Cooperative Multiagent Robotic Systems

Ronald C. Arkin and Tucker Balch
Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

1 Introduction

Teams of robotic systems at first glance might appear to be more trouble than they are worth. Why not simply build one robot that is capable of doing everything we need? There are several reasons why two robots (or more) can be better than one:

- Distributed Action: Many robots can be in many places at the same time
- Inherent Parallelism: Many robots can do many, perhaps different things at the same time
- Divide and Conquer: Certain problems are well suited for decomposition and allocation among many robots
- Simpler is better: Often each agent in a team of robots can be simpler than a more comprehensive single robot solution

No doubt there are more reasons as well. Unfortunately there are also drawbacks, in particular regarding coordination and elimination of interference. The degree of difficulty imposed depends heavily upon the task and the communication and control strategies chosen.

In this chapter, we study our approach to multiagent robotics in the context of two major real world systems:

- As part of ARPA's UGV Demo II program, we have studied the ways in which reactive control can be introduced to the coordination of teams of HMMWVs (Jeep-like vehicles) working together in dynamic, unstructured, and hazardous environments.
  - The design of formation behaviors is discussed in Section 3.
  - Means for specifying missions for robotic teams using the *MissionLab* system are presented in Section 4.
  - Team teleautonomy, where an operator can influence the behavior of a collection of robots, not just one at a time, is discussed in Section 5.
- A three robot team that won the AAAI Clean-up-the-Office competition in 1994.

2 Related Work and Background

We first briefly review a sampling of relevant research in multiagent robotics and then present some background in schema-based reactive control.

2.1 Multiagent Robotic Systems

Fukuda was among the first to consider teams of robots working together [9.]. His cellular robot system (CEBOT) is a collection of heterogeneous robotic agents which are capable of assembling and disassembling themselves. Imagine, for example, a self-assembling ship in bottle. This ability to allow complex structures to be constructed on-site and the additional capability of reconfiguring the combined units is of potentially great value for a wide range of applications in space-constrained environments.

Mataric, in her dissertation research at MIT, has studied adaptive cooperative behavior in a collection of homogeneous robotic agents, the so-called *Nerd Herd*. Using a behavior-based subsumption-style architecture [14.], she demonstrated, in hardware, group behaviors such as flocking and foraging for a team of 10-20 robots. Of interest is the phenomena of interference which occurs due to the robots, at times, getting in each other's way. In our team of robots used for the AAAI competition, described in Section 6, kin recognition is used as a means for distributing the agents more uniformly through the environment, keeping them apart and thus reducing interference.

Parker, now at the Oak Ridge National Laboratories, developed the Alliance architecture as a means for expressing and controlling heterogeneous teams of robotic systems [15.]. Researchers at the University of Michigan have used distributed artificial intelligence techniques to control small groups of robots [12.]. There is also extensive research being conducted in Japan on multirobot teams [10.].

Canadian researchers have developed a useful taxonomy for characterizing the various research approaches being developed [8.]. They are subdivided along the following lines:

- Team Size: 1, 2, size-limited, and size-infinite
- Communication Range: None, Near, Infinite
- Communication Topology: Broadcast, Addressed, Tree, Graph
- Communication Bandwidth: High, Motion-related, Low, Zero
- Team Reconfigurability: Static, Coordinated, Dynamic
- Team Unit Processing Ability: Non-linear summation, Finite State Automata, Push-Down Automata, Turing Machine Equivalent
- Team Composition: Homogeneous, Heterogeneous

Another starting point for further understanding the general issues and research efforts ongoing in robotic teams appears in a review article [7.].

2.2 Reactive Schema-based Behavioral Control

Reactive behavioral control [2.] is now a well established technique for providing rapid real-time response for a robot by closely tying perception to action. Behaviors, in various forms, are the primary building blocks for these systems, which typically operate without conventional planning or the use of global world models.

Schema-based systems [1.] are a form of reactive behavioral control that are further characterized by their neuroscientific and psychological plausibility, the absence of arbitration between behaviors (schemas), the fusion of behavioral outputs through the use of vector summation in a manner analogous to the potential fields method [11.], inherent flexibility due to the dynamic instantiation and deinstantiation of behaviors on an as-needed basis, and easy reconfigurability through the use of high-level planners or adaptive learning systems.

Motor schemas are the basic building blocks of a schema-based system. These motor behaviors have an associated perceptual schema which provides only the necessary sensory information for that behavior to react to its environment, and ideally nothing more. Perceptual schemas are an embodiment of action-oriented perception, where perception is tailored to the needs of the agent and its surrounding environment. Each motor schema produces a single vector that provides the direction and strength of the motor response for a given stimuli. All of the active behaviors' vectors are summed together, normalized, and sent to the actuators for execution.

Another coordination operator, temporal sequencing, ties together separate collections of behaviors (assemblages) and provides a means for transitioning between them [4.]. Typically, perceptual triggers are defined which monitor for specific events within the environment. If a relevant event is detected, a state transition occurs resulting in the instantiation of a new behavioral assemblage. Finite state acceptor (FSA) diagrams are typically used to represent these relationships. Examples of these diagrams appear in Sections 4 and 6.

3 Formation Control

Figure 1: One of Lockheed-Martin's HMMWVs.

Sections 3-5 focus on multiagent research in support of ARPA's Unmanned Ground Vehicle (UGV) Demo II program. The goal of this project is to field a team of robotic scout vehicles for the U.S. Army. At present, scout platoons are composed of four to six manned vehicles equipped with an array of observation and communication equipment. The scouts typically move in advance of the main force, to report on enemy positions and capabilities. It is hoped that robotic scout teams will do as well as humans for this task, while removing soldiers from harm's way. Lockheed-Martin has built four prototype robot scout vehicles, based on the HMMWV (Figure 1). This section outlines the design of an important behavior for scout teams: formation maintenance. The next two sections look at related issues: a way for humans to express military missions for robot team execution and providing variable levels of human intervention during an ongoing mission.

