From: FLAIRS-01 Proceedings. Copyright © 2001, AAAI (www.aaai.org). All rights reserved.

A Binary Tree based Approach for the Design of Fault-Tolerant Robot Team

Haihang Sun and Robert McCartney

Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269-3555
{haihang | robert } @ engr.uconn.edu

Abstract

Task allocation and load balancing are critical in fault-tolerant robotic team design. This paper deals with the problems of allocating and reallocating tasks to multiple robots in a fault-tolerant manner. Using a tree structure to store global information about active robots and unfinished sub-tasks, each robot can reallocate itself to tasks independently when robot failure occurs in the team. Organized in full distributed manner, robots in the team respond robustly, flexibly and without negotiation with other team members to robot failure and other environment changes. Simulation results are given to show the feasibility of this approach.

Introduction

One of the most important and well-known advantages in using multiple robots is that it is possible to increase the robustness and fault-tolerance of a robotic system. It can be easier, cheaper, more flexible and more fault-tolerant to build and using several simple robots than to have a single powerful, complicated robot. In a multi-robot system, the failure of a single robot may only cause partial degradation of system performance, and the robots can be much less complex since each robot in the system is only responsible for partial fulfillment of the total task. However, several challenging issues, which do not arise in single robot systems, remain to be addressed while constructing multi-robot systems (Mataric 1995):

- How do we describe, decompose and allocate problems among a group of robots?
- What is the communication and interaction structure among robots?
- How do we achieve globally coherent and efficient solutions from the interaction of robots lacking environment information?

Problems such as these are made more difficult if we allow robots to fail. Tasks need to be reallocated, communications and interaction structure must change, the robot-robot interactions will change, and so forth. To maintain efficiency, we need to provide fault tolerance without excessive redundancy: we want to get useful work from all robots available.

Considering the limited communication bandwidth in real physical robot systems, and the goal of scaling up to large robot teams, information exchange among robots needs to be limited. Any scheme requiring large amount of communication among many robots is not acceptable.

In this paper, we consider using binary trees to keep information about tasks and available robots. Using these, each robot can allocate himself to tasks independent of other robots' decisions. Robots organized in this architecture can dynamically reallocate tasks without negotiation when robot failure occurs. We have demonstrated this approach in simulation, and give examples later in this paper.

The following section gives a brief overview of the related work in this area. The next section describes in detail our approach used for homogeneous robots. Next, we present the results of our studies, followed by a discussion of these results. Finally, we conclude the paper, discussing questions yet to be resolved.

Related Work

Various approaches for achieving fault tolerance in robot teams have been proposed in the past few years.

Parker (1994, 1998) demonstrates the ALLIANCE architecture, which is used to study fault tolerant cooperation in a heterogeneous team of largely independent, loosely coupled robots. Unlike typical behavior-based approaches, in ALLIANCE, individual robots are based on a behavior-based controller with an extension for activating behavior sets that accomplish certain tasks. Each behavior set is activated by motivational behaviors whose activations are in turn determined by using the current rates of impatience and acquiescence, as well as sensory feed back and the awareness of its teammates. When the environmental changes require the robot to take over tasks from other malfunctioning team members or to give up its own current tasks, the motivations of impatience and acquiescence allow robot system to handle individual robot

Copyright © 2001, AAAI. All rights reserved.

failure and malfunction fluidly and flexibly, leading to adaptive action selection to ensure mission completion. Experiments on physical and simulated robot systems demonstrate its qualities of robustness, fault tolerance and flexibility.

Schneider-Fontán and Mataric (1998) study a territorial approach to minimize inter-robot interference, and thus to achieve high task efficiency and fault tolerance. In the distributed clean-up and collection task, the robots are assigned individual territories which can be dynamically resized corresponding to robot failure, permitting the completion of the mission. Among the interesting features of this approach is that each robot can independently (and deterministically) allocate its own task by simply knowing which robots have failed.

Dias and Stenz (2000) describe a free market architecture with which robots coordinate to solve a given task. Each robot tries to maximize its personal profit while executing the task. The robots negotiate among themselves to minimize their costs and maximize their profits. In this way, tasks are assigned to the most appropriate robots, thus dynamically allocating tasks among robots.

