Proc. IEEE Int. Conf. on Robotics and Automation, 1990. pp. 338-343

The Use of Awareness in Collision Prediction

André Foisy Vincent Hayward Stéphane Aubry

McGill Research Center for Intelligent Machines
3480 University Street, Montréal, Québec Canada H3A 2A7

Abstract

We consider a world made up of a collection of objects which are all moving with respect to each other. We wish to design a system capable to *report* and to *predict* all possible object collisions; given that, all relevant information is available in due time. Previous approaches are based on the notion of a distance function that reflects the closest distance between objects in the world at any given instant in time. By explicitly including time in the representation, we describe an algorithm based on the shortest possible time before the next possible collision. The algorithm deals with all pairwise interactions between objects, sorts the pairs with respect to their predicted collision time, and maintains the most likely-to-collide pairs at the top of a stack. A new kind of hierarchy in the representation of the world is thus introduced. To find the shortest possible time before a collision, we constrain the trajectory of objects by imposing bounds on the objects' acceleration and velocity. All interacting pairs are classified into buckets that reflect the imminence of the collision. The computing cost is kept constant by reclassifying only one pair from each bucket at each time sample.

1 Introduction

We consider a world made up of a collection of objects which are all moving with respect to each other. We wish to design a system capable to *report* and to *predict* all possible collisions given that all relevant information is available in due time.

The distance function plays a central role in real-time path planning [11] and in collision detection [12]. Its objective is to compute the closest interacting objects which are assumed to be the most relevant interacting pair in the environment. Although a lot of attention has been devoted to computing the distance function in 2D and 3D for curved surfaces, polyhedral objects, etc; [2, 7, 8, 9] less attention has been paid to the case of moving objects in the environment [4, 5, 10].

The instantaneous position of the objects is not sufficient to fully reflect the imminence of possible collisions. For that reason, we propose to use instantaneous velocities, as well as velocity and acceleration bounds. This type of information should be available either from a pre-determined model or from sensors. The intuitive motivation of the algorithm presented in this paper is illustrated in figure 1, where the "imminence" of a pair reflects the imminence of a collision for that pair.

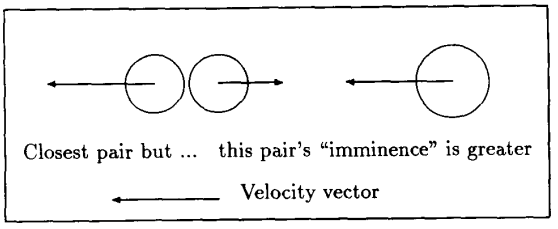

Figure 1: Closest pair, on the left, *vs* most imminent pair, on the right.

Instead of computing only the shortest distance between all objects, our approach is to evaluate τ, the *shortest possible time* before the next possible collision. A brute force approach to this problem leads to a very simple algorithm. Let N be the number of objects. At each time sample, compute the $O(N^2)$ values for τ (one for each pair of objects) and find the smallest τ. If τ is smaller than the time sample, then we predict a possible collision. This algorithm takes $O(N^2)$ computations at each time sample, which is of course unacceptable in many practical situations.

Our intuition tells us that we evaluate all the perceived relationships with the objects which surround us and we grade them on a scale of imminence or *awareness*. According to that scale, we dispense a variable amount of attention to each relationship. In addition, we dynamically adjust our awareness of the state of affairs as the situation evolves. Thus, it seems that our measure of awareness conveys the frequency at which we update a particular element of our knowledge of the world according to its importance for the task at hand. The ideas just discussed are outlined below in the form of an algorithm called the Dynamic Awareness Algorithm or DAA which explicitly includes time in the world representation. Explicit time representations have already been proposed for collision avoidance, for example in the context of planning motions in time-varying environments [10].

Although the discussion so far, and in the rest of the paper, is concerned with the prediction of possible collisions,

we believe DAA to be a general way of dealing with dynamic complex situations. Nevertheless, the particular problem of collision prediction provides us with a framework which is both time particularly relevant to spatial reasoning problems in robotics and provides a basis to illustrate and discuss DAA.

