Coordinated Deployment of Multiple, Heterogeneous Robots

Reid Simmons[1], David Apfelbaum[1], Dieter Fox[1], Robert P. Goldman[2], Karen Zita Haigh[2], David J. Musliner[2], Michael Pelican[2], Sebastian Thrun[1]

[1]School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213

[2]Honeywell Technology Center
3660 Technology Drive
Minneapolis MN 55418

Abstract

To be truly useful, mobile robots need to be fairly autonomous and easy to control. This is especially true in situations where multiple robots are used, due to the increase in sensory information and the fact that the robots can interfere with one another. This paper describes a system that integrates autonomous navigation, a task executive, task planning, and an intuitive graphical user interface to control multiple, heterogeneous robots. We have demonstrated a prototype system that plans and coordinates the deployment of teams of robots. Testing has shown the effectiveness and robustness of the system, and of the coordination strategies in particular.

1 Introduction

Mobile robots are being used increasingly in safety-critical applications, such as waste cleanup, space, and the military. Due to the sensitive nature of such domains, human guidance is important to ensure that the robots are attending to the right set of goals. On the other hand, humans may not want, or be able, to exercise detailed control over the robots. This is especially true in situations where multiple robots need to be controlled and in situations, such as military operations, where the human operators themselves are under stress. In such situations, robots must be highly flexible, autonomous assets that can carry out complex tasks with only minimal command effort from humans.

This paper describes an implemented, integrated system that enables a single user to easily control and coordinate multiple robots. The architecture (Figure 1) follows the now-common tiered approach to autonomous systems (*cf.* 3T [4]). In particular, in our architecture the top layer is a task planner, connected to a graphical user interface (GUI), that supports a “playbook” style control strategy, the middle layer is a task-level executive that flexibly coordinates heterogeneous robots, and the bottom layer, replicated on each robot, performs reliable autonomous navigation using probabilistic representations. This paper describes how the system is used to coordinate the deployment of heterogeneous robots into a previously-mapped area. In [5] and [19], we describe how the same basic system is used to coordinate multiple robots in exploring and mapping previously unknown areas.

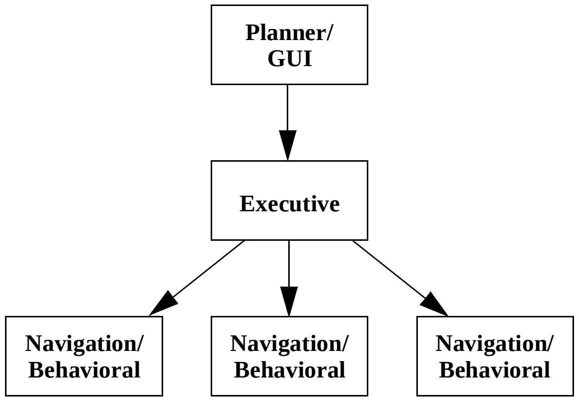

Figure 1: Tiered Architecture for Autonomous Robots

This work is being carried out under the Tactical Mobile Robot (TMR) program, sponsored by DARPA. The goal of the TMR program is to enable war fighters to easily deploy and control teams of small, portable robots in urban settings. Such robots can be used to explore buildings, guard locations, establish communication networks, provide distractions, assist the injured, etc. To avoid interfering with normal military operations, the robots must be fairly autonomous, reliable, and easy to control.

Key aspects of the work presented here include:

- The graphical user interface is designed around the notion of a “playbook”, in which a user commands the robots using a small set of intuitive, parameterized strategies. This project demonstrated the playbook concept and validated the ability to rapidly task teams of robots with minimal user input.
- The MACBeth planner accurately decomposes high-level user intentions into executable robot executive programs. MACBeth is able to communicate tasks to the executive at multiple levels of abstraction.
- The executive uses a high-level task description language to represent the coordination strategies of the robots. In particular, we show how the language enables qualitatively different methods of deployment to be specified with only minor changes in syntax.
- The navigation system uses probabilistic representations to reliably track and guide the robots. The same system works on several different platforms, with different sensor and actuator configurations.

