AIAA-94-1262-CP

THE PROBLEM WITH MULTIPLE ROBOTS

Marcus J. Huber and Patrick G. Kenny
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan 48109-2110
marcush@engin.umich.edu, pkenny@eecs.umich.edu

Abstract

Research in multiple, robotic agents is gaining the interest of an ever increasing number of researchers. Many of these researchers have previously worked in simulation or with single robots, or both. Making the transition from a simulator to the real world can be very trying and frustrating to someone with no experience with such a project. The same goes for making the transition from a single robot to multiple robots. There are a number of issues that arise, mostly of the practical and pragmatic variety, that escape consideration by researchers making these transitions for the first time. We hope to highlight the most important of these issues – discovered primarily through experience with working on multi-robot projects, two of which are discussed in the paper – so that other researchers can give them full consideration when working on their own projects. In addition, we give some suggestions as to how to eliminate or minimize the negative impact these issues might have upon the development of a multiple robot project.

Introduction

A time will come when it will be common to see autonomous robots working together in teams or interacting as individuals. Each of these robots will be performing its specific role in achieving whatever it has been given as tasks. Individual robots, regardless of whether it is working in a team or not, will dynamically interact with each other, the environment, and with humans. They will communicate necessary information in noisy environments, fill in for fallen comrades, and adapt to the temporary loss of sensor subsystems. This scenario is still a long way in the future. What will it take to make this a reality? While research in Robotics and in Distributed AI is always pushing toward this future, research is still in its infancy compared to what is necessary before robots can function as described above.

Quite a bit of research has been done regarding issues related to multiple agents,[6,7,9] and some has been done specifically for multiple robots.[1,11] However, none of this work has really looked at what a researcher faces when trying to implement his or her ideas on real robots for the first time. In this paper we present a number of issues that arise when making the transfer from simulated, theoretical, or single-robot research to working with *more than one* real, autonomous, robot situated in a real world. We discuss each of these issues in some detail and give suggestions, based on experience, for dealing with them. In the first two sections we discuss the issues related to working with more than a single robot and those inherent to working with robots situated in the real world, respectively. We then describe concrete examples of the problems faced when working on multibot projects. Throughout the paper we give suggestions on what can be done to eliminate or reduce these kinds of problems.

Multibot Issues

A number of issues arise when working with multiple robots. We divide these roughly into the issues that arise when looking at the collection of robots as a whole, and those issues that arise when looking at the individual robots that make up the collection. Many of the "collective" issues (such as those that deal with communication, organization, cooperation strategies, etc.) have been addressed in Distributed AI (DAI) research and tend to be fairly abstract in their nature. We talk briefly about these issues but we do give pointers to where more in-depth discussions can be found. Issues that arise from the collection of "individuals", primarily due to the heterogeneity between the agents, seems to be a topic of research of interest to a great number of fields of study (robotics, DAI, artificial life, etc.) but to no one field in particular. In this section we discuss the issues that we see are the most significant to researchers working with collections of these physically embodied agents (robots).

This research was sponsored by DARPA under contract DAAE-07-92-C-R012.

Heterogeneity

Heterogeneous het.er.o.ge.ne.ous , *adj.* Consisting of or involving parts that are unlike or without interrelation; having dissimilar elements; not homogeneous. [< GETERO- + Gk *genos*, kind, sex.] -het'er*o*ge*ne'i*ty *n.*

Heterogeneity among a collection, group, or team of robots is a **BIG** issue. In fact, it may be the single most important issue for researchers making the transition from simulation or theoretical work to consider. Research performed in simulation seldom lacks the completeness required to fully model the differences between robot platforms that will serve as the real-world implementation. Very small differences between simulated robots, which appear insignificant to the uninitiated, can become overwhelming when real robots are pressed into service. We identify a number of factors to especially watch out for (with respect to differences in the robotic platforms) in order to make the transition to real robots easier, and to reduce the potential impact upon various aspects of the multiple robot system if the differences are not eliminated or reduced.

Robots come in an incredible variety of sizes, shapes, and capabilities. Robots can be arms, mobile bases, gantries, snakes, or any of a number of other alternatives. This richness of design makes for a wide range of applicability of robots to different domains, environments, and applications. It is also a major source of grief for anyone wishing to do research with more than one of these robots. There are a great number of places where heterogeneity can cause problems. We divide these into innate and non-innate characteristics, discussed below.

Innate

We consider innate characteristics of a robot to be those features that a robot is "born" possessing, those which are inherent to a robot's basic design and is generally determined by the manufacturer. These include: physical characteristics such as weight, size, and shape; precision and/or accuracy of such things as odometry, positioning (e.g. robot body, camera, etc.); modality and number of sensors; characteristics of the low-level control such as dynamics, functionality, interfacing; design limitations and characteristics such as holonomic characterization, the number of degrees of freedom, bounds on speed, acceleration, reach, etc., and carrying capacity; and the number and type of actuators and/or manipulators.

