Task Allocation for Heterogeneous Robots

Brian P. Gerkey

April 21, 1998

Abstract

This paper describes the design and implementation of a multi-robot task allocation system. The system is built upon the Mover system (see [2]) and allows the user a high-level interface for posing tasks to a group of autonomous heterogenous robots. Rather than assigning a task to an individual or group of individuals, the user simply poses the task to the system as a whole (with no knowledge of individual machines). Along with a description of the task itself, the user must supply a list of “capabilities” required for each component of the task. Given the list of capabilities, a publish/subscribe messaging system is used to discover qualified and available machines. A negotiation step ensues which results in the actual doling out of the components of the task. Tasks can range from simple one-robot jobs to multi-robot, multi-step, fully-synchronized endeavors.

1 Introduction

A long-time goal of roboticists has been to coerce robots into cooperating. Multi-robot cooperation offers obvious advantages; robots who effectively cooperate with each other on tasks will assuredly be more capable than their non-cooperative brethren who must toil individually at their chores.

In order to achieve cooperation, a multi-robot control system must be developed. In order that it be useful to robot programmers, that system should accomplish as much as possible for the user. Mover, described in [2], is a an example of such a system. The Mover system provides the programmer with a flexible interface for posing complex, multi-step, multi-robot tasks; the system takes care of details such as the shipping of task components to their assigned robots, the synchronization of successive steps within multi-step tasks, and distributed error recovery. However, the user must specify by name which robots will accomplish which task components. It is the automation of this task allocation step with which this project is concerned.

Consider the example task of moving a large[1] box through a specified path across a room. Rather than instructing Robot A to push the box and Robot B to steer, it would be nice to simply indicate how many robots are needed (two) and what capabilities those robots should have (maybe both the pusher and steerer need mobile wheelbases and tactile bumpers while the steerer additionally requires sonar sensors). If we pose this task (in enough detail) to a group of robots the system should be capable of discovering which robots are available to accomplish the task, negotiating for use of resources, and actually doling

[1] I mean ‘large’ in the way that it is used in [1]: that the mass and size of the object are of roughly the same order of magnitude as those of the robot involved.

out the components of the task. These 'robots' could range from sensor-packed mobile manipulators to simple desktop workstations; each machine has different abilities which should be fully exploited in order to meet given goals.

2 Publish/Subscribe Messaging

In order to effectively work together as a team, the robots must somehow export relevant state information so that others may make intelligent decisions concerning the allocation of a task. The metaphor used to describe this communication is inspired by object-oriented (OO) programming: each robot is modeled as a network 'object' on which various methods may be invoked. These methods, or *services*, allow a uniform interface for the remote querying of information such as current position and orientation, processor load, and peripheral device status. To support the autonomy of the robots, these queries must be made anonymously; a user program executing on one robot should not be aware of or be affected by queries made against that robot by others. In addition, a querying robot should be able to make broadcast requests and then simply wait for one or more robots to respond.

In order to enable this anonymous broadcast communication, the actual implementation of the messaging system is based on two concepts: *subject-based addressing* and *publish/subscribe messaging* (see [3] for details). Subject-based addressing is a scheme in which messages are addressed to a *subject* rather than a specific node or process. Publish/subscribe messaging is a messaging model which makes extensive use of subject-based addressing. In a publish/subscribe system, the data producers make their data available by simply "publishing" it in messages to certain subjects. The data consumers can then access that data by "subscribing" to the appropriate subjects; they will receive the data messages as they are published. In short, information is put on a "bus" and anyone who wants it can have it. The result is a loosely-coupled distributed system in which the data producers have no knowledge of the data consumers and vice versa.

3 Subjects

In this system, the idea of a subject has been slightly skewed in that it can be an unordered set of subjects rather than just one. A potential receiver will evaluate any message addressed to a set of subjects which form a proper subset of those subjects to which he is listening. Each robot listens to a set of subjects which represent his "capabilities". For example, our mobile robot *elvis* normally subscribes to the following subjects: ir, sonar, speech, buttons, lights, mobile, elvis, and all. These subscriptions can and do change over time. When the system load drops below some threshold, he will subscribe to an additional subject "idle". When his battery-charging cable is disconnected, elvis will begin listening to the subject "untethered" because he is now free to move about the world. So, to reach all robots who have sonar sensors and are not busy, one can send a message to the subject (sonar idle); only those machines with the specified capabilities will actually evaluate the body of the message (which, thanks to Scheme's treatment of procedures as first-class objects, can be an arbitrary procedure).

4 Primitives

At the lowest level of the task allocation system are three messaging primitives: *publish*, *request*, and *request-exclusive*. All three procedures address messages by subject and then broadcast them to all machines on the network for possible evaluation[2].

