Sub-Object Pruning Algorithm: A Novel Pruning Strategy for Distance Determination in Complex Dynamic Environments

Raja Uppuluri[1], Juan A. Carretero[2]

[1] *University of New Brunswick, Dept. of Mechanical Engineering, Raja.Uppuluri@unb.ca*

[2] *University of New Brunswick, Dept. of Mechanical Engineering, Juan.Carretero@unb.ca*

Computing the minimum distance between objects is known to be a complex problem particularly in compact dynamic environments. Determining the minimum distance between complex objects has been solved by many different authors. Some methods rely on computational geometry techniques, while others rely on numerical optimization techniques. Most algorithms can only deal with convex objects and thus need to be partitioned concave objects into smaller purely-convex sub-objects. This usually results in increased run times as all pair-wise sub-object combinations need to be tested. In this paper, a two-stage distance determination method is proposed to perform precise and fast distance calculations for concave objects. In the first stage, a pruning strategy is used to obtain the closest pair of sub-objects. In the second stage, the set of closest features is used in a local optimization method to find the exact distance. Numerical results for different complex objects showing the proposed algorithm's capabilities are included. The preliminary implementation of the proposed algorithm has proven to be robust and computationally efficient.

1 INTRODUCTION

In the computer simulation of dynamical systems, distance determination algorithms are frequently used to determine whether two objects are in contact or not or to quantify how close the objects are from one another. During the last three decades, many authors have developed different distance determination and collision detection algorithms. These algorithms can be divided into three basic categories depending on the particular problem they solve: 1) collision detection, 2) approximate or exact separation distance, and 3) interference distance. In the first type, the algorithms only determine if there is interference between objects but do not quantify how far the objects are from one another [1, 2]. In the second type, the shortest distance between two objects is sought. More particularly, the point on one object that is closest to the other object and *vice versa* are sought. In the third type, algorithms quantify the amount of interference between two colliding objects which are often used to determine contact forces [3]. Following is a brief overview of the most relevant methods proposed to date.

1.1 Distance Determination and Collision Detection

The two main approaches to solve the minimum distance between two bodies are generically labelled as analytical and numerical approaches. Analytical methods are generally computationally very efficient. Unfortunately, such methods are complicated to develop and in some cases, when the objects are complex, not realistically feasible. In contrast, numerical or iterative methods are not as computationally efficient but are generally capable of solving much more complicated problems with little to no modifications to the algorithm itself. Amongst the numerical methods, a separation based on the type of geometries that the algorithms can handle is generally made. Namely, there are methods for convex objects and others for concave objects.

1.1.1 Convex Methods

One of the first distance determination algorithms is the one described in [4] where the approximate separation distance between two objects is reported as the separation distance between axis-aligned bounding boxes containing the original objects. A similar method was proposed in [1] where the objects are replaced by bounding spheres whose separation distance crudely approximates the distance between the objects they contain. Since then, many approximate and exact distance determination algorithms have been proposed in the literature. In general, most methods are variations or those described in [5], [6] and [7].
In [5], the separation distance problem is formulated as a numerical optimization problem where the goal is to minimize the distance between two points while each satisfies the sets of linear constraints defining the surfaces of each of the objects. This method was later extended in [8] to allow the algorithm to handle objects with quadratic surfaces.
On the other hand, the methods described in [6] makes use of a geometrical addition to simplify the problem. Namely the Minkowski sum is used to convert the distance problem into one where the distance between the Minkowski sum of the two polyhedral objects and a single point, is sought. This methods was later modified in [9] to improve upon its computational efficiency. Finally, in [7], a method was proposed where the Voronoi regions for each of the object features is obtained off-line. Then, at run-time, the closest object features are found by exhaustively checking whether for the object features that are fully contained within each others' Voronoi regions.

1.1.2 Concave Methods

To date, the vast majority of distance determination algorithms published in the literature are, in essence, limited to deal with purely convex polyhedra. As such, to allow these algorithms to deal with concave objects, all objects are pre-processed in order to partition them into purely convex polyhedra often called sub-objects [8]. The distance problem is then solved by performing $m \times n$ distance queries between all possible combinations of n sub-objects on object 1 and m sub-objects on object 2. These exhaustive methods have the advantage of relying on existing distance algorithms that have proven to be reliable for the convex problem. On the other hand, they have the disadvantage that the number of distance queries drastically increases as the complexity of the objects increases.