Many of these features are either impossible, or very difficult, to change, remove, or replace, and are a major boon and bane of robotics researchers. A robot that comes with a powerful, flexible, and complete set of innate "features" can be greatly advantageous. And conversely, a poorly designed robot, or one that may be designed well but ill suited to the task to which is applied, can be a nightmare.

Non-innate

We call the features and characteristics of a robot which are in the control of the roboticist the non-innate features of the robot, those which are a result of work done on the robot to add to or change the functionality of the robot after it arrives from the manufacturer. This includes such things as: the number, modality, precision, etc. of sensors; the number and type of actuators and manipulators; the number, power, memory, connectivity, etc. of processors; the programming language; the high-level control scheme (if any, which would then include the high-level control interface to the low-level controller); the inter-agent communications modality and characteristics; and "sugar" features like speech synthesis and recognition capabilities, graphics displays, etc. It might also include those innate characteristics that can be modified, as there may be some fuzziness to the distinction. There is generally a greater variation in the non-innate features of a robot than in the innate features, due to the wide range of add-on and upgrade possibilities, including "homemade" designs.

The problems

Heterogeneity is an inherently multi-agent issue, as it is defined as the existence of differences between two objects, in this case robots. Heterogeneity arises from both the innate and non-innate features of the robots, and may be looked upon as an advantage in situations where the heterogeneity can be exploited. However, the differences between robots can, and usually does, eventually cause problems. The problems associated with innate and non-innate feature can be very similar, but may possibly have very different solutions (as discussed below). As mentioned earlier, heterogeneity between the robots might very well be *the* most important issue to be faced by researchers working with multiple robots. Our empirical intuition is that the difficulty of implementing and maintaining a collection of multiple robots is a function of both the heterogeneity and the number of robots. We believe that the relationship is something like that of Figure 1. As you can see, we believe that the difficulties associated with increased numbers of agents increases at a higher than linear rate. We believe the same follows for heterogeneity. Of course this is totally unsubstantiated, and is based solely on past experience with multibot implementations.

The problems caused by heterogeneity usually manifest themselves not in the actual experiments conducted by the researcher, but in the *development* stage of the research, where the robots are being readied for the experiments. The development period usually serves the purpose of dealing with the differences, either to avoid them or to take advantage of them, so that when the robots are ready to run experiments the issues have already been considered and addressed. While designing and implementing the robots' sensors, control systems, processing hardware, coordination scheme, etc., a researcher may face problems in any of a number of areas, which we have divided into three broad categories: software, hardware, and functionality. For each category we describe the source of problems that can occur and their effect upon the development of a multibot system.

- Software - Software on the various robots in the

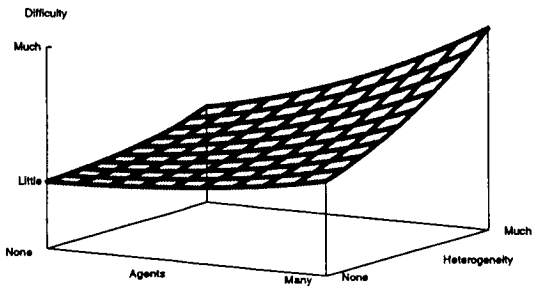

Figure 1: The relationship of difficulty of implementation and maintenance to heterogeneity and the number of robots.

"collection" may be affected by differences between any of a number of robotic characteristics, including the processors, compilers, programming languages, sensors, speed, development environments, and third-party software libraries of the various robots. Any difference in these, or any other of the innate or non-innate features, may create the necessity to modify software to suit a particular robot, which will make the robot all the more heterogeneous. Research agendas themselves may force differences in the software systems utilized by different robots, such as requiring different control architectures or obstacle avoidance algorithms, in order to study the tradeoffs associated with them. Differences in software may range from changed parameters, to modified code, to different software modules, to completely different software systems. Regardless of the source and extent of the heterogeneity, once the differences occur it can be a nightmare to make changes across all of the involved robots to account for each robot's idiosyncrasies.