The next section describes in more detail the scenario that motivates our work. Section 3 presents related work in multi-robot deployment and coordination. Sections 4, 5 and 6 present the planner/GUI, executive, and navigation layer, respectively. Section 7 describes some of our experience with the multi-robot deployment system, and Section 8 offers conclusions.

2 Scenario

The scenario here is the coordinated deployment of robots in a previously-mapped environment. For example, several robots might have been tasked with exploring an area to make a map [19] and then return to "base". After that, a larger team of robots might be deployed to strategic positions in the building, for instance to provide line-of-sight communications, or to monitor for intruders.

We want users to have to specify only the deployment locations and a general strategy for the deployment, and to have the software system determine which robots go where (based on their different capabilities) and how to navigate to the locations in a coordinated way. The deployment strategies differ in the order and degree of concurrency used in moving the robots. Each deployment strategy is a type of set "play", parameterized by the number and end locations of each robot.

We have investigated three different deployment strategies. The *group* deployment has all the robots move concurrently to the closest chosen location, then one stays behind and the rest move to the second closest location, etc., until the last robot travels (alone) to the last location. In contrast, the *wave* deployment strategy uses the notion of a "point man" -- one robot goes to the first location and, when it arrives, the second robot travels to the first position while the first ("point man") robot moves on to the second location. The robots continue to deploy in a wave-like manner until the first robot reaches the final location, at which point the last robot has also arrived at the first location. In the *leap-frog* deployment, a "point man" robot travels to the first location, then a second robot travels *past* the first robot to the second location, with each subsequent robot traveling past all the previously deployed robots to get to the next location.

Issues in this scenario include planning out which robots should go where, coordinating their interactions to implement the desired deployment strategies, getting the robots to navigate reliably, and minimizing human involvement in all this. After presenting related work, we describe our approaches to these problems.

3 Related Work

While there has been much prior work in multi-robot cooperation, we focus here on a few relevant pieces of work. Several authors have investigated multi-robot formations, mostly concentrating on the behavioral aspects of maintaining formation, rather than the sequential aspects of deployment that is the focus of this paper. Balch and Arkin [2] developed behaviors to maintain different types of formations (line, wedge, etc.) under different conditions (leader-centered, unit-centered, etc.). They showed how the behaviors were able to reliably maintain formation and smoothly switch between formations. While they did not address planning or explicit robot synchronization, their methods could be combined with our own, at the behavioral level. Other authors [6, 21] have looked at formation maintenance in a control-theoretic framework, especially with regards to the stability of formations in situations with only limited inter-robot communication. Parker [15], in her ALLIANCE architecture, is also able to handle formation maintenance, among other multi-robot tasks. While still not employing explicit coordination, the robots use the concepts of "motivation" and "impatience" to effectively cooperate, and to do so in a fault-tolerant manner.

Jennings et. al. [12] have developed a distributed executive for multi-robot coordination. The executive, based on a distributed dialect of Scheme, is similar to our executive language in the types of explicit synchronization constraints it supports. One difference with our work is that Jennings' executive is not integrated with a planner. Another difference is that his executive is truly distributed amongst the robots. This is something that we are currently implementing for our executive language.

Alami et. al. [1] present a general methodology for distributed planning and plan execution. This contrasts with our approach in which the planning is centralized. They describe methods for incrementally merging plans in a distributed fashion. Like our (and Jennings) work, temporal constraints are used to coordinate activities, and these constraints enable tasks to be invoked automatically on one robot when another robot finishes a task. An interesting aspect of the work is that the plan merging activities may themselves be scheduled and coordinated.