Looking to improve the computational efficiency of complex distance queries, certain works have concentrated on decomposition-free solutions. This is the case of the collision detection algorithm for non-convex bodies described in [2]. This method, being only a collision detection algorithm, does not quantify the distance between objects.
Recently, novel distance determination algorithms (not limited to convex or concave polyhedra) were developed and reported in [10, 11]. Such algorithms do not need to partition the concave geometries into purely convex pieces but rather rely on discretizing the geometry using an off-line generated surface mesh. One of these methods is the *MLSdist* which will be described in more detail in a later section as it is one of the key components of the pruning algorithm proposed in this paper.

1.2 Pruning Methods

To decrease the computational time needed to perform each collision or distance query, most distance determination algorithms use a relatively fast pre-processing stage to help reduce the total number of objects or object features to be used during the exact distance queries see for instance [12, 13, 14]. In general, these algorithms are termed pruning algorithms as they are used to pick only those features of the objects that are deemed important for the collision or distance query. By determining the exact separation distance only between the closest features, one can eliminate the unnecessary computational expense needed to process all features that are relatively far from each other [15, 16].
Pruning algorithms can be divided into the following two categories:

- *Global pruning*: where the pruning is done at the object level by eliminating the objects that are relatively far away from each other [15].

- *Local pruning*: where the pruning is done at the feature level by eliminating the features that are relatively far away from each other [17].

The essential component of most pruning algorithms is a bounding box hierarchical tree representation of the object. In general, the base of the tree corresponds to a bounding box containing the entire object and its leafs correspond to bounding boxes containing purely convex faces. Early work in the area concentrated on the use of Axis Aligned Bounding Boxes (AABB) whereas the latest works have reported on the use of Oriented Bounding Boxes (OBB) as the later have shown better result.

1.3 Contribution and organization

In this paper, a two-stage distance determination algorithm is proposed for both convex and concave objects. In the first stage, using the results from a fast but approximate distance determination algorithm, the closest pairs of object features are found. The closest pairs are then passed, one pair at the time, to an exact distance algorithm. As a result of the novel combination, fast and accurate results for the minimum distance problem are found.
The remainder of this paper is organized as follows. Section 2 briefly describes the two existing distance determination algorithms used in this work. Next, Section 3 describes how the approximate methods (Section 2.1) is used to create a novel pruning algorithm. The resulting pairs of features are then sent to the exact algorithm (Section 2.2). The capabilities

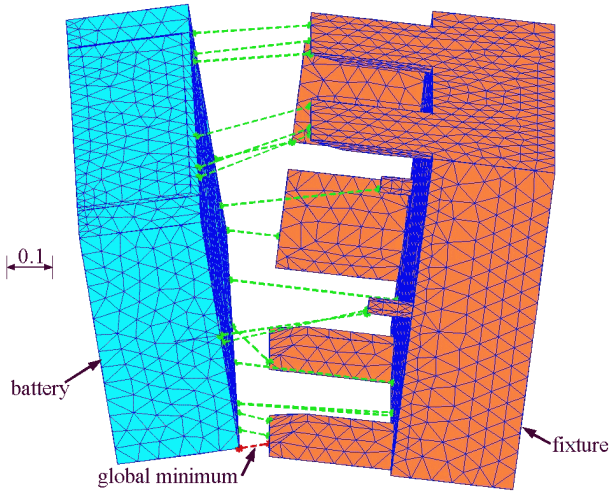

Figure 1: Finding all possible local minima using the *MLSdist* algorithm

of the overall algorithm are illustrated in Section 4 by a few numerical examples. Finally, Section 5 gives some concluding remarks.