In brief, the algorithm proceeds as follows. The entire set of $O(N^2)$ pairs is considered at initialization time. A measure of awareness, τ, is computed for each pair in terms of mutual distance, instantaneous mutual velocity, and acceleration and velocity bounds. All pairs are partitioned according to the measure of awareness and put into equivalence classes of exponentially increasing cardinality. The classes are implemented as a set of arrays that we call buckets. The pairs in the smallest buckets are sampled more frequently. Pairs percolate from bucket to bucket according to the evolving value of τ.

Without loss of generality, we detail this algorithm in the case of a world consisting of balls. For a polyhedral world, a conservative estimate of τ can always be obtained by covering the objects with spheres.

Section 2 gives a formal outline of the methodology. Section 3 shows how to compute τ. Section 4 describes the updating mechanism for the data structures. Section 5 describes the complete DAA algorithm. In section 6, we discuss the algorithm's complexity and the efficiency with which it keeps track of possible collisions. Section 7 provides a simple example of an application of the algorithm. And in conclusion, we present some possible extensions.

2 Spatio-Temporal Description

Suppose we are given a set $R = \{R_1, \ldots, R_N\}$ of N moving objects and a *conservative* covering $\mathcal{R}$ of the elements of R. (We say that $\mathcal{R}$ is conservative if and only if the closure of the elements of $\mathcal{R}$ is a superset of the closure of the elements of R). We assume in the following that the size of $\mathcal{R}$ is $O(N)$.

Suppose that to every $r \in \mathcal{R}$ we can associate a coordinate frame. That frame then defines a position function $\vec{f}_r(t)$ which we assume to be twice-differentiable with respect to time in a given interval $[0, T]$.

Let $P = \{(r_i, r_j) \mid r_i, r_j \in \mathcal{R},\ i \leq j,\ i, j \in \{1, \ldots, N\}\}$. To each element $p = (r_1, r_2) \in P$ we can associate a relative position function $\vec{x}_p(t) = \vec{f}_{r_2}(t) - \vec{f}_{r_1}(t)$, a relative velocity function $\vec{v}_p(t) = d\vec{x}_p/dt$, a relative acceleration function $\vec{a}_p(t) = d^2\vec{x}_p/dt^2$, all defined on $[0, T]$.

If p is made up of point objects, we say that p generates a collision at time t if and only if $\vec{x}_p(t) = \vec{0}$. In practice however, we consider that a collision occurs at time t whenever $\|\vec{x}_p(t)\| \leq \epsilon_p$, where ϵ_p is an arbitrary value for the minimum safe distance between elements of a pair.

Of course, if $\vec{x}_p(t)$ is precisely known, all collisions may be detected analytically. In general however, such perfect knowledge is not available, and even if it were, the determination of all collisions would be computationally prohibitive because of the large cardinality of P.

An alternative is then to test the norm of $\vec{x}_p(t)$ only at discrete time intervals. In order to minimize the number of such tests, it is desirable to perform them primarily for the elements of P which are *more likely* to lead to a collision. A function based on the pair's relative position function and its derivatives, with the Euclidean distance as a norm, is a candidate for such a measure of likelyhood.

The following relative motion equation always holds for all pairs:

$$\vec{x}_p(t) = \left(\int_{t_0}^{t}\left\{\int_{t_0}^{t}[\vec{a}_p(t)]\,dt + \vec{v}_p(t_0)\right\}dt\right) + \vec{x}_p(t_0). \quad (1)$$

Equation (1) can be used to determine an underestimate for the smallest time that can elapse before a collision occurs. The object is then to find the time τ, such that

$$\tau(p, t_0) = \min_{t \geq t_0}(t - t_0), \quad s.t. \quad \|\vec{x}_p(t)\| \leq \epsilon_p, \quad (2)$$

for all possible relative trajectories, subject to the same conditions as above. In the following section we develop the awareness measure τ.

3 Calculating the Measure τ

We first consider the case without a speed bound. From elementary mechanics we know that the higher the acceleration of a body, the further it travels. Therefore, we want the acceleration norm constant and maximum. Moreover, the particle travels furthest when the acceleration is constant in the direction which corresponds to the direction of motion.

