Efficient Control Law Simulation for Multiple Mobile Robots*

Brian J. Driessen
Sandia National Laboratories, Albuquerque, NM 87185-0439, bjdries@sandia.gov
Joseph D. Kotulski
Sandia National Laboratories, Albuquerque, NM 87185-1152, jdkotul@sandia.gov
Kwan S. Kwok
Sandia National Laboratories, Albuquerque, NM 87185-1004, kskwok@sandia.gov
John T. Feddema
Sandia National Laboratories, Albuquerque, NM 87185-1003, jtfedde@sandia.gov

Abstract: In this paper we consider the problem of simulating simple control laws involving large numbers of mobile robots. Such simulation can be computationally prohibitive if the number of robots is large enough, say 1 million, due to the $O(N^2)$ cost of each time step. This work therefore uses hierarchical tree-based methods for calculating the control law. These tree-based approaches have $O(N \log N)$ cost per time step, thus allowing for efficient simulation involving a large number of robots. For concreteness, a decentralized control law which involves only the distance and bearing to the closest neighbor robot will be considered. The time to calculate the control law for each robot at each time step is demonstrated to be $O(\log N)$.

1. Introduction

Control laws for each mobile robot of a large number of inexpensive mobile robots are typically decentralized and very simple because of the small amount of memory and compute-power available to each robot. See, for example, [6], [4], [1], [7], [2], and [5]. The number N of such robots can make the simulation of their control laws very expensive, i.e., $O(N^2)$. Tree-based hierarchical spatial decomposition however enables one to calculate the control law for every robot in $O(N \log N)$ time, as we shall see in Section 2 and Section 3. We will consider the prototypical case of a control law which involves only the distance and bearing to the closest neighbor robot. Clearly the naive approach to finding the closest neighbor vehicle at each time step would cost $O(N^2)$ because N distances would have to be calculated for each vehicle before the smallest distance is determined. The tree-based approach will allow us to find the closest robot to any given robot in $O(\log N)$ time.

2. The Method

As indicated in Section 1, the real problem boils down to finding the closest robot to any given robot in $O(\log N)$ time. For ease of presentation, we will consider the two-dimensional case; all methodology herein can be easily extended to the three-dimensional case.

The first step of the method is to create a tree-based hierarchical decomposition of the domain in which the robots reside. Figure 1 below illustrates such a spatial decomposition.

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

DISCLAIMER

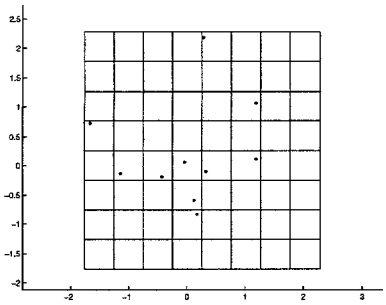

Figure 1. Illustration of Tree-Based Hierarchical Spatial Decomposition

Each dot in Figure 1 represents a mobile robot. For simplicity and without loss of generality, we consider the original domain to be a square. This domain is the parent of four subdomains which constitute the first generation. Each subdomain is further divided into four more subdomains. This process proceeds until each subdomain of the generation has no more than one robot. If the ratio of the maximum to average spatial density of the robots is upper bounded as N increases, then the number of generations of the tree can be easily seen to be $O(\log_4 N)=O(\log N)$.

The next step is to take each robot consecutively through the generations of the tree while throwing out those subdomains that could not possibly contain the closest neighbor robot. If a subdomain is thrown out, none of the children or sub-subdomains of this subdomain are considered in the next generation of the tree. It is this ability to throw out large numbers of robots at once that makes the method efficient. When we reach the final generation, we have to compare only $O(1)$ robots' distances to the robot under consideration to obtain the closest robot.

The method by which we decide what subdomains of a generation to throw out is as follows. Each remaining subdomain of a generation has a closest and farthest point to our robot. If we sketch these subdomains on a distance number line we obtain a figure analogous to Figure 2 below.

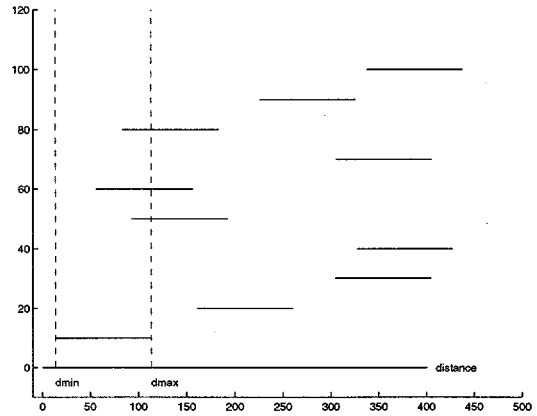

Figure 2. Graphical Illustration of Method for Throwing Out Subdomains

where the left point of each segment in Figure 2 represents the subdomain's closest point and the right point the farthest point of the subdomain. Let C denote the segment with the smallest closest-distance, denoted by *dmin* in Figure 2. Then any subdomain whose segment's interval does not overlap that of C cannot contain the closest neighbor robot. Any such subdomain is thrown out. In Figure 2, only three cells have distance intervals that overlap that of C. Thus, all but these three cells and C would be removed from the cell list.