- Hardware - Robot hardware may differ in sensors, mechanics, physical dimensions, dynamics, CPU's, equipment storage volume, etc. This may be a result of having purchased the robots at different times, implementing different sensor system designs, replacement of broken equipment with non-original parts, etc. Robots that are dissimilar in hardware may or may not create problems; If the hardware on different robots is not equivalent, in that there are enough differences in functionality, modality, speed, etc. to not be transparently switchable, software problems like those discussed above will most likely be created. And other difficulties may also arise, such as having to gain expertise on more and more varied equipment and maintaining the various robots' different hardware.
- Functionality - The capabilities that a robot has depends upon the combination of hardware and software that it has. Given a robot with a particular hardware configuration, the robot can have a range of functionality, depending upon the software written to use the hardware. Likewise, given control software and sensing algorithms, the robot can have varied functionality dependent upon the characteristics of the actual sensors, manipulators, drive motors, and other hardware that the robot is fitted with. Heterogeneity in any aspect of a robot, be it sensing, control, motion, manipulation, or some other aspect, creates a situation where the researcher must make a decision about the functionality that he/she wants the robots to actually possess. Emphasis may be on having all robots possess the same functionality, or it might be desired that the robots possess the maximal functionality possible. Choosing the latter, while understandable, causes more heterogeneity than the former*, and hence possibly exacerbates future problems similar those items discussed above.

Suggestions

The single most important suggestion that we can make to researchers is that they reduce the amount of heterogeneity in the robots that they work with. Heterogeneity between robots is probably the single largest source of problems, effort, and grief encountered while working on research. Eliminating all differences between robots would be ideal, of course, but is not always possible. Robots from different manufacturers will certainly have differences in innate characteristics, as will different models of robots from the same manufacturer, as will even the same model robot from different years. However, these differences can be eliminated *at some level of abstraction*, and it is our suggestion that an effort should be made to accomplish this.

For example, if two robots differ in their low-level motion control functions, a set of higher level functions can be built on top of these commands that removes the robot-dependent aspects. Code written using this new set of functions can then be readily ported between robots.

Of course, dealing with heterogeneity is an interesting research topic, and is therefore necessary in some situations. But it is our belief that it is much easier to introduce differences in robots by disabling functionality or changing parameters (as examples) than it is to eliminate or reduce differences.

Communication

When we talk about communication among robots we mean the intentional act of trying to convey information. And, while communication may not explicitly be used by some researchers,[1,11,13,15] it is very common.[5,8,12,14] Communication between robots is most commonly accomplished using some form of radio frequency (RF) transmission, although it might

*Unless all the robots are exactly the same in all respects, so that their maximal functionality is exactly identical and all the software and hardware required to reach this functionality can also be identical. If the robots are *not* exactly the same, the heterogeneity will show up in the software, at least, in order to achieve the same functionality, if this is even possible.

eventually become possible to explicitly pass meaningful amounts of information by visual means. Tethers or other physical links will most likely not work except for robots firmly fixed in place, such as robotic arms that are not on mobile bases.

Communication can be accomplished in a number of ways, including simple point-to-point and broadcasting (ie. to all robots within range). Complex multibot communication networks can be constructed, however, where robots may not only act as recipients and originators of messages, but also as relays, helping pass messages between two other robots. As more robots interact, communication issues become more and more of an important issue.

Communication issues are unique to multiple robot scenarios (if only because it generally does not make too much sense for a robot to send messages to itself); problems related to communications are therefore also unique to multiple robots.[†] Through our endeavors in multi-robot research, we have identified a number of the problems that seem to plague communication, and we have identified some practical suggestions to at least reduce these problems. Some of the more significant problems are listed below:

- Missing messages - messages never get to their destination.
- Wrong messages - the wrong message is sent, an agent intercepts a message meant for another agent and mistakenly takes it to be for itself, or a message header gets corrupted in transmission and is sent to the wrong agent.
- Garbage contents - a messages information is corrupted to the point of uselessness.
- Communications hardware failure - an agent suffers total loss of ability to communicate.
- Transmission delays - A message's arrival is delayed due to length of travel, number of relaying robots, etc.

All of these problems are caused by RF noise either corrupting or overpowering the intended communications. The magnitude of the problems one will face is directly related to how noisy the RF environment is in which the robots will be used, how robust the low-level communications hardware and software is to corruption (via error checking and correction, handshaking, etc.), and how robust the abstract coordination mechanism is that the robots are using in order to work together (higher level protocols, negotiation schemes, etc., if used at all).

Some more abstract issues related to communication, many of which have been studied in Distributed AI literate, include common knowledge, synchronization, and coherence. Working with real robots means that there is *always* a chance that a message will not be received by the intended robot, or that if it is, that it is corrupted. Halpern and Moses,[10] prove that the involved agents cannot be sure of achieving common knowledge about *anything* that requires communication in such situations. Synchronization of agents is related to this in that, quite often, communication is used by the agents to reach a common point in time at which they know each other's "state" (and can then go on to perform coordinated activities, guaranteed non-interfering actions, etc.)[16,17] Synchronization is usually only possible, however, when common knowledge of every involved agents' state exists so that they can realize when synchronization has been achieved. Coherence deals with coordinating agents having compatible and non-contradictory information. Coherence can be achieved through communication of the data itself, supporting or conflicting evidence, etc. so that each agent eventually believes compatible information.