Hence, setting $\vec{x}_p(t_0) = \vec{x}_0$, $\vec{v}_p(t_0) = \vec{v}_0$, dropping the reference to a particular pair p, setting the magnitude of $\vec{a}$ to A, and setting $t_0 = 0$, equation (1) becomes:

$$\vec{x}(t) = \left(\int_0^t \int_0^t \vec{a}\,dtdt\right) + \vec{v}_0\,t + \vec{x}_0 = \frac{1}{2}\vec{a}t^2 + \vec{v}_0\,t + \vec{x}_0. \quad (3)$$

From equation (3), the loci of feasible relative positions at time t can be represented by a sphere S centered at $\vec{x}_0 + \vec{v}_0 t$ and of radius $\frac{1}{2}At^2$. The boundary of S is the loci of relative positions attained when the relative motion has maximum constant acceleration. Since the radius of S grows with the square of t while its center shifts linearly, it follows that any point in space is reachable given a long enough period of time (see figure 2).

A collision *may* occur during the time interval $[0, \Delta t]$ if and only if

$$\exists t \in [0, \Delta t], s.t. \|\vec{x}(t)\| \leq \epsilon, \quad (4)$$

where Δt is the time increment before the update.

In all cases, the earliest time τ at which the above inequality can be verified occurs when $\vec{a}$ is collinear with $\vec{x}_0 + \vec{v}_0\tau$ and of opposite direction (see figure 3). Equation 4 becomes:

$$\|\vec{v}_0\tau + \vec{x}_0\| - \frac{A}{2}\tau^2 \leq \epsilon. \quad (5)$$

Rearranging terms, squaring and solving for τ, we find that the earliest possible collision time is the smallest positive root of the fourth-degree polynomial:

$$-\frac{A^2}{4}\tau^4 + (\|\vec{v}_0\|^2 - \epsilon A)\tau^2 + 2(\vec{v}_0 \cdot \vec{x}_0)\tau + (\|\vec{x}_0\|^2 - \epsilon^2) = 0. \quad (6)$$

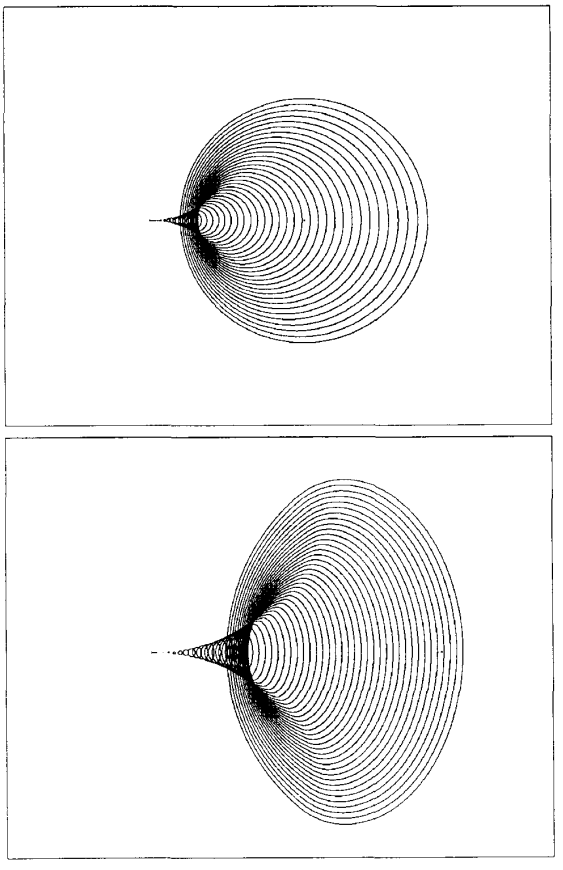

Figure 2: 2-D projections of $S(t)$; (top) without speed bound and (bottom) with speed bound.

In summary, when there is no speed bound, a *sufficient* condition for a pair not to generate a collision is that τ, as defined in equation 6, be greater than Δt.

We now include a bound V on the relative speed. As above, we can represent the loci of feasible positions at time t by the interior of a closed geometric figure S' (see figure 2). Of course, if the relative position function and its derivatives are such that the maximum speed is not achieved over a given time interval, then the results of the previous paragraphs hold over that time interval.