Scout teams use specific formations for a particular task. In moving quickly down roadways for instance, it is often best to follow one after the other. When sweeping across desert terrain, line-abreast may be better. Furthermore, when scouts maintain their positions, they are able to distribute their sensor assets to reduce overlap. Army manuals [17.] list four important formations for scout vehicles: *diamond*, *wedge*, *line* and *column*. Four simulated robots moving in these formations are pictured in Figure 2.

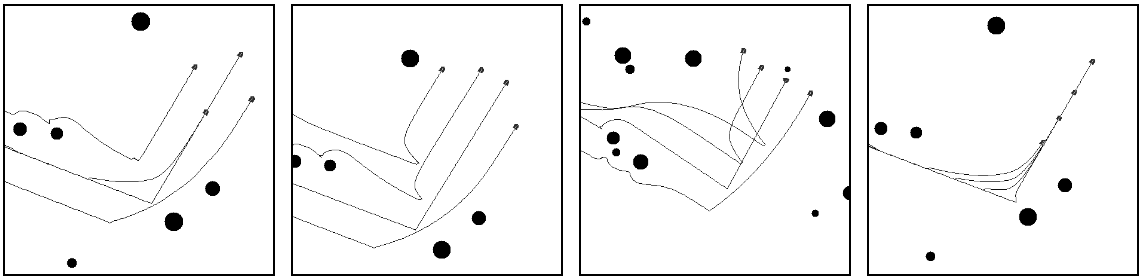

Figure 2: Four robots in leader-referenced *diamond*, *wedge*, *line* and *column* formations executing a 90 degree turn in an obstacle field.

3.1 Motor Schemas for Formation

The formation behavior must work in concert with other navigational behaviors. The robots should concurrently strive to keep their relative formation positions, avoid obstacles and move to a goal location. Formation behaviors for 2, 3 and 4 robots have been developed and initially tested in simulation. They have been further tested on two-robot teams of Denning robots and Lockheed-Martin UGVs. The formation behaviors were developed using the motor schema paradigm (Sec. 2.2) within Georgia Tech's *MissionLab* environment. Each motor schema, or primitive behavior, generates a vector representing a desired direction and magnitude of travel. This approach provides an easy way to integrate behaviors. First, each vector is multiplied by a gain value, then all the vectors are summed and the result is normalized. The gain values express the relative strength of each schema. A high-level representation of this behavioral integration is illustrated as pseudo-code in Figure 3.

```
while (task not completed)
  {
        /* Compute motor vectors in parallel.
           Each schema may call embedded perceptual
           schemas.  The result is multiplied by a
           gain value, indicating its relative strength. */
       execute_in_parallel
            {
               /* Compute repulsion from sensed obstacles. */
               vector1 = obstacle_gain * avoid_obstacle_schema(sense_obstacles);

               /* Compute attraction to the goal location. */
               vector2 = goal_gain * move_to_goal_schema(sense_goal);

               /* Compute attraction to formation position.
                  This depends on the type of formation, the heading of the
                  group, and the locations of the other robots. */
               vector3 = formation_gain *
                         maintain_formation_schema(
                               detect_formation_position( formation_type,
                                       detect_formation_heading(),
                                       sense_robots()), my_position);
            } /* end parallel execution */

     /* Move the robot according to the normalized sum of the  motor vectors */
          move_robot(normalize(vector1 + vector2 + vector3));
  }
```

Figure 3: Pseudo-code showing the behavioral assemblage for a robot to move to a goal, avoid obstacles and maintain a formation position.

The formation behavior itself is comprised of two main components: a perceptual schema **detect-formation-position**, and a motor schema **maintain-formation**. The perceptual schema determines where the robot should be located based on the formation type in use, the robot's relative position in the overall formation, and the locations of the other robots. **Maintain-formation** generates a vector towards the correct position, with the magnitude based on how far out of position the robot finds itself.

Three different approaches for determining a robot's position in formation are described in [5.]. Here we will present the unit-center approach, where the position depends on the locations of the other robots, the overall unit heading, and the formation type. A unit-center is computed by averaging the positions of all the robots involved in the formation, then each robot determines its own formation position relative to that center.

A vector generated by **maintain-formation** always directs the robot from its current position towards the formation position. It varies from zero magnitude to a maximum value depending on how far out of position the robot is (Figure 5):

- **Ballistic zone**: the robot is far out of position, so the output vector's magnitude is set at its

```
detect_formation_position(formation_type, formation_heading, robot_positions)

        /* The unit-center is the average of the robot locations.*/
       unit_center = average(robot_positions);

        /* Now compute where the robot should be if in perfect position.
           A lookup table stores the proper positions for each robot for
           each type of formation.  The value must be rotated and added
           to the unit-center to shift from local to global coordinates. */

       local_position = lookup_table[formation_type, my_position_number];
       correct_position = rotate(local_position, formation_heading) + unit_center;
       return(correct_position);


maintain_formation_schema(correct_position, current_position);

        /* Compute the vector from the present position of the
           robot to the correct position for formation. */
       initial_vector = correct_position - current_position;

        /* Adjust the magnitude of the vector based on
           dead-zone-radius and controlled-zone-radius. */
       vector = adjust(initial_vector, dead_zone_radius, controlled_zone_radius);
       return(vector);
```

Figure 4: Pseudo-code for the **detect-formation-position** perceptual schema, and the **maintain-formation-position** motor schema.

maximum, which equates to the schema's gain value, with its directional component pointing towards the center of the computed dead zone.