2 BACKGROUND

The Multi Local Search algorithm (*MLSdist*) described in [11] and the exact optimization-based algorithm described in [8] (*mindist*) are used in the work presented here as part of the proposed distance determination algorithm. Therefore, brief descriptions of the *MLSdist* and *mindist* algorithms are included in this section.
Note that the results of some processes necessary for the proposed algorithm are time invariant. That is, they require data or perform processes whose results do not depend with the objects position or their orientation. As a result, these processes only need to be computed once and their results stored with the geometrical description of the objects. In what follows, these processes are termed as off-line processes. Conversely, the on-line processes or steps are those that are executed every time a distance query is performed as their results depend on the relative position and orientation of the objects.

2.1 MLSdist Algorithm

The *MLSdist* algorithm is an approximate distance determination algorithm capable of handling distance queries with objects of any degree of concavity. The main idea behind the *MLSdist* algorithm is to discretize the geometry of the object by creating a three dimensional surface mesh on the surface of the object. The separation distance problem then becomes a combinatorial one where the pair of points, one on the surface of each object, that minimizes the distance between objects is sought. Following are some details on the off-line and on-line stages of the *MLSdist* algorithm.

2.1.1 Off-Line Processes

Before any distance query is performed with the *MLSdist* algorithm, it is necessary to first pre-process each geometry in order to produce a surface mesh for each object (concave or not). Thus the geometry of each object is replaced by an off-line generated surface mesh whose nodes lie on the surface of the original object. The mesh is stored with the object

as two different matrices; one containing the coordinates of the node points and the second listing how these nodes are interconnected to its adjacent nodes on the mesh. For more details on this refer to [10, 11].

2.1.2 On-Line Processes

At run time, the *MLSdist* algorithm is used to find the closest pair of nodes, one on each object which will give the shortest distance between them. Initially the *MLSdist* algorithm randomly generates multiple starting node pairs. Node points are individually moved along the surface mesh connections until each node pair finds its local minima, *i.e.*, no combination of adjacent nodes produces a shorter distance.
Unlike convex objects, concave objects may produce more than one local minimum. As a result, many local searches as the one described above are performed at every *MLSdist* run each with different randomly generated start point pairs in order to improve the algorithm's chances to obtain the global solution. Figure 1 shows a dark line (bottom centre) joining the nodes representing the global solution of the aforementioned combinatorial optimization problem. Also in Figure 1 are light lines showing the location of other possible local minima. The minimum of all such local minima represents the separation distance returned by *MLSdist*.
Unfortunately, since the geometry of the objects is discretized by using surface meshes, the solution given by the *MLSdist* algorithm is only approximate. The accuracy of the solution will be a direct function of the mesh density where denser meshes yield better solutions. Note that if the mesh is dense, the object is replaced by a large number of mesh nodes. Consequently, the complexity of the combinatorial optimization problem increases and thus the algorithm takes longer to find the closest pair of nodes.

2.2 Mindist algorithm

The method described in [5] for solving the exact separation distance problem is formulated as a constrained numerical optimization problem. More particularly, the separation distance is formulated as

$$\underset{\mathbf{x}_1,\,\mathbf{x}_2}{\text{Minimize}} : d^2 = (\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2) \tag{1a}$$

$$\text{subject to} : A_1\mathbf{x}_1 \geq b_1 \text{ and } A_2\mathbf{x}_2 \geq b_2 \tag{1b}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ corresponded to the Cartesian coordinates of points for bodies 1 and 2, respectively. The linear inequality constraints shown as $A_i\mathbf{x}_i \geq b_i$ in equation (1b) represent a convex set of n_i half-spaces fully defining the solid geometry of object i. In equation (1b), A_i is a $n_i \times 3$ matrix where rows represent the normal to the i^{th} half-space surface and b_i is a vector $n_i \times 1$ whose entries give the exact location of the half-space boundary. To reduce the computational expense, the objective function is the square of the Euclidian distance between points $\mathbf{x}_1$ and $\mathbf{x}_2$ which can be done since minimizing d^2 is equivalent to minimizing $\sqrt{d^2}$ if d is a positive number as is the case in the separation distance problem.
When dealing with convex objects, the problem formulated in equations (1) represent a linearly constrained quadratic problem which can be solved using local optimization algorithms such as sequential quadratic programming [8].