Suppose however that the maximum speed is achieved over a certain time interval. From elementary mechanics, the relative motion generating the earliest possible collision is split into two components: the *acceleration component*, during which the movement has maximum constant acceleration $\vec{a}$, of magnitude A, and the *velocity component*, during which the movement has zero acceleration and maximum speed $\vec{v}$, of magnitude V.

In order to determine the value of τ in the bounded speed case, we first determine T, the time at which the relative motion reaches speed V.

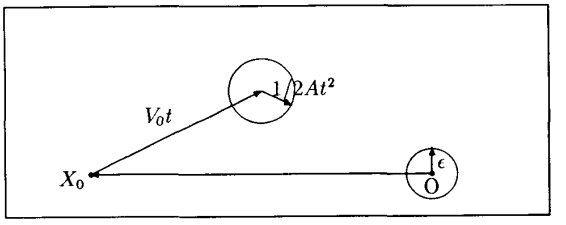

Figure 3: Graphical illustration of the four terms of equation 5 for a given increment Δt. O is the origin.

In this case, a collision may occur if and only if

$$\exists t \in [0, \Delta t], s.t. \|\vec{x}(t)\| \leq \epsilon, t \geq T, \tag{7}$$

and, as before, the earliest time τ at which the above inequality can be verified occurs when $\vec{a}$ is collinear with $\vec{x}_0 + \vec{v}_0\tau$ and of opposite direction.

However, because T is a function of $\vec{a}$, whose support is not known *a priori*, τ is still not completely determined. We need an additional equation which conveys the fact that the two conditions on the motion components are only verified when $\vec{v}(T)$ points towards the origin (the other object). Hence $\vec{v}(T)$ and $\vec{x}(T)$ are collinear.

The net effect of including the velocity bound in our calculations is to reduce the loci of feasible relative positions attainable after a given time interval. This is desirable since it makes the measure less "conservative". Namely, the more refined the awareness measure, the less conservative it is, and the fewer the false alarms. False alarms are characterized by the belief that a collision *may* occur, according to the awareness measure, where in fact better knowledge of the pair's actual relative trajectory is necessary to determine unambiguously whether a collision occurs. More involved computations is the price to pay for a less conservative measure.

4 Sequencing Data Structure

Given a Δt time increment, we can organize the $O(N^2)$ pairs into a time-varying sequencing structure $A(t)$ that indicates the awareness of possible collisions. The structure $A(t)$ will serve the following two purposes:

1. To determine the most likely elements of P (section 2) that may generate a collision. Consequently, we can perform a thorough collision check for only those elements, thereby reducing the number of computations.

2. To sequence the collision checks among elements of P which are equally likely to generate a collision, thereby keeping the computational load nearly constant.

In order to implement the two above-stated purposes, we partition the pairs into equivalence classes of unequal cardinality, the idea being to group pairs having similar "collision

imminence". The measure τ will be used to perform such a partition.

We conjecture that asymptotically exponential schemes are adequate. Here we propose a binary partitioning scheme where the cardinality of each bucket is a power of 2.

Suppose the cardinality of P is M, and $\lceil \log_2(M) \rceil = L$. Using τ as the key, we initially perform a partial sort of the pairs such that they are inserted into an array $A(t_0)$ whose elements are numbered $p_1, \ldots, p_M$. The partial sort is such that:

$$\forall k \in \{1, \ldots, (L-1)\},\ \forall i \in \{1, \ldots, 2^k - 1\},$$
$$\forall j \in \{2^k, \ldots, M\},\ \tau(p_i, t_0) \leq \tau(p_j, t_0). \quad (8)$$

In other words, the value of the key τ for any of the first $2^k - 1$ elements must be less than that of any of the remaining elements of the array. Each set $\{p_{2^{i-1}}, \ldots, p_{2^i-1}\}$ of pairs is called a bucket and is denoted by B_i. The cardinality of B_i is 2^{i-1} and there are L buckets. Bucket B_1 contains one pair only and bucket B_L contains $M - 2^{\lfloor \log_2 M \rfloor}$ pairs. We say that a bucket B_j is *lower* than bucket B_i if and only if $j > i$. Finally, we call $N(p)$ the bucket number in which pair p is.