- **Controlled zone**: the robot is somewhat out of position and the output vector's magnitude decreases linearly from a maximum at the farthest edge of the zone to zero at the inner edge. The directional component is towards the dead zone's center.
- **Dead zone**: the robot is within acceptable positional tolerance. Within the dead zone the vector magnitude is always zero.

Pseudo-code for this computation and for determining the robot's formation position are given in Figure 4. These behaviors were ported to Lockheed-Martin's UGVs and successfully demonstrated at Demo C on two UGVs in Denver, Colorado in the summer of 1995.

4 Mission Specification for Multi-robot Systems

Another pressing problem for the UGV Demo II program in particular and for robotics in general is how to provide an easy-to-use mechanism for programming teams of robots, making these systems more accessible to the end-user. Towards that end, the *MissionLab* mission specification system has been developed [13.]. An agent-oriented philosophy is used as the underlying methodology, permitting the recursive formulation of societies of robots.

A society is viewed as an agent consisting of a collection of either homogeneous or heterogeneous robots. Each individual robotic agent consists of assemblages of behaviors, coordinated in various ways. Temporal sequencing [4.] affords transitions between various behavioral states which are naturally represented as a finite state acceptor. Coordination of parallel behaviors can be accomplished via fusion (vector summation), action-selection, priority (e.g., subsumption) or other means as necessary. These individual behavioral assemblages consist of groups of primitive perceptual and motor behaviors which ultimately are grounded to the physical sensors and actuators of a robot.

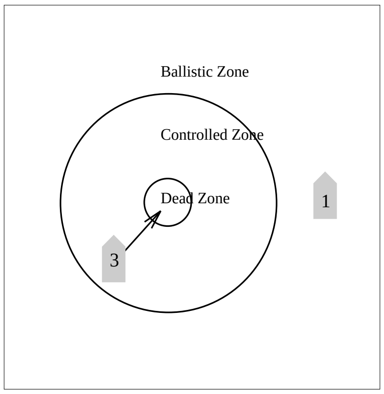

Figure 5: Zones for the computation of **maintain-formation** magnitude

An important feature of *MissionLab* is the ability to delay binding to a particular behavioral architecture (e.g., schema-based, SAUSAGES, subsumption) until after the desired mission behavior has been specified. Binding to a particular physical robot occurs after specification as well, permitting the design to be both architecture- and robot-independent.

MissionLab's architecture appears on the left of Figure 6. Separate software libraries exist for the abstract behaviors, and the specific architectures and robots. The user interacts through a design interface tool (the configuration editor) which permits the visualization of a specification as it is created. The right side of Figure 6 illustrates an example *MissionLab* configuration that embodies the behavioral control system for one of the robots used in the AAAI robot competition (Sec. 6). The individual icons correspond to behavior specifications which can be created as needed or preferably reused from an existing repertoire available in the behavioral library. Multiple levels of abstraction are available, which can be targeted to the abilities of the designer, ranging from whole robot configurations, down to the configuration description language for a particular behavior.

After the behavioral configuration is specified, the architecture and robot types are selected and compilation occurs generating the robot executables. These can be run within the simulation environment provided by *MissionLab* (Fig. 7 left) or, through a software switch, they can be downloaded to the actual robots for execution (Fig. 7 right).

MissionLab was demonstrated at UGV Demo C in the Summer of 1995 to military personnel. *MissionLab* is available via the world wide-web at:

http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab.html.

5 Team Teleautonomy

Another important control aspect is concerned with the real-time introduction of a commander's intentions to the ongoing operation of an autonomous robotic team. We have developed software in the context of the UGV Demo II program to provide this capability in two different ways.

- **The commander as a behavior**. In this approach a separate behavior is created that permits the commander to introduce a heading for the robot team using an on-screen joystick (Fig. 8 left). This biases the ongoing autonomous control for all of the robots in a particular direction. Indeed, all other behaviors are still active, typically including obstacle avoidance and formation maintenance. The output of this behavior is a vector which represents the commander's directional intentions and strength of command. All of the robotic team members have the same behavioral response to the operator's goals and the team acts in concert without any knowledge of each other's behavioral state.

- **The commander as a supervisor**. With this method, the operator is permitted to conduct behavioral modifications on-the-fly. This can occur at two levels.