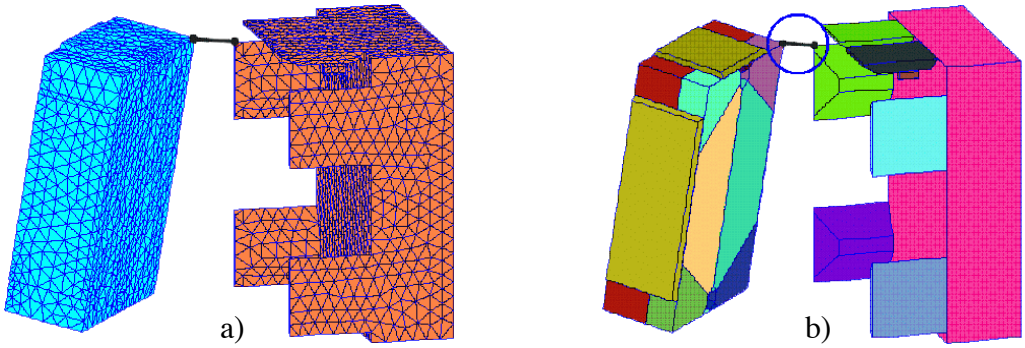

Figure 2: a) Finding closest pair of nodes by *MLSdist*. b) Matching closest pair of nodes to the particular convex sub-object of the original object from node-to-subobject database.

2.2.1 Off-Line Processes

Given that the geometries represented by the constraints in equation (1b), concave objects need to be partitioned into purely convex pieces (here called sub-objects). This process can be performed using automatic partitioning tools (*e.g.*, [18]) but is often done manually as the partitioning problem has no unique solution and human intervention is often necessary to select the best partition [11].

2.2.2 On-Line Processes

At runtime, all possible combinations of convex sub-object pairs are passed, one pair at the time, to a local optimization algorithm whose solution of each is a pair of points on the boundary of the object. The distance between such two points is then reported as the separation distance between the given sub-object pair. In general, if object 1 has m convex sub-objects and object 2 has n convex sub-objects, the outcome of the distance determination between the original concave objects consists of all $m \times n$ point pairs and all $m \times n$ distances, between all sub-object pairs. It is the smallest value amongst all $m \times n$ distances that is reported as the shortest distance between the two original concave objects. The computational time taken by *mindist* and all other exact distance determination algorithms is closely related to the total number of sub-object pairs as well as the number of surfaces representing each sub-object pair.

3 Proposed Method

Here, a distance determination algorithm (*SOPA+mindist*) is proposed to find fast and precise distance between concave objects. The developed distance determination algorithm has two stages. In the first stage the closest sub-objects pairs are found using the results from *MLSdist*, whereas in the second stage the exact distance between these sub-objects is obtained using *mindist*.

3.1 Sub-Object Pruning Algorithm

The Sub-Object Pruning Algorithm (*SOPA*) consists of two stages. In the first stage, the closest pair of nodes are found using the *MLSdist* algorithm described earlier (see Figure

2a). Next, the sub-object to which the particular solution node belongs to is found from a node-to-subobject database (see Figure 2b). This database is created off-line with the help of a pre-processing algorithm that checks which mesh node lies on the surface of which sub-object. If the node satisfies all surface constrains for a given sub-object, the node is said to be on that sub-object. Sometimes, a node may lie on more than one sub-object. Once all sets of convex sub-objects have been identified, they are all passed, one pair at the time, to the second stage of the algorithm namely the *mindist* algorithm where the exact distance between objects is computed.
Note that in simulations where multiple simultaneous contact regions may be encountered, it is necessary to use all or most local minima as sent by the *MLSdist* algorithms. These multiple minima are referred to as niches or clusters and allow to handle multiple contact regions in the form of multiple node pairs [10]. That is, the sets of sub-objects corresponding to several local minima are passed to the exact distance determination algorithm one pair at a time, and as a result, multiple contact situations may be identified and handled. Note that if a mesh node belongs to more than one sub-object, all sub-objects corresponding to such node are sent to the second stage.
Additional computational time savings can be accomplished by using coarse meshes to represent the objects. That is, since objects are pruned only to the sub-object level, it is possible to use a relatively coarse mesh to represent the objects. Using coarse mesh not only improves the time *MLSdist* needs to find the closest pair of nodes but also decreases the amount of memory required to store the meshed geometries and their related information.