Of course, if the interacting robots are unconcerned with explicitly coordinating with other agents they will most likely not communicate (as in[1]), and therefore not reason about these communications-related issues.

Suggestions

Solutions to deal with communication problems are pretty commonplace. Technical solutions for these problems include retransmission of messages, semantic message content checks, acknowledged messages, periodic confirmation of activity ("I'm alive!") messages, addressed messages, and robust error detection and correction protocols, among others. Different techniques are necessary for variations of domain, application, robot organization, environment, etc. For instance, in extremely noisy environments it might be necessary to employ error detection and correction mechanisms, retransmission of messages, and handshaking protocols. When the robots are prone to failure, but the environment is noise-free, using simple communication protocols might suffice, but periodic messages from agents indicating that they are functioning might be useful. In general, design in communication overkill. Buy high power, flexible, high quality communication hardware. Determine what will be the worst possible environmental noise that the robots will face, and then employ techniques discussed above for environments twice as noisy.

Planning, Organization, and Task decomposition

Issues related to planning, organization, and task decomposition, among other abstract multi-agent issues, are beyond the scope of this paper, and much research has been conducted on these topics in the Distributed AI field. See "Readings In Distributed Artificial Intelligence"[2] for a collection of papers dealing in detail with these issues. What should be mentioned here, however, is that each of these issues may, in some manner, be affected by the issues discussed in the rest of this paper. Some examples are given below:

- Heterogeneity - Multiagent planning routines will have to be modified to deal with heterogeneous

[†] Communication from a single robot to a base station is pretty common these days, so those researchers that do this will have some insight into multi-robot communication problems.

robots in order to model each of the robots' individual characteristics. As the robot's might differ in any of a number of ways, this might mean adding a large amount of complexity to the planner. Organization(s) of robots will be less flexible than for homogeneous robots, as each agent may not be able to fulfill the responsibilities of every other agent in the organization. Much like the planning system, task decomposition might be much more complex in order to account for the differences in the robots.

- Communication - Communication issues can be expected to work with the equipment and protocols that are going to be used, what type of task decomposition makes the most sense given the environment and domain

Real World and Simulation Issues

Working with robots in the real world may, at first glance, seem to be an easy extension of similar research performed in simulation. And, while simulators are good for testing theories, debugging designs, or just running tests due to environmental constraints, a simulator may give the false impression that the real world is simple and predictable. However, the world is not exact, and is much more complex then any simulator can realistically model. However, they should not be considered satisfactory models in which to completely design and develop algorithms and paradigms destined for real–world robotic applications. Toward this end, some researchers have totally foregone the use of simulators, and have opted to use the world as its own model.[3] We should only rely on simulators to help us prepare a robot for the real world. Though we agree that simulators can be a valuable tool to supplement research with physical robots,[18] at some point in the development cycle it will become necessary to make the transition to the real world. This may not be such an easy task for there are many issues that need to be considered. We divide these issues into three categories. These are: issues that deal with the robot hardware and platform, issues that deal with robot sensors and actuators, and issues that deal with robot software.

Hardware

The first issue, robot hardware and platform, is arguably the area with the most substantial differences between simulation and the real world. Most simulators do not simulate real world events such as when the battery gets low, when computers and sensors on a physical robot fail or start to misbehave (usually without one being aware of the fact), when motors degrade or burn out, or when gears or wheels slip or break. There is a great amount of hardware, all of which, usually, has to work flawlessly. There is hardware to control the motors, hardware to control the sensors, and usually a central computer. There is also minor hardware appliances such as batteries, power converters, actuator circuitry, and monitors that may need to be on board.

An issue that may be overlooked quite often is the actual space requirements of sensors and other hardware on a robot. While a simulator may simulate the functionality of each of the physical components, it probably does not simulate how to place all these components onto a robot base while keeping it stable, usable, and accessible. Robots in a simulator will have an array of simulated sensors, perhaps covering a gamut of modalities such as sonar, vision, range imaging, structured light, and infrared. The robots may have to communicate with each other. They may have manipulators of various sorts. A simulator may permit a all of these capabilities on a robot without consideration to the practicality of doing so.

The type of robot platform or base that the robot is built on will is another important issue to consider. If a base is purchased from a manufacturer, one may not need to worry about the base design, but one does need to worry about the ability to add hardware and software to the robot (i.e. change the non-innate features). Consideration must also be given to such details as the base drive design, which can take a wide variety of forms, from differential drive or synchro–drive, to a tricycle–like or car–like design. The different drive systems may mandate some design decisions on the type and use of sensors, the type of robot control software used, the planning system, etc.