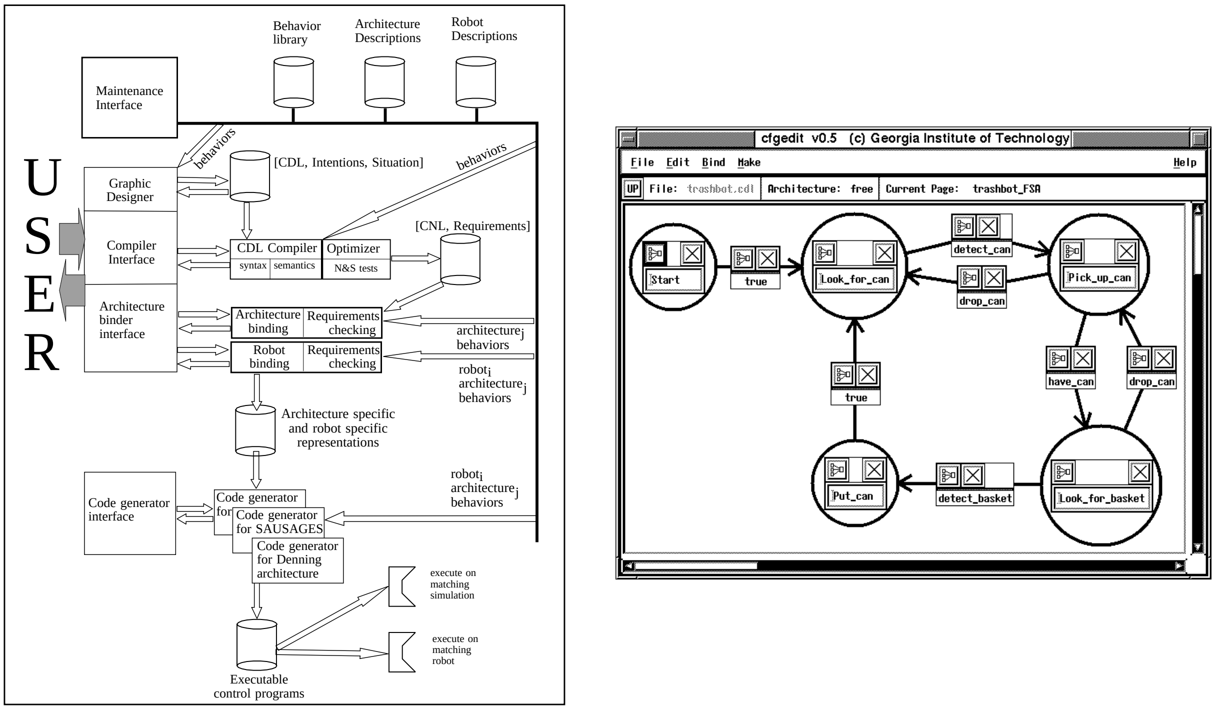

Figure 6: *MissionLab*. The system architecture appears on the left. A finite state configuration corresponding to the AAAI Robots appears on the right. This FSA differs slightly from the version implemented in Figure 12.

- For the knowledgeable operator, the low-level gains and parameters of the active behavioral set can be adjusted directly if desired, varying the relative strengths and behavioral composition as the mission progresses.
- For the normal operator, behavioral traits ("personality characteristics") are abstracted and presented to the operator for adjustment. These include such things as aggressiveness (inversely adjusting the relative strength of goal attraction and obstacle avoidance) and wanderlust (inversely varying the strength of noise relative to goal attraction and/or formation maintenance) (Fig. 8 right). These abstract qualities are more natural for the operator unskilled in behavioral programming and permit the concurrent behavioral modification of all of the robots in a team according to the commander's wishes in light of incoming intelligence reports.

An example illustrating the utility of the directional control approach is in the extrication of teams from potential traps. In Figure 9, a run is shown using two of our Denning Mobile robots. The active behaviors include **avoid-static-obstacle**, **move-to-goal**, and **column-formation**. The robots wander into the box canyon and become stuck trying to make their way to the goal point specified behind the box-canyon (top-left photograph). The operator intervenes, using the joystick to direct the robots to the right. While moving they continue to avoid obstacles and maintain formation. Once clear of the trap (top-right photograph), the operator stops directing the robots and they proceed autonomously to their goal. The overall execution trace is depicted at the bottom of Figure 9.

The directional control team teleautonomy software has been successfully integrated by Lockheed-Martin into the UGV Demo II software architecture and was demonstrated in simulation to military

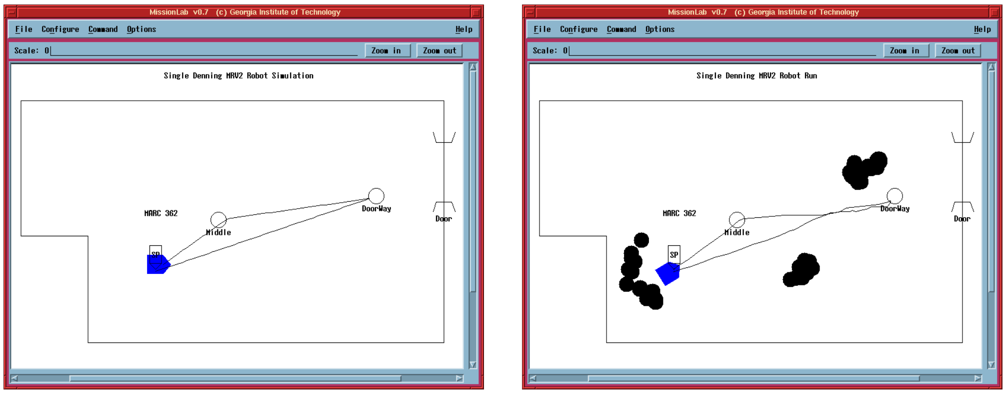

Figure 7: Left: Simulated Run on Denning Robot. Right: same code executed on actual Denning Robot

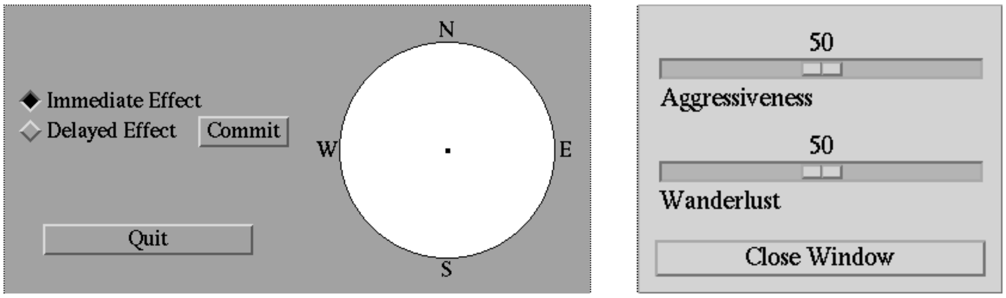

Figure 8: Left: On-screen Directional ControlRight: Personality slider bars

observers during UGV Demo C. Both directional and personality control have been integrated into the *MissionLab* system described above and is available in the *MissionLab* release via the world wide web. Additional information on team teleautonomy can be found in [3.].