We will now present some details of the algorithm. Suppose there exists a data structure that allows us to determine in O(1) time whether a cell contains any robots, and suppose that the time and memory required to create this structure are $\leq O(N \log N)$ and $\leq O(N)$ respectively. Suppose also that this same data structure allows us to determine which robot, if any, is located in any given cell of the Mth generation, also in O(1) time. One such data structure and its creation will be presented shortly. Let M denote the number of generations of geometrical sub-division of the original domain. Then M is the number of sub-divisions required to make each resulting square cell have no more than one robot. If m denotes which of these M generations is currently under consideration, then each cell of the mth generation is a square with sides of length $L\left(\frac{1}{2}\right)^m$, where L is the length of a side of the original square domain. Let n_m denote the number of remaining cells in the mth generation list of cells, where $n_0 = 1$. (Many cells are thrown out as we step down through the generations.) A cell of a generation will be denoted by its center-

point's location (x_c, y_c). Let (x^*, y^*) denote the position of the robot under consideration. Then the algorithm is as follows.

Loop m=1 to M (2.1)

From the list of n_{m-1} cells of the previous generation, divide each such cell into four child cells to create a list of $4n_{m-1}$ cells. (2.2)

Free the memory for the (m-1)th generation cell list. (2.3)

Remove from the list created in (2.2) all those cells that contain no robots. Recall the cost to test whether a cell has any robots is O(1), by the assumption of the existence of the required data structures. (2.4)

Calculate the shortest and farthest distance from (x^*, y^*) to each cell in the list from (2.4). (2.5)

Using the results of (2.5), remove from the cell list those cells that contain only the robot at (x^*, y^*). (Such cells will have zero distance to (x^*, y^*).) (2.6)

Find the remaining cell whose closest-point (to (x^*, y^*)) is smallest. (2.7)

Let $d_{\min}$ denote the associated smallest distance from (2.7), and let $d_{\max}$ denote the largest distance for that same cell. (2.8)

Remove from the remaining list of cells all of those cells whose shortest-distance to (x^*, y^*) is greater than $d_{\max}$. (2.9)

EndLoop (2.10)

After the loop (2.1)-(2.10) is completed, we have a list of generation-M cells where the length of this list is O(1). Each of these cells contains a single robot. Finally, of these O(1) robots, we find the one with the smallest distance to (x^*, y^*). This robot is the closest neighbor robot that we were seeking.

We will now describe the data structures, and their creation, that allow us to determine in O(1) time whether a cell of generation m, (m=1,...,M), has any robots and also what the robot index of a generation-M cell is. The data structures that were used were M+1 matrices. For each generation m, let $Q_m \in R^{2^m \times 2^m}$ denote a matrix of integers that tells the number of robots in the associated cell. The invertible mapping from the matrix's row/column indices (i,j) to the cell's center position (x_c, y_c) is given by

$$x_c = h/2 + (i-1)h + x_{\min} \quad (2.11)$$

$$y_c = h/2 + (i-1)h + y_{\min} \quad (2.12)$$

where $h = L\left(\frac{1}{2}\right)^m$ is the cell-size of the mth generation cells and $(x_{\min}, y_{\min})$ is the location of the lower left corner of the original domain. Equation (2.11) can be solve for i given x_c and (2.12) can be solved for j given y_c. In particular,

$$i = (x_c - h/2 - x_{\min})/h + 1 \quad (2.13)$$

$$j = (y_c - h/2 - y_{\min})/h + 1 \quad (2.14)$$

Finally let $P \in R^{2^m \times 2^m}$ denote a matrix whose P_{ij} value is equal to the index of the robot contained in the associated cell of the Mth generation (as determined by (2.11)-(2.14)). Let C_{mn} denote the nth entry in a list of cells for the mth generation. Each such cell has a centroid (x_c, y_c) attribute and a list-of-robots (contained in the cell) attribute $\hat{P}_{mn}$. Each cell also has a number-of-robots attribute N_{mn}.