4 The Playbook GUI and MACBeth Planner

In our work, multi-robot deployment is commanded using a prototype *playbook GUI* [13], which provides a high-level command and control interface requiring minimal human attention. The concept is derived from sports playbooks, where each player knows the broad outline of a "play," and each play has been well-practiced by the team. During an actual game, the play is "tailored" by a coach or

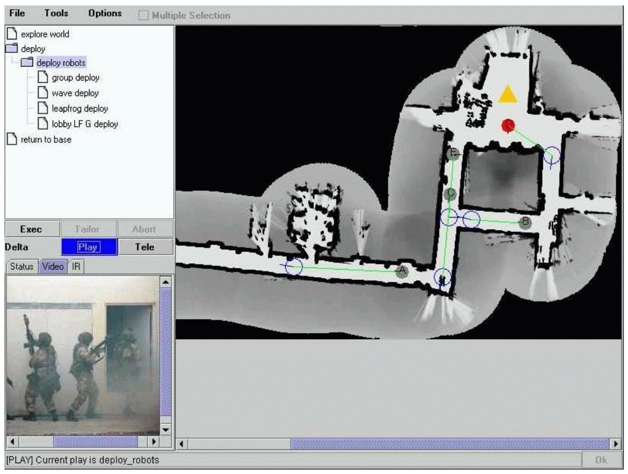

Figure 2: Playbook GUI for TMR Deployment Domain

team captain to meet the conditions on the field. Similarly, the playbook interface provides easy access to a set of pre-defined mission profiles (deployment maneuvers), along with tailoring functions that allow the user (a soldier) to quickly and easily customize the deployment for a particular scenario.

Figure 2 shows a sample of the playbook GUI for the robot deployment task. The GUI is written in Java to operate in a web browser. In the upper left corner, the GUI displays the list of available plays, organized in a browser metaphor. The user can select and execute plays at any level of the browser structure, thus providing more- or less-detailed commands to the robot team. In the lower left corner, the GUI shows status information for a selected robot. The right hand side displays the map, the home base (triangle), the robots' current locations (filled circles), and the robots' goal locations (clear circles). After the user selects a play, he can choose either Execute, and the MACBeth planner will generate a plan and send it to the executive, or Tailor, and MACBeth will generate a (constraint-free) plan, and then allow the user to set constraints and otherwise tailor the play to the current situation.

To support the playbook commanding metaphor and allow the user to command at various levels of abstraction, the GUI interacts with the MACBeth planner to build complete robot team plans. MACBeth is a constraint-based tactical planning engine for multi-agent teams. MACBeth is designed for domains in which a human user must quickly specify a mission to a team of autonomous agents. In these domains, "puzzle-mode" thinking to come up with novel plans is not important; the key task is to rapidly and accurately tailor existing plans to novel situations. To this end, MACBeth combines hierarchical task network (HTN) planning and modern constraint reasoning techniques in a mixed-initiative planning system. The playbook GUI uses MACBeth to generate, check, and modify team plans.

In this paradigm, "calling a play" means that the user declares that a certain task needs to be accomplished. The user can "tailor" the play by setting parameters and adding constraints specified by operating conditions. Tailoring a play essentially allows for adjustable autonomy on the robots: The user can command the robot team at different levels of detail, leaving unspecified details up to the judgment of the autonomous systems, when desired. For the TMR task, the target user is a soldier squad leader or robot operator within a squad. The plays contain internal timing constraints, and certain actions (e.g., take a photo) have related hardware requirements. Most remaining constraints are map- or environment-related, including navigation waypoints and goal locations (specified by the user) and required locomotion capabilities (specified by the route planner in the navigation layer).

In tactical applications, the mission planning system must *ensure that plans are feasible*. If the user attempts to specify an infeasible plan, conflicts and inconsistencies should flagged by the planning algorithm as soon as possible, so that the user can retract choices or reconcile conflicting objectives early in the planning process. For example, a user might request a team of robots to map an area that is too large for them to cover in the required time. The planner should identify the inconsistency quickly, ideally before it expends planning effort on low-level details, and immediately alert the user, so that he may either modify his objectives or assign additional resources.