We will at every time interval test only one pair from each bucket. Hence, we check L pairs that will be held in a structure $W(t)$ (see below). Since there are fewer pairs in the higher buckets, those pairs will be tested more often: this implements purpose 1 above. Furthermore, the computational load will remain constant at each step, since the number of selected pairs is constant: this implements purpose 2. Since the pairs from the lower buckets are assumed not to be as likely to collide as the ones in the higher buckets, they require less frequent verifcations.

5 The Algorithm

We assume that $A(t_0)$ has been properly initialized as above.

Let $W(t_n)$ be the array of pairs to be examined at a given time $t_n = (\Delta t)n$, where

$$W(t_n) = \langle p_1, p_{2^1+(n \bmod 2^1)}, \ldots, p_{2^L+(n \bmod 2^L)} \rangle. \quad (9)$$

Note that for the binary partitioning scheme the last element of the above set does not always exist for all values of n. Each bucket of $A(t_n)$ contributes one pair to $W(t_n)$.

Once the pairs are selected, we read in the current values for $\vec{x}_0$ and $\vec{v}_0$, which we assume can be made available in real-time from sensor readings. Then, we recalculate the value of τ for each element of $W(t_n)$ using one of the methods outlined in section 3.

The elements of $W(t_n)$ are sorted according to the new value of τ, and then replaced in the buckets in such a way that the ordering in $W(t_n)$ is preserved in $A(t_n)$. Finally, the value of n is incremented.

Reporting a collision depends on τ and on the cardinality of $B_{N(p)}$. For example, suppose the following holds:

$$\tau(p, t_n) > 2^{N(p)-1} \Delta t. \quad (10)$$

Equation 10 means that p is guaranteed not to generate a collision until it is updated. That update will not occur until a number of time increments, equal to the cardinality of $B_{N(p)}$, elapses. Hence no further check is necessary. Obviously, since this test is immediate, it is to be hoped that a large number of pairs will satisfy that property. If, on the other hand, equation 10 does not hold, we say that p flags the possibility of a collision. Whether a collision is indeed imminent can then be checked by an exact collision detection module [3, 6].

6 Complexity Analysis

The method's complexity is driven by the number of pairs. There are originally $M = O(N^2)$ pairs, where N is the number of spheres.

The original pre-processing partitioning step using τ takes $O(N^2)$. This can be seen considering that $A(t_0)$ is obtained by repeated application of the linear-complexity median algorithm over a decreasing geometric series, the size of the input being $O(N^2)$[1].

At every time increment, we need to consider one pair from each bucket. There are $O(\log N^2) = O(\log N)$ buckets. If we choose to update the array using a true sorting algorithm, the on-line complexity is $O(\log N \log\log N)$, while if a partial sort such as that performed at preprocessing time is chosen, the on-line complexity becomes $O(\log N)$. Intuitively, the total sort is better because it performs the best possible ordering of the pairs of objects.

We say that the algorithm is *complete* if it can guarantee that all collisions are flagged. Let Q be the time necessary to process the $O(\log N)$ pairs in the main loop of the algorithm. A *sufficient* condition for the algorithm to be complete is that

$$Q < \Delta t. \quad (11)$$

Given a limited computational capacity, this cannot in general be guaranteed. On the other hand, the algorithm is capable to report whether its capacity is exceeded.

If Δt is chosen too small, equation 11 shows that the processor may not be able to perform all the overhead associated with every step: memory accesses, re-computation of τ, and resorting of the pairs.

If however Δt is chosen too large, equation 10 shows that too many calls (false alarms) to the exact collision detector may be made. These calls are in general costly because they depend on the actual geometry of the elements of a pair.