3.2 Exact Distance Calculation

With a small subset of sub-objects, *mindist*, the exact distance determination algorithm, is executed. Note that to solve this second stage, other exact approaches such as analytical or exhaustive feature to feature methods could be used to determine the exact distance between the convex sub-object pair. The *mindist* algorithm is used here as the available implementation includes the ability of obtaining the interference distance described in [19] thus allowing the algorithm to be used in contact dynamics simulations [20].

4 Numerical Examples

In this section, numerical examples are presented where the minimum distance problem is solved for two different sets of complex objects while travelling through a given trajectory. The numerical tests are first performed on the scenario shown in Figure 2 consisting of a set of two objects simulating a version of the ORU battery and its fixture for International Space Station. Figure 3 represents second scenario which consists of simplified version of two hands about to do a handshake. The first example will be referred to as the BF case whereas the handshake case will be abbreviated as HS.
In the context of the distance determination algorithms, robustness and computational time are the most important criteria for evaluating the performance of a particular algorithm. That is, algorithm must **always** find the global solution by taking the least amount of computational time. Thus, the tests on the proposed algorithms presented here mainly concentrate these two criteria.
For the tests presented here and in order to evaluate the worst case scenario, no prior knowl-

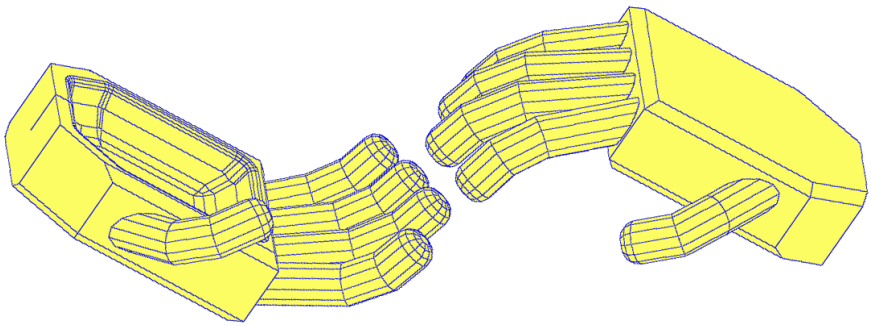

Figure 3: Complex geometry Hand-Hand (simplified version of human hand).

edge of the solution is assumed in each run. That is, although the previous time-step solution is usually a good start point for the next time step, all start points for the *MLSdist* were generated at random. Since each test is performed over many runs, the average of the particular performance measure being considered (*e.g.*, computational time) is reported while its robustness is reported as an absolute value.
Note that all the tests were performed a large number of times under similar conditions. In the present study, all tests are presented as the result of running the entire trajectory up to $10,000$ times with the same parameters but different randomly generated start point pairs. Moreover, each test trajectory consists of 151 time steps between the start and end positions for a total number 1.51 million distance queries per test.

4.1 Geometries and Trajectories

In the BF case, the battery ($31\times74\times92$ cm) was originally set at a distance of approximately 2 metres to the left of the fixture ($62 \times 86 \times 99$ cm). The convex partitioned version of the BF geometries shown in Figure 2b consist of the battery divided into 14 convex sub-objects and the fixture divided into 9 sub-objects for a total sub-object pair combination of 126. The trajectory for the BF case presented here consists of both linear and angular motions where at the end of the trajectory the battery is inserted in the fixture.
As explained earlier, the HS case consists of a set of two hands where one hand ($131 \times 185 \times 90$ mm). The convex partitioned version of the hand is divided into 92 purely convex sub-objects for a total sub-object pair combination of $8,464$. Trajectory in the HS case consists of only linear motion starting at a distance of approximately 120 mm from one another and ending close a handshake configuration. Figure 3 shows the hands in a point in the middle of the trajectory.