The combination of HTN planning, constraint programming, and playbook GUI was chosen to meet these needs. Hierarchical task nets provide a representation for plans that is relatively easy for people to understand. They can also support very efficient planning, when the ability to construct novel plans (completeness) is less of interest. The HTN plans provide a skeleton on which to perform constraint reasoning, and form a very natural interface to the executive (see Section 5).

Constraints on MACBeth's plans capture the complex relationships between (otherwise unrelated) tasks, allowing MACBeth to reason about the tradeoffs of different resource allocations. MACBeth's constraint handling also assists human planners in resource management and in appropriately assigning mission roles to team members. Thus, users do not have to reason about which agent does which task, or how to reallocate agents in order to achieve mission goals.

Constraints also provide a clean interface to external knowledge sources. For instance, in this application the navigation layer provides route planning information based on a metric map. The route planner is interfaced to MACBeth via constraints: MACBeth does not understand the geometric character of routes, so the route planner gives it summary information in the form of constraints, such as the time the route is expected to take and the length of the route. MACBeth uses this information to make decisions about which routes to take. For example, the user may task two robots to deploy to two different locations, but not specify which robot goes to which location. In this situation, MACBeth fills in these missing details by querying the route planner for routes for each robot to each destination, and then chooses the lowest-cost robot-to-destination assignment according to the constraints calculated by the route planner.

5 Executive

The executive dispatches tasks and coordinates the multiple robots. It acts as a bridge between the high-level symbolic task planner and the lower-level navigation layer. When the user chooses to execute a play, MACBeth sends the hierarchical task net to the executive using the generic Plan Representation Language. PRL defines each task in terms of parameters, subtasks, and temporal constraints between subtasks.

MACBeth transmits the complete hierarchy, rather than just the leaf tasks, for several reasons. First, if the executive knows the hierarchical relationships between tasks, it can more flexibly report on status and can terminate complete subtrees with a single command. Second, as described below, some tasks must be augmented to cope with interference and other aspects of the real world that the planner does not model. Thus, the executive potentially needs to know about all the tasks in the hierarchy. Third, this allows for a flexible boundary between planner and executive. The planner can decide to plan some tasks at a high level of abstraction and to expand some to a lower level, while relying on the executive to fill in the details.

The primary role of the executive is to execute plans according to the constraints imposed by the planner. The executive keeps track of the synchronization constraints between tasks and dispatches tasks when the constraints are satisfied. The executive is implemented using the Task Description Language [18]. TDL is an extension of C++ that contains explicit syntax to support task decomposition, task synchronization, execution monitoring, and exception handling. The TDL compiler transforms TDL programs into pure C++ code, plus calls to a general-purpose task management library. The transformed programs can then be compiled and linked with a normal C++ compiler.

One feature of TDL is the expressive constructs it supports for specifying temporal constraints between tasks, including both qualitative and quantitative relationships between the start and end points of tasks. For instance, a TDL task can be constrained to precede another (meaning that it, and all of its subtasks, must complete before the other task, or any of its subtasks, can start), or a task can be constrained to start 10 seconds after another, or a task can be constrained to terminate whenever another task completes. While TDL was originally developed to support coordination amongst multiple behaviors on a *single* robot, we have found it to be very useful for synchronizing the tasks of multiple robots, as well. We are currently working to distribute TDL so that each robot can have its own executive that transparently coordinates with the other executives.

Both MACBeth and the executive view multi-robot deployment as having the same basic form: For *N* "stages" (where *N* is the number of robots, which equals the number of locations), move some group of *n* robots to some set of *m* locations. To a first approximation, the various deployment strategies differ with respect to the number of robots moving per stage, and the order in which they move. For example, Figure 3 shows TDL code (simplified for clarity) for the three deployment strategies that we have implemented (see Section 2). "deployList" is an N-element array, where the i^{th} element of the array indicates which robot is to end up in which location. In all the deployment strategies, no matter how the robots move initially, they all must end up in the locations specified by the "deployList".