The publish procedure is used to send asynchronous messages. Its return value is simply a list of acknowledgements from all the machines on the network. There is no indication of whether or not the message was actually evaluated on the receiver's end. For general synchronous messaging, the request procedure is used; it has the same syntax as publish, but waits for a response from each machine on the network. The return value is a list of results from those machines, if any, which evaluated the message. There are some devices, such as a wheelbase, of which a user program will probably want exclusive use (it would in general be bad for two different tasks to both move the robot around at the same time). For situations such as these, the third messaging primitive, request-exclusive, is required. When a receiving machine evaluates a request-exclusive message, it first unsubscribes from those capabilities included in the address of the message. The message body is evaluated; when the evaluation is complete, the subscriptions are reinstated. In this way, resources can be exclusively reserved so that only one task has access to them.

5 Negotiation & Allocation

While of general interest, the messaging primitives are not usually called directly by user programs, but rather by the high-level procedure *pose-task*. This procedure is the standard interface to the system and allows the user to pose a task in what is hopefully a natural manner. The arguments to pose-task are the following: number of robots required (i.e. number of task components), list of capabilities required for each task component, list of metrics[3], and list of procedures (the actual executable task components). To allow for exclusive reservation of resources, the syntax for specifying capability lists allows the user to indicate which resources are needed exclusively and which are not.

Given this information about the task, the pose-task procedure initiates a simple negotiation protocol in order to decide which machine will play which role. For each task component, a list of eligible machines is determined through various requests; if the component has exclusive resource requirements, those resources are reserved at this time. These candidate lists are sorted by the appropriate metrics and any conflicts are resolved (the system must make sure that the same robot is not assigned to more than one task component). At this point, pose-task has (if possible) chosen a specific machine to execute each task component and can determine whether or not the task can actually be accomplished. If the task cannot be accomplished, any reserved resources are released and notification of the failure is returned to the user. If the task can be accomplished, the first step for pose-task is to release unneeded resources; that is, each machine which has reserved resources for a task component to which it is not assigned is told to release those resources. The chosen machines are then given their respective task components (via the team synchroniztion interface described in [2]) for evaluation. Their

[2]It should be noted that there are a variety of commercial implementations of publish/subscribe messaging (most notably *Rendezvous*, described in [3]) that accomplish this broadcast step extremely efficiently through the use of, for example, IP multicasting.

[3]When more than one robot is qualified for a task component, the user-supplied metric procedure for that task component is used to choose the best candidate. Example metrics include location-metric (choose the robot closest to some target position) and load-metric (choose the robot with the lowest system load).

collective response is returned to the user.

6 Conclusions

While somewhat unpolished, the system described here is a novel approach to the problem of task allocation in that it applies in an effective manner ideas borrowed from the industrial distributed control arena to the study of heterogeneous robot control. The result is an automated task distrution method that offers some clear advantages over the standard manual distrution method. Perhaps most important is the anonymity that is provided for the machines involved. The user has no explicit knowledge of the existence of any of the robots and the robots themselves have no explicit knowledge of each other. Further, rather than relying on some centralized capability repository, each robot privately keeps track of its capabilities, changing them as necessary. From the user's point of view, a task is posed to an unknown set of hopefully qualified robots who will volunteer their available resources (e.g. sonar, wheelbase, CPU) in order to accomplish the task.

There are many avenues for further work on this project. One useful extension would be a new primitive to allow for "non-exclusive reservations", which would reserve resources in such a way that they are available for general use but may not be exclusively reserved by any task. This primitive would be used by tasks which absolutely require the use of certain resources but do not mind if other tasks simultaneously use them. Also, for this system to be truly useful to robot programmers, the entire task-posing interface should be merged with the existing team synchroniztion interface (see [2]) to produce a single, fully-integrated system which can automate task allocation and ensure synchroniztion for arbitrarily complex, multi-step, multi-robot tasks.

Acknowledgments

I would like to thank my project advisor, Dr. James S. Jennings, for all the help and advice that he given me in the past year. I would also like to thank Nicole Terry for bringing a coffee maker into the lab.

References

[1] Russell Gregory Brown. *Localization, Mapmaking, and Distributed Manipulation with Flexible, Robust Mobile Robots.* Ph.D. Dissertation, Cornell University, May 1995.

[2] James S. Jennings and Chris Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In 4th *Intl. Symp. on Distributed Autonomous Robotic Systems*, 1998. Karlsruhe, Germany.

[3] TIBCO Software, Inc. *TIB®/Rendezvous™ Concepts*, August 1997.

Biography

After four years of the virtual war zone that is uptown New Orleans, Brian Gerkey will finally graduate in May of 1998 with a B.S.E. in Computer Engineering (secondary major in Mathematics and minor in Robotics & Automation). In the fall, he is off to glorious Los Angeles to enter the Computer Science Ph.D. program at the University of Southern California. Why a Ph.D.? Because he is allergic to reality.