4.2 Fine and Coarse Mesh Results

After several tests, it was found that 68% for the BF case and 61% for the HS case of the total computational time is consumed by the *MLSdist* algorithm. In order to reduce the computational time taken in this first stage, the objects can be replaced by relatively small number of nodes. Here, when the objects are replaced by a relatively small number of nodes the case will be referred to as a "coarse mesh" case. Conversely, cases with relatively large number of nodes will be referred to as cases with "fine mesh". Table 1 shows the number of nodes on the surface mesh for each of the geometries as a function of the mesh size. Table 1 also shows the number of surfaces and sub-objects for each object.

Table 1: Details on the battery and fixture geometries.

	Battery	Fixture	Hand
Convex pieces	14	9	92
Faces	97	64	1058
Nodes in fine mesh	1,241	1,842	2129
Nodes in coarse mesh	209	308	839

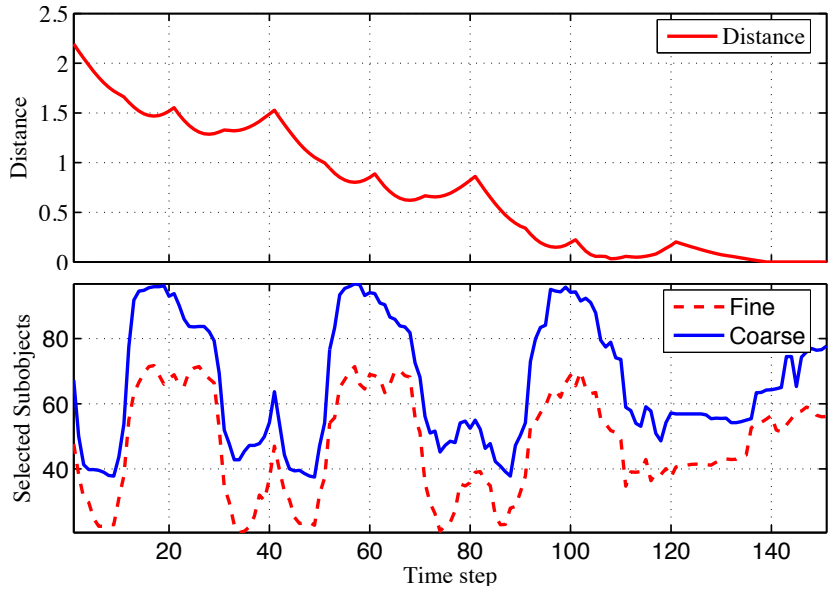

Figure 4: 1) Distance (in metres) between Battery and Fixture for the BF case using *SOPA+mindist*. 2) Number of sub-object pairs sent to the second stage of the algorithm for both coarse and fine meshes.

It was noticed that the number of surfaces/sub-objects being sent to the second stage of the algorithm was higher for the coarse meshed objects than for the fine meshed objects due to more mesh nodes belonging to more than one sub-object. On the other hand, due to the smaller number of nodes on the coarsely meshed objects, the computational time required by the *MLSdist* was greatly reduced. It was also found that a smaller number of randomly generated start points are sufficient for the cases using the coarse meshes as compared to those with fine meshes.
In Figures 4 and 5, the upper graph shows the time history of the separation distance between objects for the BF case and the HS case, respectively. On the other hand, the lower portion of Figures 4 and 5 shows the number of sub-object pairs sent to *mindist* algorithm for the same two cases.

4.3 Comparison between SOPA+mindist and the exhaustive mindist algorithm

Each convex sub-object for the BF and HS cases has an average close to 7 and 11.5 faces, respectively. As a result of all the sub-object combinations, the total number of surfaces passed to the 126 distance queries is $1,832$ or an average close to 15 faces per distance query for the BF case. At the same time, the total number of surfaces passed to *mindist* in the $8,464$ minimum distance problems is $194,672$ or an average of approximately 23 per