Group deployment is distinguished by the fact that the number of robots moving decreases over time, starting with *N* and ending with one. In contrast, the number of moving robots *increases* in the *wave* deployment. In the third approach, *leap-frog* deployment, the number of robots moving is constant (one).

The deployment strategies in Figure 3 essentially treat each robot as a point object, ignoring potential interference from other robots. For instance, the *group* deployment strategy has robots moving to the *same* location. In the real world, we need to take account of potential interference caused by robots with volume and mass. While the local obstacle avoidance behaviors of the navigation layer ensure that the robots do not crash into each other, the emergent behavior may not be very efficient.

To remedy this, we *augment* each of the basic deployment strategies with coordination behaviors that act to reduce interference. Interference in the group deployment strategy comes from two sources. First, in each stage the strategy is supposed to deploy a group of robots to a given location. Since real robots cannot occupy the same space, the executive actually tasks them with slightly different destinations. The robot that is supposed to end up at that location is sent directly to the location specified in the "deployList". The other robots are deployed along a line between that location and the next goal location, separated by a fixed distance. The other type of interference is that the robots move concurrently, starting from the same general location. To minimize the likelihood that they have to avoid one another, we stagger the robots. Each robot

```
Goal GroupDeploy (DEPLOY_PTR deployList)
{
    with (serial) {
        for (int i=0; i<length(deployList); i++) {
            spawn GroupDeploySub(i, deployList);
}   }   }
Goal GroupDeploySub (int phase, DEPLOY_PTR deployList)
{
    with (parallel) {
        for (int j=phase; j<length(deployList); j++) {
            spawn Deploy(deployList[j].robot,
                         deployList[phase].location);
}   }   }

Goal WaveDeploy (DEPLOY_PTR deployList)
{
    with (serial) {
        for (int i=0; i<length(deployList); i++) {
            spawn WaveDeploySub(i, deployList);
}   }   }
Goal WaveDeploySub (int phase, DEPLOY_PTR deployList)
{
    with (parallel) {
        for (int j=0, k=length(deployList)-1; j<=phase; j++) {
            spawn Deploy(deployList[k-phase+j].robot,
                         deployList[j].location);
}   }   }

Goal LeapFrogDeploy (DEPLOY_PTR deployList)
{
    with (serial) {
        for (int i=0; i<length(deployList); i++) {
            spawn LeapDeploySub(i, deployList);
}   }   }
Goal LeapDeploySub (int phase, DEPLOY_PTR deployList)
{
    with (serial) {
        for (int j=0; j<=phase; j++) {
            spawn Deploy(deployList[phase].robot,
                         deployList[j].location);
}   }   }
```

Figure 3: Deployment Strategies in TDL (Simplified)

monitors the position of the next robot in the “deployList” and does not start moving until that robot is a given distance ahead of it. This combination of coordination strategies keeps the team of robots well disciplined (see Figure 5).

Interference in the leap-frog deployment strategy occurs when one robot moves past other robots on its way to a goal location. This is exacerbated because we typically operate in narrow corridors. To minimize such interference, the already-deployed robots monitor the position of the moving robot. When it starts approaching, the robots move away (roughly perpendicular to the route of the moving robot), wait until the robot has passed, then move back into position. In this way, the designated deployment locations are left unoccupied for a minimal period of time. For the wave deployment strategy, there is no robot-robot interference, so no augmentation is necessary.

One additional role of the executive is to send MACBeth and the GUI dynamic state information. After each task completes (or if it fails), the executive sends a status message back to the planner and GUI. It also monitors, and periodically transmits, the position and status of each robot. The GUI can synchronously request map information, which is used to display where the robots are, and the planner can request route information between arbitrary locations, which it uses to decide how to deploy the robots. Finally, the GUI can request camera images from particular robots, to give the user a robot-eye-view of the situation.