6 A Team of Trash-collecting Robots

This section describes a team of robots designed for trash-collecting. Specifically, the task for these robots is to gather items of trash, primarily red soda cans, and deposit them near blue wastebaskets. They must operate in an office environment including obstacles like tables and chairs. The design we present builds on motor schema research presented in earlier sections of the chapter. These robots show how simple individual primitive behaviors may be combined, sequenced and instantiated on several robots to yield a successful cooperating team. A detailed account of this effort is available in [6.].

Io, Ganymede and Callisto (Figure 10) were built primarily with off-the-shelf, commercially available components. The base was purchased as a radio-controlled tank with two separately motorized treads. Motor and perceptual schemas run on a PC-compatible motherboard, while control and sensing tasks are implemented on a separate micro-controller board. Each robot is equipped with a forward looking color camera and a gripper for grasping trash (Figure 11).

The robots use color vision to find trash items (attractors), other robots (kin), and wastebaskets. To facilitate the vision task, the robots were painted bright green, trash items are presumed to be red (Cola Cans) and wastebaskets are blue recycling containers. A set of sequenced behavioral assemblages, presented next, leads the robots through the states necessary to complete the task.

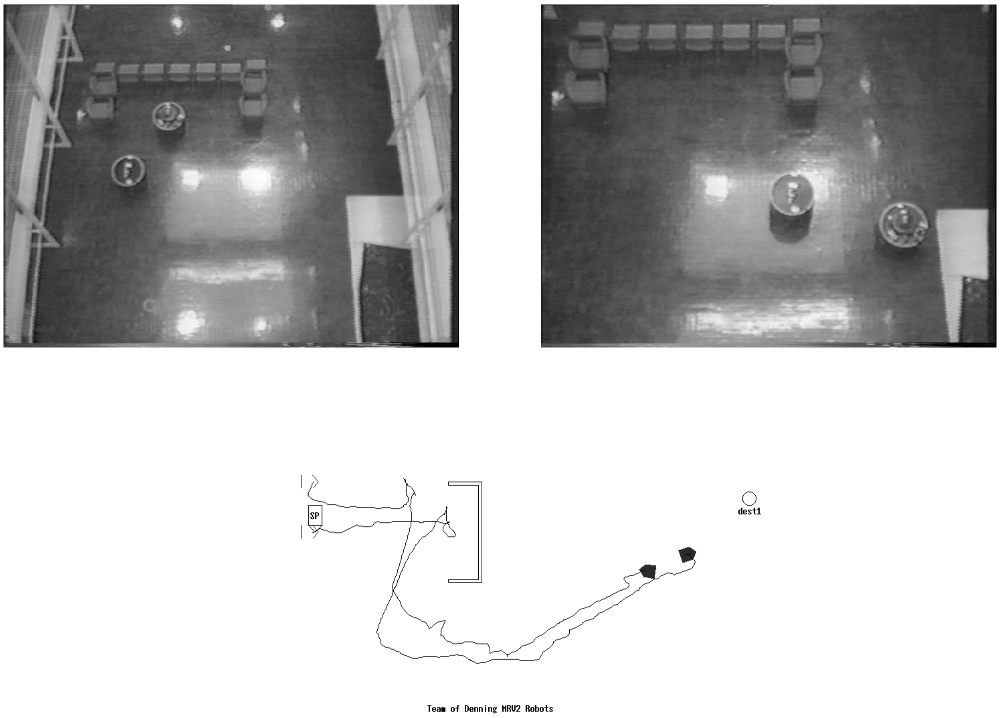

Figure 9: Teleautonomous extrication from a box canyon of a team of 2 Denning mobile robots (viewed from above).
Top: Robots trapped in box canyon (left) and after teleautonomous removal (right).
Bottom: Execution Trace of Robotic Run, (rotated 90 degrees clockwise relative to the photographs above).

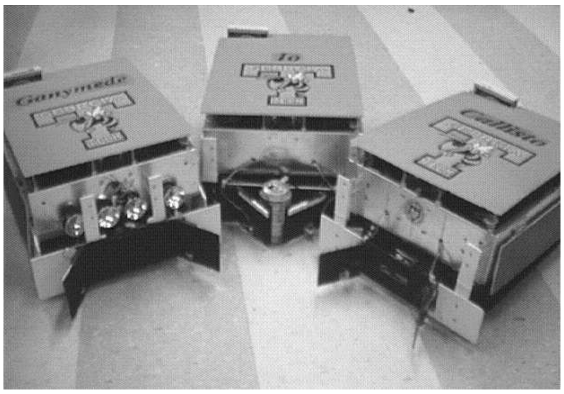

Figure 10: Ganymede, Io, and Callisto.

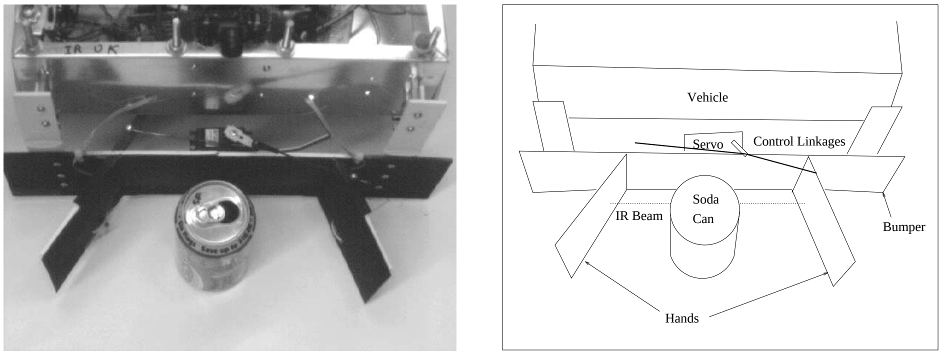

Figure 11: Close-up of trash manipulator.