The approach proposed in this paper stores global information about tasks and robots in a binary tree. This information allows individual robots to make workload balancing decisions quickly and independently, while keeping the average workload in each sub-group within a tolerable range. This makes physical cooperative multiple robot team construction feasible with limited communication: allocation information is shared, but reallocation decisions are made without negotiations among the robots.

Approach

Representation of global knowledge

In this approach, robots use balanced binary trees for the purpose of accumulation and utilization of the global knowledge about the environment.

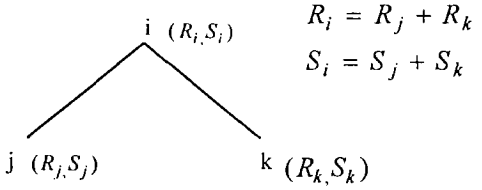

Figure 1: Binary tree structure used for storing global information.

A given task can be divided into N sub-tasks, then assigned to the leaves of our tree (if N is not a power of two we can pad the subtasks with the appropriate number of dummy subtasks that we consider as already finished. This ideally reflects a natural hierarchical decomposition, that is, the two children of a single node can reasonably be combined into a single subtask; an example of this is using a tree to partition a region so that siblings combine into a contiguous region.)

In subsequent figures, we label nodes corresponding to the indexing where node i has node $i/2$ as its parent and nodes $2i$ and $2i+1$ as its children. We assign robot and task information to the nodes as follows: each leaf k, which corresponds to a single subtask, is given the value (R_k, S_k), where R_k denotes the number of robots assigned to its task, and S_k is 1 if the task is unfinished, 0 if the task is finished. In general, the value of node i is based on the sum of the values in its children's subtrees, as illustrated in (Figure 1): for value (R_i, S_i), R_i denotes the number of active robots in the sub-tree and S_i denotes the number of unfinished tasks in the sub-tree. Figure 2 gives some example trees with values at nodes.

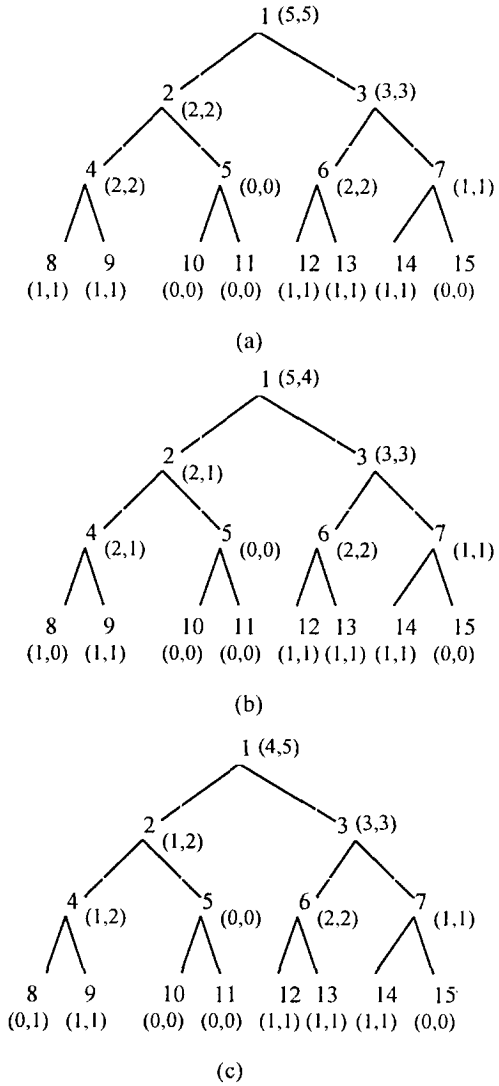

Figure 2: Global information kept by robots.

Each robot keeps a copy of the tree itself, and makes control decisions based on the tree information. When a robot changes its location or finishes a task, this information is broadcast to the other robots, as well as sufficient information for all to determine that (and where) a robot has died.

For example, in Figure 2.a, suppose the robot in Node 8 finishes the task: it changes the value of this node to (1,0), and propagates this change up the tree, resulting in the tree in Figure 2.b.