6 Navigation

Our robots employ a probabilistic system for navigation and localization. At the beginning of the coordinated deployment, we assume that a map of the environment is readily available, such as the one shown in Figure 4. Maps like these are easily built using the software approach described in [20], which extends Elfes’ and Moravec’s occupancy grid mapping algorithm [8, 14] by a real-time method for concurrent localization. The map shown in Figure 4 was acquired by a single robot; the issue of multi-robot coordination during mapping is beyond the scope of this paper and approaches can be found in [5, 19].

Equipped with such an occupancy grid map, the navigation task can be decomposed into three components: *localization*, *motion planning*, and *collision avoidance*. Such a decomposition is a *de-facto* standard in mobile robotics: Localization is a pure observer, estimating robot position based on sensor data, without impacting the way the robot behaves. The control problem is decomposed into two components: The motion planner generates globally near-optimal paths, and the collision avoidance algorithm is responsible for generating motor commands that meet the dynamical constraints imposed by the physical robot.

In brief, our localization module implements the *Monte Carlo localization* (MCL) algorithm [7], a Bayesian solution for the localization problem. This algorithm estimates a robot’s position, denoted x, as a posterior, using the following recursive estimator:

$$\mathrm{Bel}(x) = P(y|x)\sum P(x|u, x')\mathrm{Bel}(dx')$$

Here y is the most recent sensor measurement and u is the controls (or, alternatively, a differential odometry reading). The conditional probabilities are easily modeled using simple mixture models, as described in [10].

The MCL algorithm implements this equation using a particle filter [16], which generates a set of (weighted) random particles that, as a whole, represent the desired

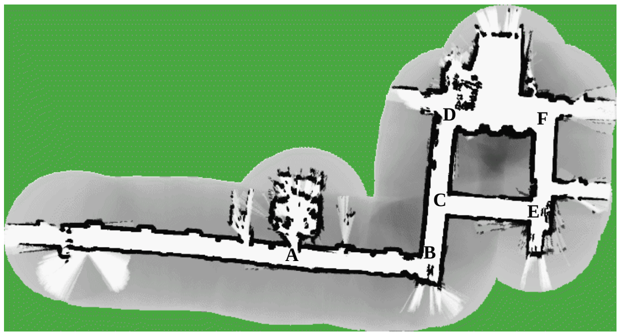

Figure 4: Learned Map of Part of Wean Hall at CMU

posterior Bel(x). The basic algorithm is very straightforward:

- Generate a set of random x' from Bel(x').
- For each x', guess x according to $P(x \mid u, x')$.
- Weight each x numerically by $P(y \mid x)$. These weights are taken into account in the next iteration.

As argued in [7], this algorithm is extremely efficient, robust, and accurate.

The path planner uses value iteration [3], a version of dynamic programming, to determine the shortest path to the goal. Value iteration computes the distance to a goal point from arbitrary starting locations, hence is capable of recovering from unanticipated motions generated by the collision avoidance module. For computational reasons, our approach considers two dimensions only, ignoring important factors such as inertia, torque limits, and obstacles (such as people) not present in the map.

Those other factors *are* taken into account by DWA, the collision avoidance algorithm [9]. DWA computes motor commands from the motion plans and a set of hard and soft constraints. Hard constraints are necessary to ensure the safety of the robot and its environment. They make the robot avoid configurations that inevitably would lead to a collision. For example, hard constraints force the robot to decelerate when approaching an obstacle, so that a safe stop can be guaranteed at all times. Soft constraints express preferences in control space. Currently, DWA combines three soft constraints: the desire to move towards a goal location (as set forth by the motion planner), the desire to move fast, and the desire to stay clear of obstacles. Together, these three soft constraints lead to smooth and effective behavior.