6.1 Behaviors for Trash-collecting

This task lends itself to a sequential state-based solution: search for trash objects; if one is found, move towards it; if close enough, grasp it; now look for a wastebasket; and so on. The sequence of states is naturally represented as a Finite State Acceptor (FSA). The sequence developed for our robots is depicted in Figure 12. Paartial pseudo-code for the FSA is shown in Figure 13.

We first examine the behavioral design starting at the highest level of abstraction, then successively narrowing our focus to lower levels of the sensor software. Each state, or behavioral assemblage, in the FSA represents a group of activated perceptual and motor schemas. Outputs of the active motor schemas are combined as described earlier (Section 2.2) and output for motor control. Pseudo-code for one of the assemblages, **wander-for-trash** is listed in Figure 14.

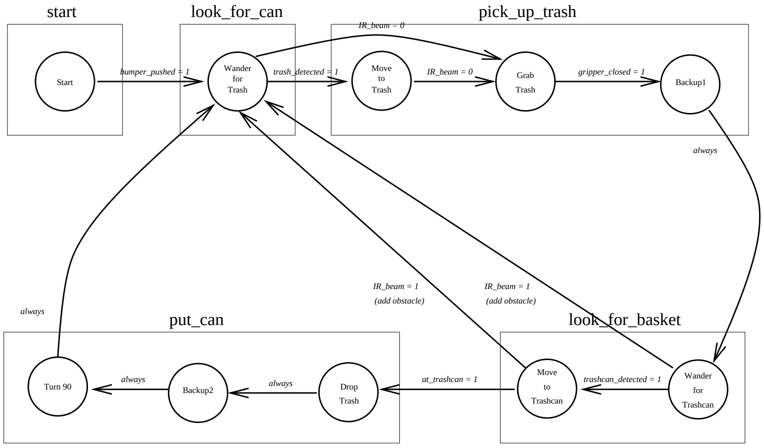

Figure 12: Implemented robot behavioral state diagram for the trash-collecting task. The blocks correspond to the more abstract states depicted in Figure 7. Some additional transitions were added for fault-tolerance.

6.2 Image Processing for Kin Recognition

We now focus the perceptual component of the **wander-for-trash** behavior concerned with kin recognition. The perceptual schema **sense-robots** uses a combination of vision and short-term memory to track other robots. When another robot is detected visually, its position is noted and added to a list of robots recently seen (a check is made to ensure there are no duplicate entries). Entries are removed when they get stale after 60 seconds. Short-term memory is important since nearby agents are often lost from view as the robot moves or turns. Pseudo-code for **sense-robots** appears in Figure 15.

```
state = START;
do forever
        /*
         * Execute behavioral assemblages according to which state the
         * agent is in.
         *
         * bumper_pushed == 1 means one of the contact sensors is pushed.
         * ir_beam == 1       means the IR beam in the gripper is intact.
         *                    It is 0 if the beam is broken, in the case of
         *                    a potential trash object.
         */
        switch(state)
                case START:
                        do forever
                                start();
                                if (bumper_pushed == 1)
                                        state = WANDER_FOR_TRASH;
                                        break; /* out of do forever */
                        break; /* out of this state */
                case WANDER_FOR_TRASH:
                        do forever
                                wander_for_trash();
                                if (bumper_pushed == 0 and ir_beam == 0)
                                        state = GRAB_TRASH;
                                        break; /* out of do forever */
                                else if (trash_detected == 1)
                                        state = MOVE_TO_TRASH;
                                        break; /* out of do forever */
                        break; /* out of this state */
                case MOVE_TO_TRASH:
                        do forever
                                move_to_trash();
                                if (trash_detected == 0)
                                        state = WANDER_FOR_TRASH;
                                        break; /* out of do forever */
                                else if (bumper_pushed  == 0 and ir_beam == 0)
                                        state = GRAB_TRASH;
                                        break; /* out of do forever */
                        break; /* out of this state */

        /* OTHER STATES DELETED FOR BREVITY */
```

Figure 13: Partial pseudo-code implementing three states of the FSA for a sequenced trash-collecting behavior. This is the highest level control code for each robot.

```
wander_for_trash();

        /* Compute motor vectors in parallel.
           Each schema may call embedded perceptual
           schemas.  The result is multiplied by a
           gain value, indicating its relative strength. */
        execute_in_parallel {

                /* Compute repulsion from sensed obstacles. */
                vector1 = obstacle_gain * avoid_obstacle_schema(sense_obstacles);

                /* Add a random conponent to motion to avoid local minima */
                vector2 = noise_gain * compute_noise();

                /* Compute repulsion from sensed robots. */
                vector3 = robot_gain * avoid_robot(sense_robots());
          } /* end parallel */

        /* Move the robot according to the normalized sum of the motor vectors */
        move_robot(normalize(vector1 + vector2 + vector3));
```

Figure 14: Pseudo-code for the **wander-for-trash** state. In this state, the robot is repulsed by other robots.

```
sense_robots();

        /* Remove stale memories of other robot locations. */
        robot_list = age(robot_list);

        /* Get a list of visually acquired robots. */
        visual_robots = find_robots_in_image();

        /* Add new visually-acquired robots to the list, and
           refresh the memory of ones that we've seen again. */
        for (each robot in visual_robots)
                if (robot not in robot_list)
                        add_to_list(robot_list, robot);
                else
                        refresh(robot_list, robot);

        return(robot_list);
```