If a robot detects the failure of another robot (doesn't receive "I am alive" message for an amount of time, e.g.), it will change the robot value part of the binary tree correspondingly. Take Figure 2.a for example: if the robot in node 8 fails, the other robots will change the tree as Figure 2.c

```
i = node # of robot L;
finish =FALSE;

while ((i>1) && (!finish)) {
   if (i %2 ==0)
        j=i+1; // i is left child, j is its sibling
   else
        j=i-1; // i is right child.
  //  Check for "S_j ==0"
   if ( S_j == 0 )
      Ratio = ( ρ1 + ρ2 )/2 ;
   else
     Ratio = (S_i / R_i) / (S_j / R_j);
   if (Ratio < ρ1 ) {
      Pr1=Random(); // Pr1 ∈[0,1]
      if (Pr1<P) { //move
           robot L moves from sub-tree L to
           sub-tree L.
           Broadcast the changes to other robots.
           }
      finish =TRUE;
      }
  else if (Ratio > ρ2 )
      finish =TRUE; //keep in same place
      else
         i =i/2; // go to upper level.
  }
```

Figure 3: Dynamic task allocation algorithm

Dynamic task allocation

Whenever there is any changes in the tree, each robot reallocates itself task based on the information of the tree.

Suppose robot L is currently doing task i in leaf i. The number of total active robots in this node is R_i, the number of total unfinished tasks in this node is S_i. (R_i, S_i) denotes its node value.

Robot L compares its node value (R_i, S_i) with its sibling's node value (R_j, S_j).

If $\frac{S_i}{R_i} < \frac{S_j}{R_j} \cdot \rho_1$, the average workload in node i is less than that in node j by a factor of at least $1/\rho_1$ ($\rho_1 < 1$, ρ_1 is the load balancing threshold coefficient which defines the unbalance tolerance in the two sibling sub-trees), robot L may decide to leave node i and go to node j, in order to balance the workload between node i and node j.

If the task balance in these two nodes is comparable (equal within a factor of ρ_1, robot L will compare its parent node with its parent's sibling node, and determine in similar fashion whether it can help balance between its parent sub-tree and its parent's sibling sub-tree. This procedure is to balance the workload with its cousin nodes. Figure 3 shows the details of the algorithm.

Conflict resolving and self adaptation

It is inevitable that two or more robots may simultaneously choose to help their sibling node. In Figure 4.a, robot in node 4 and node 5 may move to their parent's sibling sub-tree at the same time, and the tree becomes as Figure 4.b.

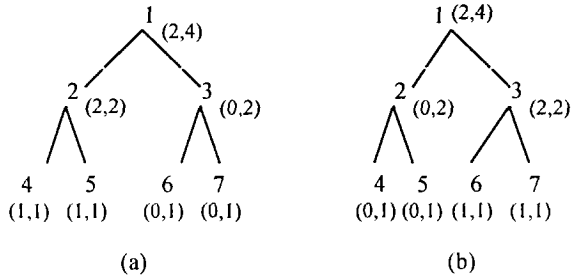

Figure 4: Robots oscillate back and forth between two states.

In the worst case, these two robots may oscillate back and forth between tasks. In order to minimize likelihood of such oscillation, we make the decision random: the move is chosen with probability p. This p defines the probability that an individual robot makes a decision to move when it can.

The move probability is determined by the number of robots that might potentially move to the new location. Considering the Figure 4 example: the robots in node 2's subtree each self-assign a p of 0.5; the probability of both robots choosing to move is 0.25, while the probability of exactly one robot moving is .5. There is also the probability of .25 that neither robot moves, but the

remaining unbalance can be addressed on the next situation evaluation.

Collapsing subtasks

One potentially bad behavior occurs when there are less robots in a subtree than tasks: a robot that is the only one assigned to a task may move to its sibling that has no robot, leaving its current task unassigned. One solution would be to prohibit such moves; if there are not enough robots, some tasks have to wait for other subtasks to finish.

In some cases this is not acceptable: in situations where subtasks never finish, or in situations where tasks cannot reasonably left unassigned. As an example of both of these, consider the task to patrol a region, which can be partitioned into disjoint regions. A possible solution is to collapse two subtasks into their parent, and to assign the robot to the parent task (that is, one robot is assigned to both subtasks). How the individual robot allocates its effort between these subtasks may be determined by the individual problem constraints.