7 Example

In this section we present a simple example that illustrates the validity of the algorithm. The world comprises only three objects in a 2D plane. Each object, named O_1, O_2, and O_3, is a point (zero length radius) and is represented by a dot in figure 4. The vectors in the figure represent the velocities. At each sample, the position and velocity of each object is

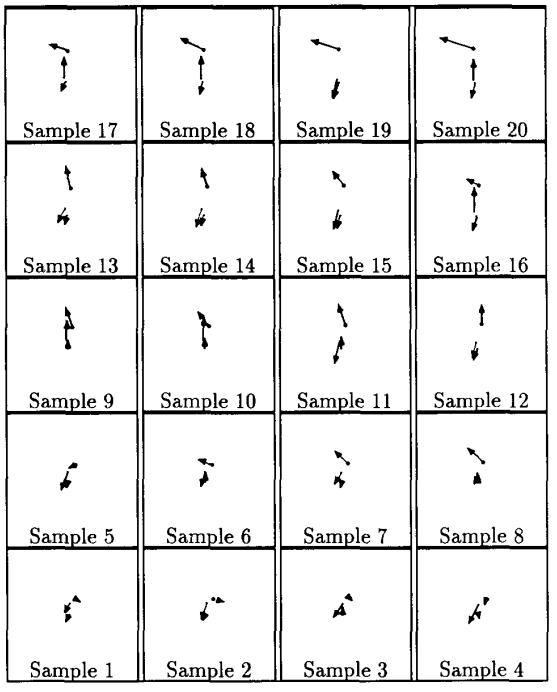

Figure 4: Example in a 2D world with 3 objects. Each dot represents an object: small dot is O_1, medium dot is O_2, and the big dot is O_3. A vector represents the velocity of an object.

updated with a randomly generated velocity variation. The time increment Δt is 0.05 units of time.

Three algorithms are implemented:

- DAA: the algorithm discussed in this paper, without a speed bound,
- full-DAA: the complete calculation of τ and total sort for all pairs,
- full-DF: the complete computation of the distance and total sort for all pairs.

The task of DAA and full-DAA is to report the pair of dots that have the smallest value of τ. The task for full-DF is to report the closest pair of dots. The first element of $A(t_n)$ of each algorithm is used to compare them. At each sample, the three algorithms agree if they all give the same most imminent possible collision (pair) in the world. The results for this particular simulation are shown in figure 4 and are summarized in the following:

- number of iterations: 20, number of objects: 3,
- (1) DAA and full-DAA agree on: 20,
- (2) DAA and full-DF agree on: 15,
- (3) full-DAA and full-DF agree on: 15.

Result (1) shows how DAA keeps track of the evolving world; i.e., the partial update is sufficient to do the same job as the complete update. Result (2) shows the difference between DAA and full-DF. In general, the algorithms will not agree if the closest objects are not going towards each other. If DAA cannot keep track of the changes, the difference between the measure τ and full-DF is seen in result (3).

The most interesting situation is when DAA and full-DF do not agree. Such a case arises in samples 1 to 5 (see figure 4). For example, the numerical values for sample 2 are:

- DAA says that the most imminent possible collision is between O_1 and O_2, where:
  - pos. O_1 is $(-0.5, 2.1)$, vel. O_1 is $(-0.9, -2.4)$,
  - pos. O_2 is $(-0.6, 0.4)$, vel. O_2 is $(-0.5, -1.1)$,
  - distance 1.76, τ is 0.95.
- Full-DF says that the most imminent possible collsion is between O_1 and O_3, where:
  - pos. O_1 is $(-0.5, 2.1)$, vel. O_1 is $(-0.9, -2.4)$,
  - pos. O_3 is $(0.5, 2.9)$, vel. O_3 is $(1.7, -0.6)$,
  - distance 1.3, τ is 3.1.

Obviously, this means that the velocities of O_1 and O_2 can create a conflict that is not suspected by DF.

After running the algorithms several times some interesting tendencies emerge:

- The measure τ throws into relief the objects which are converging. This confirms our initial intuition.
- Objects with high velocities are marked as more "dangerous"; i.e., the value of τ is small. They are the troublemakers.

8 Conclusions

In this paper we considered a supplement to the classical distance function DF: the dynamic awareness algorithm DAA. When dealing with dynamic worlds, DAA takes into account the dynamic aspect of an environment and predicts all possible collisions. Like DF, DAA can be used as a base for numerous schemes in path planning, collision avoidance, etc.

Rather than looking at a specific trajectory, DAA picks the *worst-case* trajectory by considering τ, the shortest possible time before a collision. The algorithm has a good performance because, using τ, it maintains the most probable future interaction in the environment at the top of a stack. Hence, the more information available to compute τ the better the predictions.

