

Distributed Intelligence in Autonomous Robotics

Assignment #6

Out: Tuesday, April 8, 2003

DUE: Thursday, April 24, 2003

Introduction: Negotiation-Based Task Allocation

This assignment explores the use of negotiation for task allocation and its impact on multi-robot team performance. It involves developing a protocol for negotiation and implementing it using a team of 6 robots using the Nomad 200 simulation environment. You will explore the benefits of negotiated task allocation over random task allocation through experimentation and data collection.

Tasks and Bids

In this application, there is only one type of task and all robots have equivalent capabilities. Here, a “task” involves “visiting” a specified goal position by a robot. There is one task manager that randomly generates the (x,y) goal locations (i.e. “tasks”). This task manager will “announce” the next task to all robots on the team using broadcast communication. Each robot will make a bid for a task based upon the distance from its current position to the goal position. This bid is returned only to the task manager. Only idle robots will make bids for new tasks. If a robot is idle, it will make a bid for an announced task. If a robot is assigned a task in this application, it moves from its current location to the goal location specified by the task. More details are provided below.

Task Manager/Data Collector (TM/DC)

In this assignment, there is one task manager/data collector (TM/DC) process that is a variation of the data collection program you developed in Assignment #3. As in Assignment #3, TM/DC uses the communications system by calling `init_communications`, but it does not connect to the Nserver. The TM/DC does the following:

- First, it generates 6 non-overlapping starting positions, **randomly generated within a circle of radius 1500**, centered at (0,0). The TM/DC then assigns one of these positions to each robot using the communications server.
- The TM/DC queries the user (through a simple user interface) on how many tasks should be generated for this run.
- The TM/DC queries the user (through the simple user interface) whether the run should involve (1) tasks assigned through negotiation or (2) tasks assigned randomly.
- The TM/DC then begins generating tasks and using the negotiation process (described below) to assign the tasks to robots. This process should handle one task at a time, announcing the task, accepting bids, and awarding the task before going on to the next task. (The TM/DC does not wait on assigned tasks to complete before assigning the next task.) It also outputs negotiation process information to the user (described further below).
- The TM/DC collects data on the time to complete all tasks for this run. You will use this data to generate a graph of your results (described further below).
- When the final task is completed, the TM/DC sends a “stop” message to all robots, which then exit their programs.

Based upon the number of tasks specified through the user interface, the TM/DC generates that number of “tasks”, which are (x,y) goal locations. These goal locations should be generated randomly to lie with the area between a circle of radius 3000 and a circle of radius 5000, centered at (0,0). That is, the (x,y) positions should lie on a circle of radius r such that:

$$3000 \leq r \leq 5000$$

Task assignments are made in one of two ways, depending upon the user specification:

- (1) To the robot that provides the “best” bid (i.e., the robot which returns the smallest distance).
- (2) To any idle robot.

In both of these cases, the negotiation messages are sent and processed between the TM/DC and the robots. However, in the case of random assignment, the TM/DC will ignore the bid values from the robots. It is important that the negotiation messages are processed even in the latter case to ensure that data collection measures task execution efficiency, not the implementation efficiency of the communication server.

Multiple runs for each of these cases will be made to compare the degree of benefit that negotiation provides for task execution over random task assignment. The data collection requirements are described further below.

Robots

The robot control program should do the following:

- First, robots receive their assigned position from the TM/DC and use a “place_robot” command to place themselves appropriately.
- Then, they do the following until the mission is complete (indicated by a “Stop” command from the TM/DC):
 - Receive task announcements from the TM/DC
 - Bid on task announcements (only when idle). The bid includes the distance of the robot from the specified goal (i.e. “task”).
 - (Sometimes) receive a task award, in which case they execute the task. “Executing the task” means that the robot moves from its current position to the awarded goal location. When it reaches the goal location, it stops and then sends a final report to the TM/DC that indicates that the task is done. The robot is then available to bid on other task announcements.

When the robot receives a “Stop” message from the TM/DC, it should exit the program.

Communicated messages

In this assignment, you will use communicated messages using the communications server to do all of the following: to assign the initial robot positions, to process the task announcement and bidding process, and to stop the robots. Specifically, you will have 6 types of messages – one that gives a robot its starting position, one that tells the robot to stop (and exit the program), and 4 that are related to the negotiation protocol (described in more detail below).