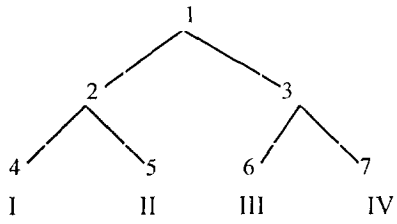

Figure 5

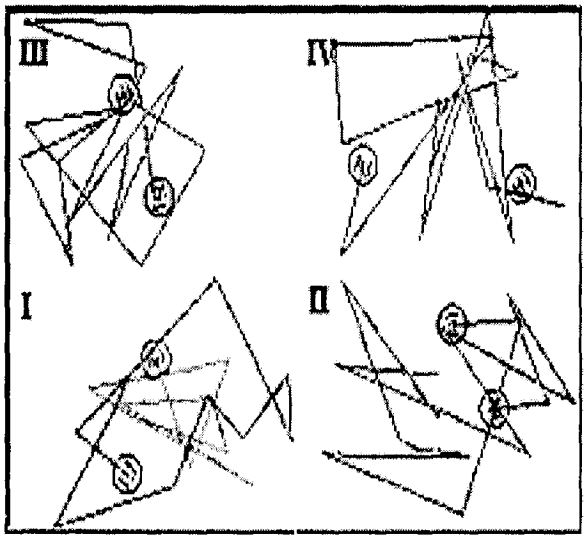

Figure 6.a 8 robots simultaneously work on four areas with 2 robots at each area.

Simulation of task reallocation

The simulationts were carried out on a multiple robot simulator, which models the physical robots in our lab, various aspects of which are described in Netter, 1998, and Wurst & McCartney, 1996. The overall task is sampling a square area (see McCartney & Sun, 2000a, 2000b) using eight robots; this is done by partitioning the area into four regions (I,II,III, and IV, Figure 5), and

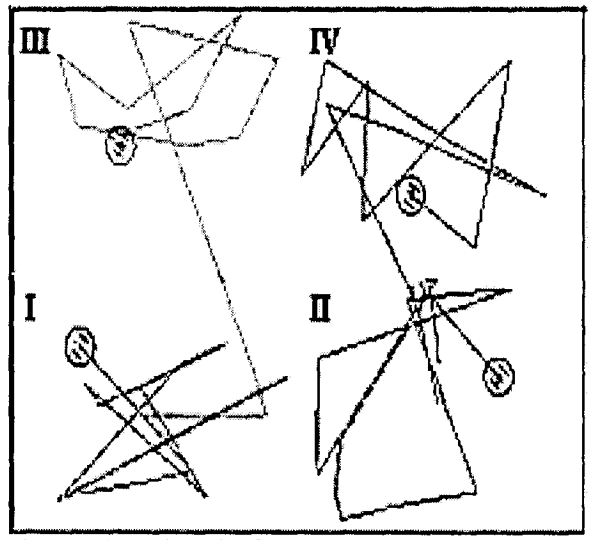

Figure 6.b When the four robots working at area III & IV fail, one robot working at area I will go to area III, one robot at area II will go to area IV.

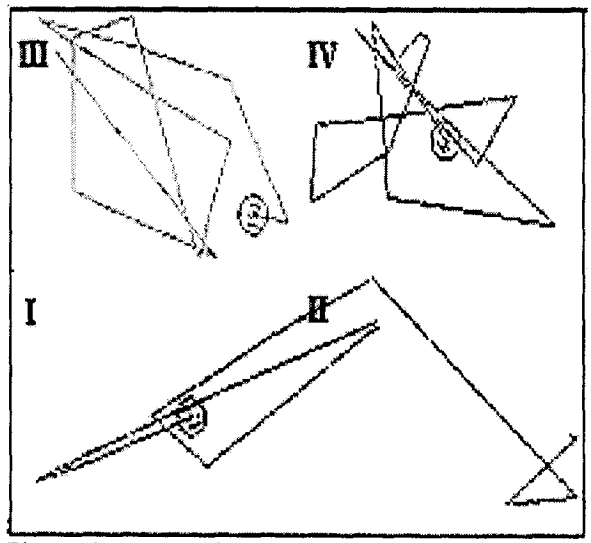

Figure 6.c When the robot working at area I fails, robot working at area II will take charge of the work of sampling both area I and area II.