Power constraints are another serious issue often overlooked when dealing with real robots. A great deal of a robot's life is spent charging its batteries, the more so if it is heavily loaded with sensors, actuators, and other electrical equipment. Also, the size and number of batteries restricts the amount of power (current) available, strictly limiting the amount of electrical equipment that may run concurrently on the robot. Even when within this limit, the battery life of a robot is determined by the load on its batteries while running. A robot that must operate over a long period of time must therefore have a light complement of electrically driven sensors, actuators, and other electrical equipment.

Sensors

The second issue to consider are the sensors that a robot might use in the real world. One must consider the kinds of sensors that will be necessary for the robot to perform its task and how many sensors the robot is going to need. A simulator can only approximate the data returned from a sensor based upon its internal sensor model. The more accurate or complex the sensor , the more sophisticated the model will have to be. As an extreme point of view, it may not be possible to realistically simulate real sensors (except perhaps extremely simple sensors like limit switches) in a simulator due to the complexity and uncertainty of the real world. No one can can anticipate *all* of the possible ways that a sensor may fail and/or err. Problems that can (and do) occur include cameras that are not calibrated to specifications, lenses that are dirty or misaligned, sonars with shorted circuitry, incorrect sensor values due to low input voltage, and motor decoders that perform differently than specified or degrade in performance over time.

Another issue to consider is whether the sensors

modeled in simulation return realistic data. For example, a simulated robot may have a sensor that can return the identity of a nearby human. But, one must ask, is there a sensor in the real world that can do that? Perhaps, but probably only at the cost of a great deal of design and implementation effort, and probably one that is quite fallible.

Software

It may be quite a formidable task to track down a software error on a real robot. In a simulator, the only place an error would normally arise would be in the user developed code. In the real world, one also has to consider that an error that might cause the robot to fail or perform differently from the simulator may be the result of a sensor failure, low battery, computer failure, cable problem, or some other seemingly unrelated cause.

Depending upon how carefully the software was designed with the real world in mind, the software run in simulation will most likely run in the real world. When tested in the real world, however, one may find that algorithms need to be changed, libraries modified or rewritten, parameters changed, etc. When making the transition to real robots, one must also consider the size of the software and speed of the robot's processor. Porting code to a minirobot will require great creativity if the software was developed on a Cray supercomputer and requires 128 megabytes of memory. Due to the increased complexity of the real world with respect to that in a simulated model, robot tasks will always be harder than that faced in a simulator. It may require more subtlety or complexity in the planning and control software, more sensors and computational resources, or integration of software not included in the simulator model, to accomplish even a simple task.

Suggestions

To ease the transition of a robot design from a simulator to the real world we offer several suggestions.

- Expect that some additional code will be needed on the physical robot. Some helpful test programs that will test each component to assure that it is working properly, whether it is a sensor or a subroutine, will be useful. After three weeks of running a camera, for example, it may have been bumped out of alignment or had its aperture closed so that it is causing strange and unexpected results in the robots behavior. Hardware problems can have very misleading symptoms and it is good to have test programs that can easily rule out simple causes. And always check connectors, as they are apt to become disconnected. This is a very common problem, as everything on the robot uses cables (cameras, disk drives, monitors, keyboards, etc.) and simply the vibration caused by the robot moving around is enough to loosen cables.
- A useful place to look when a problem arises is the battery. A low battery may cause errors that did not previously exist and be of a type that has never been encountered before. A robot may work correctly for months and then develop unexpected errors because the battery is running low.
- It always helps to work in stages. Test each component and routine as it is moved from the simulator to the real robot(s).
- Document the errors that are encountered. One does not want the next person working on the robot(s) to repeat the same mistakes.
- Last of all, always have a remote emergency robot stop button handy, especially for large robots. Some robots can move very fast and weigh several hundred pounds, and may unexpectedly go out of control.

It may seem that the task of transitioning a robot design from a simulator to the real world will be quite simple and straightforward. There are many issues that need to be considered. There will always be unexpected problems to deal with, but a user that is prepared and considers the issues mentioned above might ease the transition.

Examples

We recently worked on a couple of projects that involved the cooperative interaction of two heterogeneous indoor mobile robots. One project, involving exploration and navigation of an office–like environment, was implemented with some care and consideration to the issues described in this paper, but was so beset by problems that it failed to be completed within a hard time deadline and was never fully implemented. Another project, where the robots had to push boxes across an obstacle strewn floor, was implemented extremely quickly and serves as an excellent example of where failure to implement according to the suggestions above resulted in a very brittle, failure–prone, and frustrating multibot system but which, with some effort to resolve the problems using the suggestions in this paper, can become much more robust and successful.