The creation of the Q_m (m=1,...,M) and P data structures is as follows.

m=0 (2.15)

While $\max_n(N_{mn}) > 1$ (2.16)

m=m+1 (2.17)

Divide each entry in the (m-1)th list of cells to produce a list of $4n_{m-1}$ cells. (2.18)

Loop through all of the robots of each of the (m-1)th generation cells while assigning each such robot to one of the four mth-generation children cells of the (m-1)th generation cell. (2.19)

Delete all cells from the list from (2.18) that have no robots, while setting the appropriate $Q_m(i,j)$ value to 0. (2.20)

For each cell in the remaining list, set the associated $Q_m(i,j)$ value equal to the number of robots in that cell. (2.21)

EndWhile (2.22)

For each cell in the list of Mth generation cells, set the associated P_{ij} value equal to the index of the (single) robot the cell holds. (2.23)

This completes the creation of the $Q_m(i,j)$ and P.

3. Numerical Example

The example considered herein is one in which a large set of inexpensive mobile robots are to converge to a target without colliding with each other . If we let $\vec{v}$ be the velocity of the robot, then the control law takes the following form

$$\vec{v} = \frac{-C_1}{r^3}\vec{e}_1 + C_2\vec{e}_2 \qquad (3.1)$$

where the first term is a repulsive term directed away from the closest neighbor vehicle r distance units away and $\vec{e}_1$ is the bearing vector to the target. Clearly to simulate this very simple control law efficiently, one must be able to find the closest robot efficiently. See [6] for more details of this problem. This problem is what motivated the development of the method presented herein.

In this section we would like to demonstrate the O(Nlog$N$) complexity of the simulation method. The creation of the data structures in (2.15)-(2.23) is well known to require O(Nlog$N$) time and O(N) storage, based upon the theory of the well-known Barnes-Hut algorithm [3].

To our knowledge the cost of (2.1)-(2.10) is not established in the existing theory as it is a new approach to finding the closest neighbor robot. In this paper we will only demonstrate empirically that (2.1)-(2.10) has O(log$N$) run time. To do this it is sufficient to show that the average (over the M generations) number of cells in the cell list after step (2.9) is O(1). For this empirical demonstration, we considered a Gaussian distribution of robots and determined how this average number of cells per generation varied with N. Table 1 illustrates the results of this experiment. We see that this average number of cells is in fact essentially independent of N. Thus, the method of (2.1)-(2.10) shows promise of being O(log$N$). In future work we will provide a proof of the O(log$N$) complexity of (2.1)-(2.10).

Table 1. Demonstration of Algorithm Complexity

N	Avg Number of Cells After Step(2.9)
80	4.3
160	3.9
320	5.3
640	4.8
1280	3.6

4. Conclusion

In this paper we presented a method for simulating control laws involving large numbers of mobile robots. Such robots are typically very inexpensive and use very simple decentralized control laws. The prototypical case considered involved a very simple decentralized control in which the only interaction (between robots) in the control law was a function of the distance and bearing to the closest neighbor robot. Letting N denote the number of robots, the method presented allowed for the calculation of all N control laws in O(Nlog$N$) time, which is more efficient than the typical $O(N^2)$ approach. The method is a novel one that uses a hierarchical decomposition of the spatial domain containing the robots to obtain the closest neighbor robot for all robots in O(Nlog$N$) time.

References

[1] Arkin, Ronald C., "Cooperation Without Communication: Multiagent Schema-Based Robot Navigation," *Journal of Robotic Systems* 9(3), 1992, pp. 351-364.

[2] Asama, H., "Distributed Autonomous Robotic System Configured with Multiple Agents and Its Cooperative Behaviors," *Journal of Robotics and Mechatronics*, Vol. 4, No. 3, 1992.

[3] Barnes, J. and Hut. P., "A Hierarchical O(NlogN) Force-Calculation Algorithm," *Nature*, Vol. 324, No.4, December 1986, pp. 446-449.

[4] Brooks, Rodney A. and Flynn, Anita M., "Fast, Cheap and Out of Control: A Robot Invasion of the Solar System," *Journal of the British Interplanetary Society*, Vol. 42, 1989, pp. 478-485.

[5] Chen, Q., and Luh, J., "Coordination and Control of a Group of Small Mobile Robots," IEEE International Conference on Robotics and Automation, vol. 3, 1994, pp. 2315-2320.

[6] Driessen, B., Feddema, J., and Kwok, K., "Decentralized Fuzzy Control of Multiple Nonholonomic Vehicles," American Control Conference, 1998.

[7] Kube, C. and Zhang, H., "Collective Robotics: From Social Insects to Robots," *Adaptive Behavior*, Vol. 2, No. 2, pp. 189-218.