Under the name of BeeSoft, this navigation system has been distributed by RWI Inc., a major robot manufacturer, and is now in widespread use in academic institutions world-wide. It has been ported to a large number of mobile robot platforms, including Pioneer I and II (by ActivMedia), ISR Pioneer AT, B14, B18, B21, Nomad Scout, Superscout, and XR 4000, and also was at the core of two museum tour-guide robots.

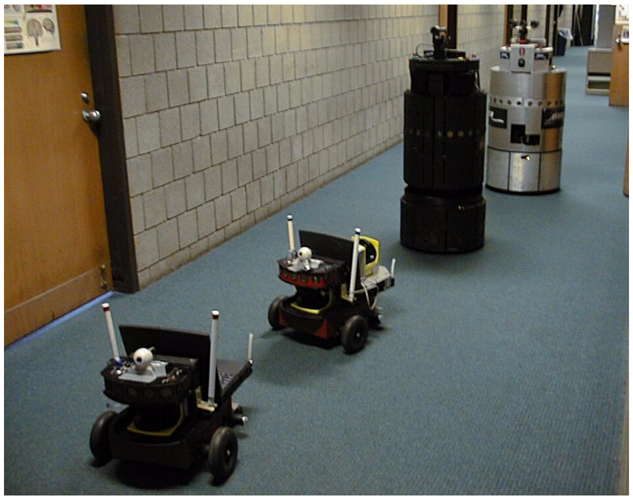

Figure 5: Coordinated Deployment of Robin, Marion, Amelia and Xavier

7 Experience

The multi-robot deployment scenario was tested in several locations, including a remote TMR demonstration. Most of our testing was done in the corridors of our building at Carnegie Mellon (Figures 4 and 5). A typical scenario used four robots: Robin and Marion, two Pioneer AT's modified to hold a SICK laser range finder, Amelia, an RWI B21 with a SICK laser, and Xavier, an RWI B24 that used only sonar sensors. Robin, Marion and Amelia all use the BeeSoft navigation package described in Section 6. Since Xavier uses a somewhat different system [17], we added a layer that translates executive commands into Xavier's formats. However, the executive still has to explicitly deal with the fact that some robots have different map representations, different sensors, and that some do not have cameras.

A typical scenario was to have all four robots execute one group deployment stage to take them from the "base" (our lab, labeled **A** in Figure 4) to the intersection labeled **B**. Then, two robots would leap-frog deploy to locations **C** and **D**, and the other two would concurrently group deploy to **E** and **F**. The whole deployment would take about 3-4 minutes, and the robots would average about 50 meters of travel. After deployment, the robots executed a set play to return to base (in a concurrent, but uncoordinated, fashion).

One problem with this scenario is that the first leap-frogging robot could conceivably interfere with the two robots performing the group deployment. While we could have augmented the plan to explicitly coordinate these robots, it was far easier (and nearly as robust) just to make sure they were ordered appropriately after the first group deployment (in particular, the leap-frogging robot should be ahead of the others).

Our testing indicates that the coordinated strategies were effective and efficient. By the end of our testing, the strategies were also quite robust. This was mainly due to the explicit coordination behaviors (described in Section 5) that were added to alleviate possible interference between robots.

8 Conclusions

To be useful in many applications, mobile robots need to be fairly autonomous and easy to control. This is even more important for teams of robots, especially in situations where the robots can potentially interfere with one another. This paper has described an integrated multi-robot system that (1) uses an intuitive "playbook" interface to simplify user control and understanding, (2) plans complex tasks scenarios, (3) uses explicit synchronization techniques to coordinate and monitor the behaviors of multiple, heterogeneous robots, and (4) uses probabilistic techniques to reliably and efficiently navigate the robots.

The system has been tested in the area of deployment of teams of robots. Qualitatively different deployments were developed based on small syntactic changes to a basic underlying strategy of deploying *N* robots to *N* locations in *N* separate stages. This not only facilitates development, it also made it easier for users to switch strategies dynamically. Subsequent work has extended the system to multi-robot mapping and exploration [5, 19] and a similar system is being developed for multi-robot assembly and construction [11].