Dynamic Duo

One of the two robots involved in these projects is CARMEL, a mobile robot based upon a Cybermotion K2A mobile platform. BORIS, the other robot used in the projects, is based upon a TRC Labmate platform. These robots are shown in Figure 2. The major innate difference between the platforms lies primarily in the motion characteristics - CARMEL is essentially holonomic, being able to move in any direction at any time without first having to turn its body, while BORIS must first turn to face the direction of travel before moving. Hardware differences other than those of the platform itself consisted of substantially different sonar rings (CARMEL has a circular ring of sonars, BORIS only has sonars facing in the forward direction), different CPUs (not only do they have IBM 80486 compatibles with differing characteristics, but CARMEL had an IBM XT compatible running the sonar system that BORIS does not have), and differing vision systems (CARMEL has a camera mounted on an independently rotating table, while

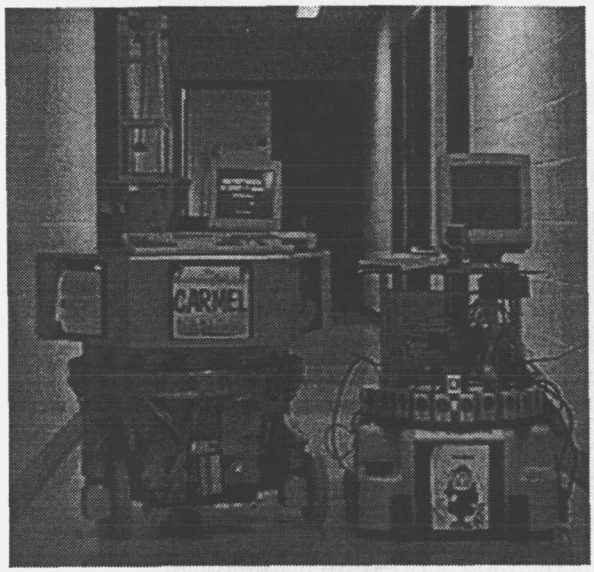

Figure 2: CARMEL (left) and BORIS (right), robots used in several multibot projects.

BORIS's camera is fixed to BORIS's top facing directly ahead of BORIS.) Low level functionality, was similar, but the robots had different implementations of some basic functions, such as obstacle avoidance.

Exploration

One of the events in the National Conference of Artificial Intelligence's '93 robot competition was to autonomously explore an "office" environment looking for a visually tagged object, which was then to be delivered to a predetermined room in the office complex. Walls in the "office" were three foot high panels of sonar–reflecting plastic and arranged to form rooms and halls. Each entry's robot was placed within the office without knowing where within the office it was or in which direction it faced. However, the robot(s) were told in which quadrant of the office environment they started. And each robot was given mostly accurate knowledge of the office layout with respect to wall placement and metric measurements; the actual office layout could differ from the map in that some doors could appear where not indicated on the map, and some doors on the map could disappear.

While most entries were single robot approaches, which we also tried with CARMEL, we attempted a multibot approach (both BORIS and CARMEL) as we saw a distinct advantage in exploring the office in parallel with two cooperating robots. However, while CARMEL was already a fully autonomous mobile robot, with obstacle avoidance and vision systems and a great deal of experience in research and robot competitions,[4] BORIS was initially only a bare TRC Labmate robot base upon which a small amount of obstacle avoidance research had been performed. We realized from the onset that we should attempt to design BORIS to be as *functionally* similar to CARMEL as possible.

A great number of engineering changes had to be made both in hardware and software to accommodate the many differences in the two robots. An example is the feature detection algorithms used to detect the various configurations of "office" halls and openings. The same basic sonar–based obstacle avoidance system was pre–existent in the robots before development started, so that including algorithms to perform feature detection seemed straightforward. The two robots had very different implementations of the sonar system, however, which made porting of the algorithms developed on one robot to the other robot extremely time consuming and is perhaps the single most important factor in the difficulty we had implementing the multibot approach. In addition, the sonar hardware differences (complete vs. partial rings of sonars) require BORIS to rotate in certain situations when CARMEL does not have to, requiring modifications to some low–level functions as well as the planner. Parameters in the sonar system also had to be fine–tuned to each robot to account for differences in the robot's size, sonar filtering system implementation, and other factors, and we were never able to perfectly match the results of the two robots.[‡]

Box pushing

The box pushing task at the AAAI '93 robot competition involved locating boxes (marked with distinctive visual tags) and moving them into a predefined pattern while avoiding obstacles (boxes that were not to be moved). Because a robot pushing a box cannot "see" with its sonars and therefore cannot perform obstacle avoidance, this meant that one robot either scout out a clear path beforehand and then go back and get the box, or that two robots help each other, with one robot acting as the navigator and one robot pushing the boxes. We chose to implement the multibot design, which made the design more challenging, difficult and, as we found out, frustrating.