Experimental setup

In this assignment, you will use the same experimental setup that you used in Assignment #3 – a square of size 10,500 x 10,500 centered at (0,0). You can re-use your map from Assignment #3. The robots will begin at randomly placed, non-overlapping positions in a circle of radius 1500, centered at (0,0).

Develop negotiation protocol

Using a simplified version of the Contract Net Protocol as inspiration, develop a simple negotiation protocol that includes the following message types:

- *Task announcement:* the task manager announces a task. This message consists of: (1) task manager ID, (2) task announcement message type, (3) contract ID, (4) task abstraction that includes the (x,y) location of the goal position.
- *Bid:* a robot's response to a task announcement. This message consists of: (1) robot ID, (2) bid announcement message type, (3) contract ID, (4) bid value, which consists of the distance of this robot from the announced goal position.
- *Announced award:* the task manager awards a task to a robot. This message consists of: (1) task manager ID, (2) announced award message type, (3) contract ID, (4) task specification that includes the (x,y) location of the goal position. (Announced awards only go to the robot receiving the task award.)
- *Final report:* the robot reports to the task manager that it has completed its task (i.e., reached its goal). This message consists of: (1) robot ID, (2) final report message type, (3) contract ID, (4) result description that indicates task is "done".

Negotiation Process Output

The TM/DC should show the status of the negotiation process while it is running by generating the following output:

- [timestamp] Task ID and (x,y) position of task currently being announced
- [timestamp] List of robots bidding on the task and their bid values (i.e., their communicated distances)
- [timestamp] Task ID and Robot ID who has been awarded bid
- [timestamp] (When received) Task ID and Robot ID of task that has become complete.
- [timestamp] Application completion.

These status messages should be generated as the application is being run, and should be formatted as nicely as possible to facilitate user interpretation. Each event is preceded with a timestamp to show the time at which that event occurred. When all tasks are complete, the TM/DC should state this, send the "stop" message to all robots, and then exit.

Data collection

For numbers of tasks equal to 3, 6, and 12, collect time of application completion data for 2 runs of each of the following:

- Task assignments made based upon "best" robot bid
- Task assignments made based upon "random" assignment to any idle robot.

(There will be a total of 12 runs.) "Time of application completion" is the time at which all tasks of the application are completed. The data should be collected electronically by the TM/DC module.

TURN IN THE FOLLOWING:

1. A typed writeup that specifies your negotiation protocol, clearly giving the syntax for each type of message.
2. An example output of the negotiation process for one run of your application, using negotiated task assignments and 12 tasks.
3. An electronically-generated graph that shows your results as follows:
 - x-axis is the number of tasks
 - y-axis is the time
 - Two sets of data are plotted together on this graph:
 - The average time (over 2 runs) of application completion for a given number of tasks when awarding tasks based upon robot distance bids.
 - The average time (over 2 runs) of application completion for a given number of tasks when awarding tasks based upon random assignments to any idle robot.

Your graph should have a title, the axes should be clearly labeled, and the legend should clearly distinguish the two sets of data. Nothing on the graph should be hand-written.

4. A writeup (typed) describing your findings as depicted in your graph. This writeup should explain the impact of task allocation based upon selecting the “best” robot for a task, as compared to the random selection. Your writeup should also discuss any differences you found for varying numbers of tasks and explain why the results show what they do.
5. A writeup (typed) of any issues you had to deal with in implementing this assignment.
6. A hard copy of your programs, fully documented as per earlier assignment instructions. This hardcopy must be printed out using a 2-column “enscript” command, or something similar. (“enscript -2r myfiles”, where “myfiles are your file names).
7. **Email a tar file** of your code, named *yourlastname-6.tar* to parker@cs.utk.edu. **(Zip files no longer accepted.)** This tar file should contain the following:
 - All files needed to compile your code from an empty directory.
 - A Make file that compiles all your code with a “make all” command.
 - A script that enables the user to execute your code from one xterm window.

Be sure to confirm the correctness of your tar package by compiling and executing your code from the tar file from scratch in an empty test directory. As before, points will be deducted if your compilation or execution instructions are missing, incomplete, unclear, or incorrect, or if your code has compiler errors, or if files are missing.