The naive method of complete re-classification at each time sample has a complexity of $O(N^2)$ per time sample. In contrast, DAA has an initial classification step of complexity

$O(N^2)$ and a steady state complexity of $O(\log N \log\log N)$ per time sample.

Completeness for the algorithm is defined with respect to its capacity of reporting all possible collisions. DAA is complete in the sense that it can report all failures to keep track of potential collisions.

The Dynamic Awareness Algorithm is general, and one can think of numerous possible variations:

Serial computing: With serial computers, one can easily see that buckets can be assigned to memories of increasing capacity and decreasing access speed. The pair sampling algorithm bears comparison with the type of scheduling algorithm used in time-sharing operating systems.

Parallel computing: Parallel computing could be achieved by assigning a computing unit to each bucket, or by assigning a computing unit to each pair. Note that no scheduling scheme is necessary in the latter case.

Partioning: As mentioned in section 4 the buckets' size can follow various laws, Fibonacci series, for example. Hopefully, the priority distribution of the interacting objects in the environment is exponentially decreasing, which implies that we assume that not many simultaneous important interactions take place. The partitioning strategy could require to be tuned according to applications.

Information on objects: The more available information, the better the prediction measure τ is. DAA can use various relations to sort the pairs and improve predictions. Furthermore, to get in due time the information on the most important interacting pairs in the world, DAA can be connected to a perceptual mechanism to shift the focus of attention to the important features.

The flexibility of the algorithm comes from the fact that we separated the problem of dealing with dynamical worlds into two sub-problems: classifying the pairs using the awareness measure τ and scheduling the pair updates using bucket partitioning.

An example of its use in robotics is in real-time collision avoidance for redundant manipulators [12] where DAA would replace the distance function used to evaluate the closest object to the robot.

9 Acknowledgments

Work described in this paper has been supported in part by a research contract with Spar Aerospace Ltd., Toronto, Canada, by a grant from NSERC (Natural Sciences and Engineering Research Council of Canada), and Fonds FCAR (Formation des Chercheurs et l'Aide à la Recherche, Québec).

References

[1] Aho, A.V., Hopcroft, J.E., & Ullman J.D. (1982), *Data structures and algorithms*, Reading, MA, Addison-Wesley.

[2] Ahuja, N., Chien, R.T., Yen, R., & Birdwell, N. (1980), Interference Dectection and Collision Avoidance among Three Dimensional Objects, *First Ann. Nat. Conf. on AI*, Stanford, pp. 44–48.

[3] Boyse, J. W. (1979), Interference Detection among Solids and Surfaces, *Communications of the ACM*, vol. 22, pp. 3–9.

[4] Buckley, C.E., (1989), A Foundation for the "Flexible-Trajectory" Approach to Numeric Path Planning, *Int. Journal of Robotics Research*, vol. 8, no. 3, pp. 44–64.

[5] Cameron, S., (1985), A Study of the Clash Detection Problem in Robotics, *IEEE Int. Conf. on Robotics and Automation*, St-Louis, pp. 488–493.

[6] Canny, J., (1986), Collision Detection for Moving Polyhedra, *IEEE PAMI*, vol. 8, no. 2, pp. 200–209.

[7] Faverjon, B., Tournassoud, P., (December 1988), *A Practical Approach to Motion-Planning for a Manipulator with Many Degrees of Freedom*, INRIA, Rapport de Recherche 951.

[8] Gilbert, E.G. and Johnson, D.W., (1985), Distance Functions and their Application to Robot Path Planning in the Presence of Obstacles, *IEEE J. of Robotics and Automation*, vol. 1, no. 1.

[9] Hayward, V., (1986), Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace, *IEEE Int. Conf. on Robotics and Automation*, San-Francisco, pp. 1044–1049,

[10] Kant, K., Zucker, S. W., (Fall 1986), Toward Efficient Trajectory Planning: The Path-Velocity Decomposition, *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 72–89.

[11] Khatib, O. (1986), Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The Int. J. Robotics Res.*, Vol. (5)1, pp. 90–98.

[12] Maciejewski, A.A. and Klein, C.A., (Fall 1985), Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments, *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117.