications and additions (\textit{i.e.} flops) to verify feasibility would be 240, whereas only 11 flops are needed to evaluate the objective function. Thus the number of flops required to evaluate the objective function is about 22 times smaller than the number required to check the point’s feasibility.

To avoid having to calculate the constraints at every iteration, it is proposed to replace the geometry of the objects by a finite number of points on the surface of the object. Thus, if a fixed number of points is given, the minimum distance problem is reduced to a combinatorial optimization problem. That is: which is the best combination of two points (one on each body) which yields the minimum distance between them. Since the points on the surfaces of each body can be generated off-line, the number of algebraic calculations per iteration will be substantially reduced.

Although this method eliminates the computational expense involved on checking the feasibility of any point, it poses some accuracy problems. That is, the obtained solution will be an approximation to the real solution and will be governed by the number of points on each object and the distribution of these points.

This method also imposes a few implementation challenges such as the creation evenly-distributed points on the entire surface of both bodies.

#### 3.3.1 Random Points on the Surface

One possible method to obtain points on the surface of each body is by randomly distributing points on the surface of the body. To ensure an even distribution of points, one could calculate the point density by calculating the entire object’s area and dividing it by the total number of points. It is then possible to locate a limited number of points on

\(^3\)Special care must be taken when representing the geometry of a concave object since the simple inequality constraints method described in equation (1) can only describe convex objects.
each surface proportional to its area. Furthermore, while working with linearly bounded objects, it is possible to add extra points at every vertex and evenly on each edge since it is there that the solutions are most likely to be found.

3.3.2 Points on a Mesh

Meshes can be used in order to create an even distribution of points on the surface of each body. The use of meshes allows the possibility of also using a local minimum search. That is, at every iteration, any combination of points obtained at random, could be locally optimized by moving the actual points along the edges of the mesh using a gradient based search. This is possible since any point on the mesh is ‘aware’ of its neighbours.

A unstructured mesh allows motions of a single step at a time, i.e. the gradient can be calculated at every point in the direction of all its neighbours and the gradient with the largest change in the negative direction is used as the direction where the point moves next.

4 Implementation

Although many examples of SA can be found in literature, this stochastic method has to be tailored to the particular problem in order to solve it more efficiently than with a generic implementation. This section describes the most relevant aspects of the implemented algorithm.

4.1 Coding

One of the most important aspects of the implementation of any SA optimization is the coding of every trial point, i.e. the search variables. In the present problem, the coding was done by having only a set of two indices each referring to a particular point on the surface of each body such as \( X = [ n_i \quad n_j ] \). When the mesh is created, each node on the mesh is assigned a particular index. In the present case, the nodes of each object’s mesh are stored on a single matrix where each row contains the three Cartesian coordinates of the node point with respect to a body-fixed frame. Thus, \( n_i \) corresponds to the point stored on row \( i \) of the node matrix of body 1 (i.e., \( \mathbf{P}_1 \)). Similarly, \( n_j \) corresponds to the point stored on row \( j \) of the second body’s node matrix \( \mathbf{P}_2 \).

The objective function used in both algorithms corresponds to the square of the Euclidean distance between the two points. This is done to avoid the computational expense of computing the square root in equation 1 and can be done since minimizing \( a \) is equivalent to minimizing \( a^2 \) if \( a \) is a positive number. In order to evaluate the objective function, the points are first extracted from the database. Then, they are expressed with respect to a common frame.

In order to reduce the computational expense of expressing both points with respect to the inertial frame, it is possible to express both points with respect to the same body fixed frame, namely frame \( \Sigma_1 \) attached to body 1. Thus, only the points extracted from the database of body 2 have to be multiplied by the homogeneous transform describing body 2 with respect to body 1 as follows

\[
\mathbf{T}_2 \left[ \begin{array}{c} \mathbf{P}_{2_1}^T \\ 1 \end{array} \right] = \mathbf{T}_0 \mathbf{T}_2 \left[ \begin{array}{c} \mathbf{P}_{2_1}^T \\ 1 \end{array} \right]
\]  

(2)

Where \( \mathbf{T}_b \) corresponds to the homogeneous transform that describes frame \( b \) with respect to frame \( a \) and \( \mathbf{P}_{2_1} \) corresponds to the point \( \mathbf{p}_{2_1} \) stored in the node matrix \( \mathbf{P}_2 \), i.e., row \( j \) of matrix \( \mathbf{P}_2 \). The extra row containing 1 used to express \( \mathbf{P}_{2_1} \) is required in order to keep the dimensions consistent. The use of this method was found to reduce the computational expense between 30 to 50%.

Note that such transformation is possible without affecting the distance calculation since the Euclidean distance is frame invariant.

4.2 Cooling Schedule

A Boltzmann probability function expressed as \( p_b = e^{-\frac{f(x_0) - f(x_i)}{k_B t}} \) was used, where \( k_B \) is Boltzmann constant, \( t \) is the temperature and \( f(x_0) \) and \( f(x_i) \) are the values of the objective function at the initial point and the current point, respectively. The Boltzmann constant is usually found by experimentation, i.e. by allowing the SA process run a few times and making sure there is no premature or excessively slow convergence. In the present case, \( k_B \) was found after some trials to be dependent on the initial temperature. Since some runs were allowed to run for much longer than others, \( k_B \) was given two different values.

4.3 Local Optimization

Based on the information that can be extracted from the unstructured mesh, a local optimization procedure was implemented. A connectivity matrix is obtained when the mesh is created that provides the point index of all the neighbouring points to a particular node.