Our approach to the task was to use CARMEL as the "Boss", or navigator, and BORIS as the "Worker", or box pusher. This task assignment was made because BORIS is built upon a square base, making it more conducive to pushing boxes than CARMEL, which has a cylindrical base. It was important that each robot have some shared knowledge of the world to properly navigate around the arena; for CARMEL to act as the navigator and tell BORIS where to push boxes, it was necessary that each robot initially knew where the other one was located in a global coordinate system. Each robot also had an internal map of the arena indicating the boundaries and the single interior wall. Obstacle and object locations were *not* known beforehand.

The interaction of the robots in this task was defined as follows:

The robots first job was to synchronize themselves using a sequence of handshaking messages. Once synchronized, BORIS would drive to one of a number

[‡]Further ramifications of the difficulty experienced with implementing the multibot approach was that development of the single robot technology was slowed a great deal due to the "thinned" person–power of trying to bring two robots up to competition speed instead of only one.

of predetermined viewing positions in the arena and look for boxes. If it found one (or more) it would approach it, stop just before the box, and communicate its location and orientation to CARMEL. Meanwhile, CARMEL would be waiting at its initial location until it received this message. CARMEL would travel to a location in line with the goal point where the box was to be dropped off and a few meters in front of BORIS's position. CARMEL would then travel in three meter segments to the goal point while avoiding obstacles. At each of these via–points, CARMEL would send its current position to BORIS as an obstacle–safe position. BORIS would attempt to move between these points in such a way as to keep the box securely in front of it (using relatively slow, wide turns) and would "follow the leader" to the point that the box would be placed. Because CARMEL moved in such small increments, the hope was that BORIS's path between via–points would keep it away from obstacles. Once CARMEL reached the box dropoff location, it would then move a safe distance away and wait for BORIS to find another box. BORIS would continue to push the box until the final location was reached. Free of the hindrance of the box, BORIS could then use its own obstacle avoidance system to move to the next viewing position and search for the next box.

As expected all did not work out the way it was planned. There were some problems encountered, some with BORIS, some with CARMEL, and some with the cooperative aspects of the task. BORIS's problems began with its vision system. On the initial attempt, the camera BORIS was using to locate boxes was pointed too low. As it turned out, the boxes we used for testing the robots were upside–down compared to the ones actually used in the arena, causing the tags on the boxes to be higher than the camera could see. BORIS moved from box to box, not recognizing anything, while CARMEL sat motionless waiting for a message from BORIS. After approximately five visual scans, BORIS suffered an unexplained lockup and we had to restart. We fixed the camera angle on BORIS and tried again.

On the second run, BORIS located the box correctly, approached it and transmitted a message to CARMEL communicating he was ready for the 'bosses' orders. CARMEL's map had been initialized improperly, so CARMEL thought that it was on the opposite side of the arena. It moved to the correct location in its own map where it thought BORIS was, not to where BORIS actually was, and plotted out a path to the goal position. The map error resulted in a large discrepancy in positions, however, and the resulting path was useless. Unaware of this, CARMEL then sent the command to BORIS telling it to proceed. CARMEL proceeded to move along its planned path toward the phantom goal destination, sending position messages that were uncorrelated with BORIS's map. BORIS, also unaware of the problem, started to execute the navigation path transmitted from CARMEL, but started to drive in the wrong direction and we had to start over again.

On the third and final run, the situation improved slightly. The robots synchronized, BORIS immediately found a box, CARMEL acknowledged the communications, moved in front of BORIS and the box, and traversed an obstacle–free path for BORIS to travel. CARMEL moved a safe distance away from the box drop off position and waited for BORIS to complete its pushing. However, at some point in the transmission of the initial path via–point from CARMEL to BORIS the message was corrupted and values for parameters were radically in error. This caused BORIS's control program to crash and hang, leaving both robots hanging.

While implementing the design for this project we ran into a number of issues discussed in this paper. Because we were able to actually implement and perform this task, we encountered both issues like those encountered during the exploration task as well as run–time and coordination issues that we did not have an opportunity to discover in the abortive attempt at the exploration task. Communication related issues figured quite prominently among those we ran into. Both during development and actual competition runs we experienced delays, losses, and corruption of messages sent between the robots. Because of the short time period in which we implemented our design (approximately 24 hours), we were unable to build in many of the safeguards recommended above. We successfully synchronized the robots using a set sequence of messages between the robots. We did not, for example, implement any form of semantic contents checking to detect invalid messages. Nor did we implement high–level retransmissions of messages if an expected response from the other robot never arrived. Heterogeneity in hardware did not surface as a significant issue for this task, primarily because we used the robot's heterogeneity to best advantage, we were not trying to achieve equivalent functionality, and the robots were to work have different task responsibilities. Software was more of an issue for the exact same reasons; we had to develop entirely different control software for each robot and could not develop software on one robot and port it to the other.