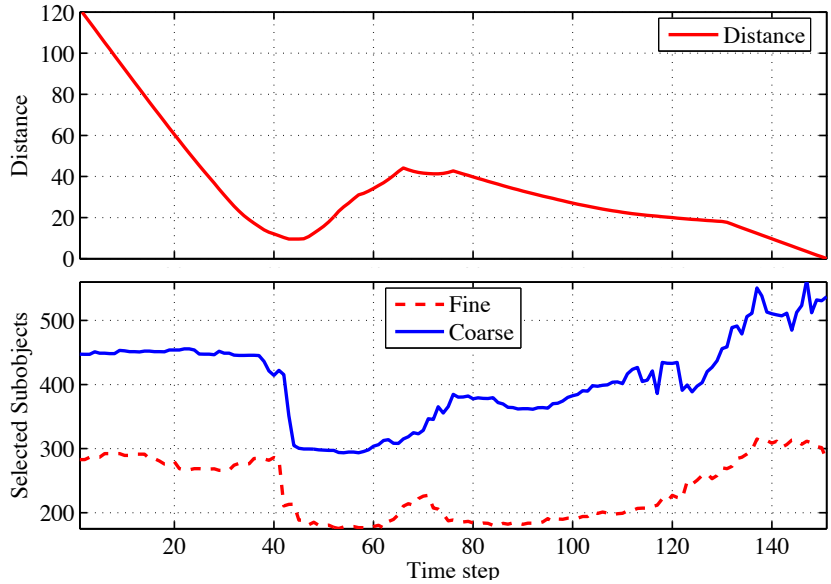

Figure 5: 1) Distance (in mm) between Hands in HF case using *SOPA+mindist*. 2) Number of sub-object pairs sent to the second stage of the algorithm for both coarse and fine meshes.

Table 2: Timing test results for *SOPA+mindist*, and *mindist*.

	BF			HS		
	time [ms]	Sub	Sur	time [ms]	Sub	Sur
mindist	4.2	126	1,832	422	8,464	194,672
SOPA+mindist fine	4.5	48	712	49	235	11,803
SOPA+mindist coarse	3.5	67	965	48	410	21,441

distance query for HS case. Note that in the current trajectory most of times the local and global minima are located on the fingertips whose number of surfaces are 82 faces per sub-object. Because of that the average number of surfaces used per distance query in the HS case is often greater than 50.

The difference in the separation distance between *SOPA+mindist* and *mindist* algorithms was found to be in the order of 10^{-9} to 10^{-17} for both coarse and fine meshed objects[3]. This indicates the *SOPA+mindist* and *mindist* algorithms give equivalent results as the termination tolerance used for the local optimization algorithm within *mindist* is set to 10^{-6} in both cases. This, suggests the first stage of the proposed algorithm is successfully passing the correct sets of sub-objects to the exact distance query thus having a robustness of 100%.

Since the numerical results from both algorithms are equivalent, the next most important piece of information is related to timing. In Table 2, the time in milliseconds, the number of surfaces (Sur) and the number of sub-objects (Sub) used by *mindist* and *SOPA+mindist* are reported. All timing tests were performed on a 2.8 GHz PC with 512 MB of RAM. It can be seen there that the fastest overall is the *SOPA+mindist* using a relatively coarse mesh. Its speed was close to 1/10 of the time taken by *mindist* for the complicated scenario whereas it was only 20% faster than *mindist* in the simple BF case.

[3]Note that the units are dependent on those used to represent the objects. That is, the ORU battery and fixture problem have dimensions in metres whereas the handshake has dimensions of millimetres.

5 CONCLUSIONS

A two stage distance determination algorithm was developed to find the distance between convex or concave objects . In the first stage, *SOPA* is used to obtain the closest features (convex sub-objects) between a pair of concave objects. Then, a local optimization algorithm is used to determine the exact distance between objects.
It was shown that the *SOPA* uses an off-line generated surface mesh to reduce the problem's complexity. When the objects are replaced by coarse mesh the computational time taken by the *SOPA* is reduced considerably, while maintaining accuracy. Due to the use of *SOPA* as pruning algorithm, number of sub-objects pairs used by *mindist* algorithm is reduced considerably (50% in relatively simple cases whereas the reduction was up to 95% more complicated scenarios). The results showed that as the complexity of the object increases, the time taken to find the distance between objects by *SOPA+mindist* only increases by a small amount whereas time taken for *mindist* is increasing in quadratically.

6 ACKNOWLEDGMENTS

MD Robotics and Dr. M. A. Nahon (Department of Mechanical Engineering, McGill University) are acknowledged for allowing the authors to use their *mindist* algorithm to perform the numerical tests presented in this paper. Also, the financial support from NSERC, the Canadian Space Agency and MD Robotics is greatly acknowledged.