Using the connectivity matrix this method looks to reduce the distance between the two selected points by moving the selected nodes along the edges of the mesh, i.e. to a neighbouring node. First the selected node \( n_i \) on object one is moved to its neighbours while the node on object two \( n_j \) is fixed. The neighbour of \( n_i \) that minimizes the distance between \( n_i \) and \( n_j \) is accepted as the new point \( n_i \) and the search for a closer neighbouring point starts again. Once the best \( n_i \) is found the process is repeated by fixing \( n_i \) and moving \( n_j \).

This procedure is repeated, alternating the moving and the fixed node until the distance between the nodes is minimized, i.e., until no neighbouring node can be found that reduces the distance between \( n_i \) and \( n_j \).

Since this is a gradient based method it is clear that, on the local scale, the solution will greatly depend on the start point. However, in the larger scale, the SA algorithm is expected to introduce the necessary ability to jump out of local minima ‘traps’.

1510
5 Examples

A three dimensional test case is presented here. It consists of a set of two objects where one is a body of revolution (cylinder with a groove) while the second one is a more complex object with a few concavities, see Figure 4.

Figure 5 shows the top view of the first test case (objects in position A) where three possible initial guesses ($\mathbf{X}_1^0$, $\mathbf{X}_2^0$ and $\mathbf{X}_3^0$) and the final (optimal) point pairs returned by the SA algorithm ($\mathbf{X}_1^*$ and $\mathbf{X}_2^*$) are shown. The SA process was implemented in Matlab. Similarly, for a second test case (objects in position B), Figure 6 shows the initial guesses and the final solutions found by the different SA algorithms.

Tables 1 and 2 show the results obtained from the three different start points using the two SA algorithms tested. SA-1 and SA-2 are similar except that the latter algorithm uses the local search described in Section 4.3 at each iteration. SA-1 was run for 10000 cycles whereas SA-2 was only run for 100 since it was seen after a few tests that the optimal combination of points was always found within the first 60 iterations.

Figures 7 and 8 are examples of the time history of the minimum distance for SA-1 and SA-2, respectively, for the objects in position A.

It is interesting to notice that, when no local optimization is used, due to the stochastic nature of the optimization method, the algorithm does not always converge to the true closest point. On the other hand, when the local optimization is combined with the global method, the solution returned by the SA-2 algorithm is always the same, no matter which initial point is used.

To compare the present method with an optimization based method that uses convex partitioning, one could compare the computational expense of solving the same problem. In order to do this comparison, a Matlab implementation of the method described in [10] was developed. The number of flops taken to solve each pair of simple convex objects was found to be around 50 $k$flops. In the present case, object one could be broken into two convex pieces and object two into twelve. Hence, the total number of object pairs to solve the overall minimum distance problem would be 24 which translates to around 1.2 $M$flops. This figure is close to 2.5 times larger to what SA-1 takes and 6.5 times larger to what SA-2 takes.

Modern high-end PCs can perform more than 500 $M$flop per second (see, e.g., LINPACK benchmarks of [19]). Although more extended timing test have to be done, preliminary results suggest that the SA-2 algorithm, if implemented in a lower-level language such as C, might have an execution time of less than 1 ms to solve the problem shown here.

<table>
<thead>
<tr>
<th>$\mathbf{X}_1^0$</th>
<th>$d_0$</th>
<th>$\mathbf{X}_1^*$</th>
<th>$d^*$</th>
<th>$k$flops</th>
<th>$n^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA-1 $X_1^0$</td>
<td>4.16</td>
<td>$X_1^*$</td>
<td>0.116</td>
<td>486</td>
<td>4035</td>
</tr>
<tr>
<td>SA-1 $X_2^0$</td>
<td>4.16</td>
<td>$X_2^*$</td>
<td>0.116</td>
<td>171</td>
<td>2</td>
</tr>
<tr>
<td>SA-2 $X_3^0$</td>
<td>1.35</td>
<td>$X_2^*$</td>
<td>0.116</td>
<td>487</td>
<td>1458</td>
</tr>
<tr>
<td>SA-2 $X_2^0$</td>
<td>1.35</td>
<td>$X_1^*$</td>
<td>0.116</td>
<td>180</td>
<td>21</td>
</tr>
<tr>
<td>SA-1 $X_3^0$</td>
<td>2.11</td>
<td>$X_3^*$</td>
<td>0.190</td>
<td>485</td>
<td>4165</td>
</tr>
<tr>
<td>SA-2 $X_2^0$</td>
<td>2.11</td>
<td>$X_1^*$</td>
<td>0.116</td>
<td>184</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 1. Results for SA-1 and SA-2 runs for objects in position A.

6 Conclusions

A new method to solve the minimum distance problem between concave objects was presented. The method involves the use of global optimization techniques such as Simulated Annealing.
The optimization process is converted to a combinatorial process where the objective is to find the combination of points, one on the surface of each body, that minimizes the Euclidean distance between them.

Furthermore, if the points are distributed on a mesh, the convergence process is greatly improved by using a local optimization (based on a gradient or steepest descent approach) inside the global optimization method.

Currently, the authors are working on improving the accuracy of the results which is now a function of the mesh size and the distribution of its nodes. One of the proposed solutions is to use the global optimization method presented in this paper as a pre-processing algorithm at each iteration. The solution point of the global search, which locates the region where the exact minimum occurs, would then be passed to a local minimum distance method which will find the exact solution, within machine precision, of the minimum distance problem.

**Acknowledgment**

The authors would like to acknowledge financial support of MD Robotics and the Canadian Space Agency in the development of this work through an NSERC strategic grant.

**References**