Figure 15: Pseudo-code for the **sense-robots** perceptual schema.

```
find_robots_in_image();
        detected_robot_list = empty;

         /* grab the image */
        image = digitize_image();

         /* enhance and threshold green component */
        for (each pixel x and y )
                supergreen[x,y] = image.green[x,y] - (image.red[x,y] + image.blue[x,y]);
                thresholded[x,y] = threshold(threshold_value, supergreen[x,y]);

         /* Extract blobs of pixels above the threshold from the green image */
        blobs = find_blobs(thresholded, min_blob_size);

        for (each blob in blobs)

            /* compute range and azimuth to the blob based on camera parameters */
                blob.range = camera_height * arctan(blob.bottom_pixel * degrees_per_pixel);
                blob.azimuth = blob.center_pixel * degrees_per_pixel;

            /* estimate the other robot's relative position, +x is forward */

                local_robot = convert_from_polar(blob.range, blob.azimuth);

            /* convert into global coordinates and add to list */
                global_robot = rotate(local_robot, -current_heading) + current_position;

                add_to_list(detected_robot_list, global_robot);
        return(detected_robot_list);
```

Figure 16: Pseudo-code for **find-robots-in-image**, which processes a color image to find the green blobs corresponding to other robots. For these computations, the center of the camera image is assumed to be [0,0] and negative azimuth is to the left. The robot's egocentric coordinates have +y straight-ahead and +x to the right. The robot tracks its position and the position of other robots.

Robots are detected visually in the **find-robots-in-image** routine (Figure 16). We were limited to simple image processing since complex schemes would limit the robot's real-time performance, or might not have fit in the limited RAM. Overall, the approach is to isolate colored blobs in the robot camera's view, then use simple trigonometry to estimate the locations of objects associated with the blobs in the environment. Processing for green blobs finds the positions of other visible robots (kin).

The first step is to extract separate red, green and blue components from the image. One might begin the search for green objects by simply inspecting the green image component, but a problem with using just one of the color components occurs because many bright objects (e.g., specular reflections) have strong red and blue components in addition to green. In other words, one cannot infer that an object is green just because it has a large green component alone. To get around the problem, a super-component for each primary color was computed first. Super-components for one color are computed by subtracting each of the other color components from it. Supergreen, for example, is computed as $green - (red + blue)$. So a white blob (as in a specular reflection) will have low supergreen component, while a green blob will have a bright supergreen component. This approach significantly improves performance when searching for specifically colored objects. A sample luminance and supergreen image are shown in Figure 17. Notice how well the green robot stands out.

After the green super-component is computed, the resultant image is thresholded so that pixels below a certain value are set to 0; those above are set to 255. Although the system is not extremely sensitive, the best performance results when the threshold value is tuned for ambient lighting conditions. Groups of adjoining bright pixels (255) are classified as a blob if there are more than a certain number of them in the group. The minimum blob size is typically set to 100 pixels. The angular offset (azimuth) to the object is computed from the field of view of the camera (pixels per degree). We are able to compute range to an object by finding the lowest pixel of the corresponding blob. Since all perceptual objects (trash, robots and wastebaskets) rest on the floor and the camera sits at a fixed height above the ground, all objects the same distance away will lie equally low on the image plane. Range is estimated using the trigonometric relation:

$$r = \frac{h}{tan^{-1}(\theta)} \tag{1}$$

where r is range, h is the camera height, and θ is the apparent angle to the bottom of the object from the center of the image. θ may be computed by counting pixels from the center of the image to the bottom of the blob, based on the number of degrees covered by one pixel. In a final step, the range and azimuth data are converted to cartesian coordinates within the robot's reference frame. The entire process takes about one second.

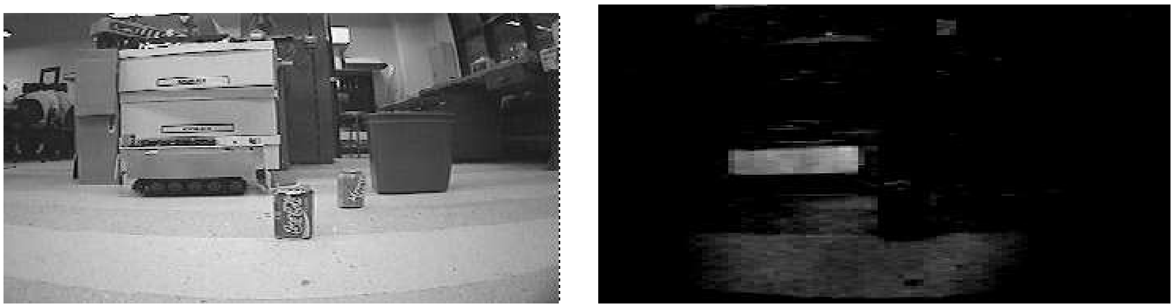

Figure 17: A robot's-eye view of a laboratory scene including another green robot, two red soda cans and a blue wastebasket. The image on the right shows the image after processing to highlight green pixels. Notice how well the robot stands out.

6.3 Performance of the Trash-collecting Team

The robots competed in three preliminary trials and in the final competition, winning the Clean-up task at the AAAI-94 Mobile Robot Contest [16.]. In our laboratory, the robots have sometimes gathered as many as 20 soda cans in 10 minutes. But in the best run at the contest, the robots collected only 15 cans. One reason for not equaling the earlier performance is that the contest required the use of black wastebaskets, rather than the blue ones the robots had been designed for.