BIBLIOGRAPHY

[1] J. W. Boyse, "Interference detection among solids and surfaces", *Communications of the ACM*, vol. 22, no. 1, pp. 3–9, 1979.

[2] F. Thomas and C. Torras, "Interference detection between non-convex polyhedra revisited with a practical aim", in *Proc. of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, California, USA, May 1994, pp. 587–594.

[3] O. Ma, "Contact dynamics modelling for the simulation of the space station manipulators handling payloads", in *Proceedings of the 1995 International Conference on Robotics and Automation*, Nagoya, Japan, May 1995, vol. 2, pp. 1252–1258.

[4] K. A. Maruyama, "A procedure to determine intersection between polyhedral objects", *Int. J. of Computer and Inf. Sciences*, vol. 1, no. 3, pp. 255–266, 1972.

[5] J. E. Bobrow, "A direct minimization approach for obtaining the distance between convex polyhedra", *Int. J. of Robotics Research*, vol. 8, no. 3, pp. 65–76, June 1989.

[6] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space", *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[7] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance determination", in *Proceedings of the 1991 IEEE Conference on Robotics and Automation*, Sacramento, California, USA, April 1991, pp. 1008–1014.

[8] O. Ma and M. Nahon, "A general method for computing the distance between two moving objects using optimization techniques", in *Proc. of ASME Conf. on Advances in Design and Automation*, Phoenix, Arizona, USA, 1992, vol. 1, pp. 109–117.

[9] S. Cameron, "Enhancing GJK: Computing minimum distance and penetration distances between convex polyhedra", in *Proceedings of the 1997 IEEE Conference on Robotics and Automation*, Albuquerque, New Mexico, USA, April 1997, p. pp. 6.

[10] J. A. Carretero, M. A. Nahon, and O. Ma, "Using genetic algorithms with niche formation to solve the minimum distance problem amongst concave objects", in *Proceedings of the 2002 ASME Design Engineering Technical Conference*, Montréal, Québec, Canada, September 29 - October 2 2002.

[11] J. A. Carretero, *Distance determination algorithms for convex and concave objects*, PhD Dissertation, Dept. of Mech. Eng., University of Victoria, Canada, May 2002.

[12] B. Kim and J. Rossignac, "Collision prediction", *Journal of Computing and Information Science in Engineering*, vol. 3, no. 4, pp. 295–301, December 2003.

[13] M. Lin and D. Manocha, *Collision and proximity queries*, chapter 35 in Handbook of Discrete and Computational Geometry, Second Ed., pp. 787–808, CRC Press, 2003.

[14] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models", in *Proceedings of the ACM Symposium on Solid Modeling and Applications*, Genova, Italy, June 2004.

[15] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi, "I-COLLIDE: An interactive and exact collision detection system for large-scaled environments", in *Proceedings of ACM Int. 3D Graphics Conference*, 1995, pp. 189–196.

[16] S. Gottschalk, M. Lin, and D. Manocha, " OBB-tree: A hierarchical structure for rapid interference detection", *Computer Graphics, Annual Conference Series (SIGGRAPH '96 Proceedings)*, vol. 30, pp. 171–180, 1996.

[17] J. A. Carretero and R. Uppuluri, "A novel optimization-based pruning strategy for concave minimum distance problems", in *Proceedings of the 2004 CSME Forum, London, Ontario*, London, Ontario, Canada, June 1-4 2004.

[18] B. Joe, "Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions", *Int. J. for Num. Methods in Eng*, vol. 37, pp. 693–713, 1994.

[19] M. Nahon, "Determination of the interference distance between two objects using optimization techniques", in *Proceedings of ASME Conference on Advances in Design Automation*, Minneapolis, Minnesota, USA, 1994, vol. 1, pp. 1–6.

[20] O. Ma, K. Buhariwala, N. Roger, J. MacLean, and R. Carr, "MDSF - a generic development and simulation facility for flexible, complex robotic systems", *Robotica*, vol. 15, pp. 49–62, 1997.