Acknowledgments

Thanks to Greg Armstrong for maintaining all the robots and assisting with the experiments. This work is sponsored in part by DARPA via TACOM contract DAAE07-98-C-L032.

References

[1] R. Alami, F. Ingrand, S. Qutub. "A Scheme for Coordinating Multi-robot Planning Activities and Plans Execution." In *Proceedings of European Conference on Artificial Intelligence*, Brighton UK, 1998.

[2] T. Balch and R.C. Arkin. "Behavior-based Formation Control for Multi-robot Teams." *IEEE Transactions on Robotics and Automation*, **14(6)**, 1998.

[3] A.G. Barto, S.J. Bradtke, S. P. Singh. "Learning to Act using Real-Time Dynamic Programming." *Artificial Intelligence*, **72**, pp. 81-138, 1995.

[4] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp , D. Miller and M. Slack. "A Proven Three-tiered Architecture for Programming Autonomous Robots." *Journal of Experimental and Theoretical Artificial Intelligence,* **9(2)**, 1997.

[5] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun,. "Collaborative Multi-Robot Exploration." In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, April 2000.

[6] Q. Chen and J.Y.S. Luh. "Coordination and Control of a Group of Small Mobile Robots." In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2315-2320, San Diego CA, 1994.

[7] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. "Monte Carlo Localization For Mobile Robots." In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[8] A. Elfes. "Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation." PhD Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[9] D. Fox, W. Burgard, S. Thrun. "The Dynamic Window Approach to Collision Avoidance." *IEEE Robotics and Automation*, **4(1)**, 1997.

[10] D. Fox, W. Burgard, and S. Thrun. "Markov Localization for Mobile Robots in Dynamic Environments." *Journal of Artificial Intelligence Research*, **11**, pp. 391-427, 1999.

[11] D. Hershberger, R. Burridge, D. Kortenkamp and R. Simmons, "Distributed Visual Servoing with a Roving Eye." In *Proceedings of Conference on Intelligent Robotics and Systems* (this volume), Takamatsu Japan, 2000.

[12] J.J. Jennings, G. Whelan, W.F. Evans. "Cooperative Search and Rescue with a Team of Mobile Robots." 1996.

[13] C. Miller and R. P. Goldman. "'Tasking' Interfaces: Associates that Know Who's the Boss. In *Proceedings of the Fourth Human Electronic Crew Conference*, Kreuth, Germany, September 1997.

[14] H.P. Moravec. "Sensor Fusion in Certainty Grids for Mobile Robots." *AI Magazine*, pp. 61-74, Summer 1988.

[15] L. Parker. "Heterogeneous Multi-Robot Cooperation." PhD Thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., 1994.

[16] M. Pitt, and N. Shephard. "Filtering via Simulation: Auxiliary Particle Filter." *Journal of the American Statistical Association*, 1999.

[17] R. Simmons, R. Goodwin, K. Zita Haigh, S. Koenig. and J. O'Sullivan. "A Layered Architecture for Office Delivery Robots." In *Proceedings of First International Conference on Autonomous Agents*, Marina del Rey, CA, February 1997.

[18] R. Simmons and D. Apfelbaum. "A Task Description Language for Robot Control." In *Proceedings of Conference on Intelligent Robotics and Systems*, Vancouver Canada, 1998.

[19] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun and H. Younes. "Coordination for Multi-Robot Exploration and Mapping." In *Proceedings of the National Conference on Artificial Intelligence*, Austin TX, August 2000.

[20] S. Thrun, W. Burgard, and D. Fox. "A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping." In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco CA, April 2000.

[21] P.K.C. Wang. "Navigation Strategies for Multiple Autonomous Robots Moving in Formation." *Journal of Robotic Systems*, **8(2)**, pp. 177-195, 1991.