The competition team revised the image processing software to seek dark objects, as opposed to blue ones in the **move-to-trashcan** phase. Unfortunately, this led to robots occasionally "hiding" soda cans under tables or other shadowy areas, as they confused them with wastebaskets. Finally, we took advantage of another perceptual clue for wastebaskets: the fact that from a robot's point of view all wastebaskets must cut through the horizon. In other words, part of the wastebasket is below camera level, and part of it is above camera level. This ruled out most non-wastebasket blobs.

Except for the complications regarding black wastebaskets, the system performs very well. The robots easily find red soda cans and move quickly towards them. The robot-robot repulsion is usually obvious to the observer and clearly helps the team spread out in the search phase.

7 Summary and Open Questions

In this chapter we have presented several of the many aspects of multiagent control. These include the coordination of motion to maximize effort as seen in formation maintenance; methods by which programming of multiagent systems can be made easier through the use of visual programming using tools such as *MissionLab*; and methods by which a human operator can effectively interact with teams of mobile agents without becoming overwhelmed by the sheer numbers of robots. Two very different problem domains have motivated our research: military scouts and janitorial robots. Successful implementations of many of these ideas have been completed in these domains.

Other important questions still confront multiagent robotics researchers. A few include:

- How can we introduce adaptation and learning to make these systems more flexible within a changing environment?
- How can we ensure robust inter-robot communication that is both task and environment sensitive?
- How well will these ideas scale to large swarms of robots on the order of perhaps ten thousand or more?
- How can biological systems inform us to ensure we are providing a sound ecological fit of the robot to its environment, producing long-term survivable systems?

Our laboratory and others continue to pursue answers to these and other important questions regarding cooperative multiagent robotic systems.

Acknowledgments

Funding to support this research has been provided by the National Science Foundation under Grant #IRI-9100149 and ARPA/ONR under Grant #N00014-94-1-0215. Doug MacKenzie and Khaled Ali provided the figures for *MissionLab* and team teleautonomy respectively and are primarily responsible for their development. Matt Morgenthaler and Betty Glass at Lockheed-Martin were primarily responsible for the porting of software developed in Georgia Tech laboratories to the UGV Demo II vehicles. Development of the trash-collecting robots was funded by the CIMS/AT&T Intelligent Mechatronics Laboratory at Georgia Tech. The AAAI provided a generous grant for travel to the AAAI-94 competition in Seattle.

References

1. Arkin, R.C., 1989. "Motor Schema Based Mobile Robot Navigation", *International Journal of Robotics Research*, vol 8(4), pp. 92-112.

2. Arkin, R.C., "Reactive Robotic Systems", article in *Handbook of Brain Theory and Neural Networks*, ed. M. Arbib, MIT Press, pp. 793-796, 1995.

3. Arkin, R.C. and Ali, K., "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems", *Proc. Third International Conference on Simulation of Adaptive Behavior, (SAB94) [From Animals to Animats]*, Brighton, England, Aug. 1994, pp. 473-478.

4. Arkin, R. and MacKenzie, D., "Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation", *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 3, June 1994, pp. 276-286.

5. Balch, T., and Arkin, R.C., "Motor schema-based formation control for multiagent robot teams", *First International Conference on Multiagent Systems*, June 1995, San Francisco, pp. 10-16.

6. Balch, T., Boone, G., Collins, T., Forbes, H., MacKenzie, D., and Santamaría, J. "Io, Ganymede and Callisto - a multiagent robot trash-collecting team", *AI Magazine*, 16(2):39–51, 1995.

7. Cao, Y., Fukunaga, A., Kahng, A., and Meng, F., "Cooperative Mobile Robotics: Antecedents and Directions", *Proc. 1995 IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS '95)*, Pittsburgh, PA, Vol. 1, pp. 226-34 Aug. 1995.

8. Dudek, G., Jenkin, M., Milios, E., and Wilkes, D., 1993. "A Taxonomy for Swarm Robots", *Proc. 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Yokohama, Japan, pp. 441-447.

9. Fukuda, T., Nakagawa, S., Kawauchi, Y., and Buss, M., "Structure Decision for Self Organising Robots Based on Cell Structures - CEBOT", *IEEE International Conference on Robotics and Automation*, Scottsdale Arizona, 1989, pp. 695-700.

10. International Symposium on Distributed Autonomous Robotic Systems, RIKEN, Saitama, Japan, 1992 and 1994.

11. Khatib, O., "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", *Proc. IEEE Int. Conf. Robotics and Automation*, p. 500, St. Louis, 1985.

12. Lee, J., Huber, M., Durfee, E., and Kenny, P., "UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications", *AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*, March 1994, pp. 842-849.

13. MacKenzie, D., Cameron, J., Arkin, R., "Specification and Execution of Multiagent Missions", *Proc. 1995 Int. Conf. on Intelligent Robotics and Systems IROS '95*, Pittsburgh, PA, Vol. 3, pp. 51-58.

14. Mataric, M., "Minimizing Complexity in Controlling a Mobile Robot Population", *Proc. IEEE International Conference on Robotics and Automation*, Nice, FR, May 1992.

15. Parker, L., "Adaptive Action Selection for Cooperative Agent Teams", *From Animals to Animats: Proc. 2nd International Conference on the Simulation of Adaptive Behavior*, MIT Press/Bradford Books, Honolulu, HI, 1993, pp. 442-450.

16. Simmons, R., "The 1994 AAAI mobile robot competition and exhibition", *AI Magazine*, 16(2):19–30, 1995.

17. U.S. Army, *Field Manual No 7-7J.* Department of the Army, Washington, D.C., 1986.