Future Work

The examples above give vivid accounts of the problems likely to be faced by researchers working with multiple real robots. We continue to be interested in multibot systems and applications, despite the extra effort that such work entails. The next major project is to develop a team of cooperating outdoor robotic vehicles (new military jeeps called HMWMMV's, or "Hummers") that can perform a task such as forward reconnaissance. We have designed and built one prototype vehicle (MAVERIC) based upon an electric utility cart, with which we can do development and experimentation without requiring access to the large and costly Hummers. When building another vehicle – to develop the multiple robot coordination technology necessary for this type of task – we will pay great attention to the issues raised in this paper. First and foremost, we will

try to make the two vehicles as identical as possible. This will most likely entail outfitting the second vehicle in exactly the same manner as MAVERIC in many aspects. It will also probably require upgrading MAVERIC with improvements, based upon our experience with MAVERIC and discovery of shortcoming in its design or implementation, that will be implemented from the onset on the new vehicle.

We would also like to explore using minimal resources on multiple minirobots to perform cooperative tasks. The robots that we are using are based on the MIT miniboard and use an MC6811 microcontroller. While computing power and controlling software are the same for each of the robots, there may be some small variations in the bases and sensors. The general idea is to use the ideas we have discussed here to help design robust multiple robots and apply them in the real world.

Conclusions

A great deal of research has been performed regarding how we can get multiple agents cooperating together in wondrous harmony. Much of this research has not, as yet, involved working with the agents that will actually end up doing the work, robots. A researcher who has not already attempted this technology transfer is in for a lot of headaches unless he or she has some insight to the issues that are associated with such a process. We have introduced and discussed the most significant issues and their causes, and have given many suggestions on how to deal successfully with them. A great deal of work and aggravation can be avoided by paying attention to these issues *before* implementation of a system on real robots. We have described the problems and solution faced when implementing two multibot projects in particular in order to fully illustrate what might occur during implementation, and accentuate the importance of paying heed to the issues raised in this paper.

References

[1] Ronald C. Arkin. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, 3(9):351–364, 1992.

[2] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[3] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1):14–22, March 1986.

[4] Clare Congdon, Marcus Huber, David Kortenkamp, Kurt Konolige, Karen Myers, Alessandro Saffiotti, and Enrique Ruspini. CARMEL vs. flakey: A comparison of two winners. *AI Magazine*, 14(1):49–57, Spring 1993.

[5] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. On the utility of multiagent autonomous robot systems. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 101–108, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[6] Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 875–883, Milan, Italy, August 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 285–293, Morgan Kaufmann, 1988.).

[7] Eithan Ephrati and Jeffrey S. Rosenschein. Constrained intelligent action: Planning under the influence of a master agent. In *Proceedings of the National Conference on Artificial Intelligence*, July 1992.

[8] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 125–129, Washington, D.C., August 1983. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 200–204, Morgan Kaufmann, 1988.).

[9] Michael Georgeff. A theory of action for multiagent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 121–125, Austin, Texas, August 1984. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 205–209, Morgan Kaufmann, 1988.).

[10] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *Third ACM Conference on Principles of Distributed Computing*, 1984.

[11] Marcus J. Huber and Edmund H. Durfee. Plan recognition for real-world autonomous agents: Work in progress. In *Working Notes: Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, AAAI Fall Symposium, pages 68–75, Boston, MA, October 1992. American Association for Artificial Intelligence.

[12] Kouji Ishioka, Kazuo Hiraki, and Yuichiro Anzai. Cooperative map generation by heterogeneous autonomous mobile robots. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 58–67, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[13] Maja Mataric. Synthesizing group behaviors. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 1–10, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[14] Ei-Ichi Osawa. A scheme for agent collaboration in open multiagent environments. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 352–359,

Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[15] Lynne Parker. Learning in cooperative robot teams. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 11–23, Chambery, France, August 1993. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[16] Jeffery S. Rosenschein. Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119, Pittsburgh, Pennsylvania, August 1982. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 187–191, Morgan Kaufmann, 1988.).

[17] Christopher J. Stuart. An implemenation of a multi-agent plan synchronizer. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1031–1033, Los Angeles, California, August 1985. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 216–219, Morgan Kaufmann, 1988.).

[18] Mark C. Torrance. The case for a realistic mobile robot simulator. In *Working Notes: Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, AAAI Fall Symposium Series, pages 181–184, Cambridge, Massachusetts, October 1992. American Association for Artificial Intelligence.