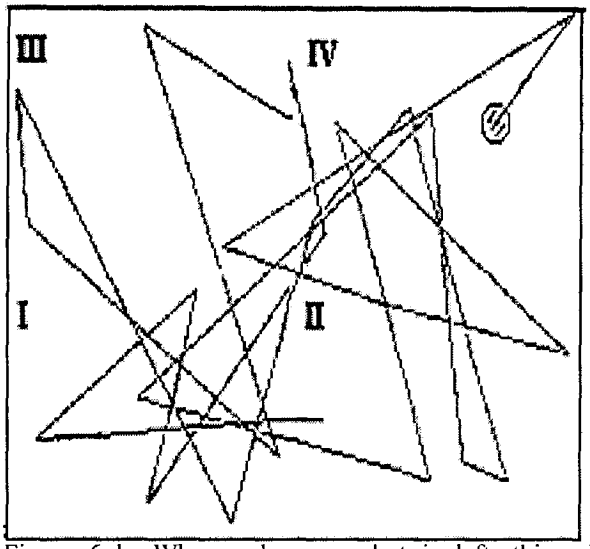

Figure 6.d When only one robot is left, this robot's working area is the whole area, which allows the completion of the task.

assigning two robots to each region. Within a region, the robots sample the area at random and determine area characteristics; if we set no stopping conditions this is equivalent to eight robots patrolling a region.

Figure 6.a. through 6.d. illustrate some observed behaviors when robots fail. The lines in the drawings indicate the individual robot paths.

Figure 6.a. indicates the initial configuration: each quadrant has two robots wandering. Given a single robot failure in any area, nothing reallocates (given a $\rho_1 < .5$). However, if two robots fail in the same area, then a robot will be reallocated from a sibling (if possible). Figure 6.b. indicates what happened after all of the robots in sectors III and IV failed: robots were reallocated from sectors I and II.

Figure 6.c. shows the results of a robot failing when there is only one robot per task: the unassigned sector is combined with its sibling into a larger sector. The ultimate failure result is shown in Figure 6.d.; when only one robot remains it assigns itself to the root, merging the two remaining subtasks into one.

Conclusions

We have proposed a new approach for designing fault-tolerant cooperative multiple robot team, by using a binary tree as a tool to store global information and enable balance of the workload. This paper demonstrates that workload balance can be done efficiently and independently. This approach makes it feasible in a real world multiple robot system with limited communication bandwidth. Experimental results show that robots in the team can respond effectively to robot failures, allowing successful performance even if only one robot remains.

Much work remains, however. For example, a more general model of probabilistic movement decisions needs to be developed and analyzed to characterize how well tasks are balanced, and how long it takes to regain balance given randomized behavior. We should consider the costs of reallocating robots, which has multiple components. One possibility for this would be to consider other tree structures that better capture the spatial characteristics of the environment. Furthermore, we need to develop metrics that enable quantitative measurement of the robustness and efficiency of this approach.

References

Dias, M. B.; and Stenz, A., 2000. A Free Market Architecture for Distributed Control of a Multirobot System. *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pp. 115-122.

Fontán , M. S.; and Mataric, M. J, 1998. Territorial Multi-Robot Task Division. *IEEE Transactions on Robotics and Automation,* 14(5).

Kanellakis, P. C. and Shvartsman, A. A., 1997. Fault-Tolerant Parallel Computation, *Kluwer Academic Publishers*.

Mataric, M. J., 1995. Issues and Approaches in the Design of Collective Autonomous Agents, Robotics and Autonomous Systems, 16 (2-4), Dec. 1995.

McCartney, R.; and Sun, Haihang, 2000a. Random Sampling with Mobile Robots. In *Proceedings of 31st International Symposium on Robotics (ISR 2000),* Montreal, Canada, pp. 89-95.

McCartney, R; and Sun, Haihang, 2000b. Sampling and Estimation by Multiple robots, In Proceedings of 4th International Conference on Multiagent Systems, Boston, MA, pp. 415-416.

Netter, C., 1998. A Hardware/Software Architecture for A modular multiprocessor robot control system, M.S. Thesis, Dept. of CSE, University of Connecticut.

Parker, L. E., 1994. Heterogeneous multi-robot cooperation. Ph.D. Thesis, Dept. of EECS, Mass. Inst. Of Technology.

Parker, L. E., 1998. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation, *IEEE Transactions on Robotics and Automation, 14 (2).*

Wurst, K.; and McCartney, R., 1996. Physical Implementation of Performing Agents. In *Proceedings of the Ninth Annual Florida Artificial Intelligent Symposium,* pp. 132-136.