IO, GANYMEDE, and CALLISTO
A Multiagent Robot Trash-Collecting Team

Tucker Balch, Gary Boone, Thomas Collins, Harold Forbes, Doug MacKenzie and Juan Carlos Santamaría

■ The Georgia Institute of Technology won the Office Cleanup event at the 1994 AAAI Robot Competition and Exhibition with a multirobot cooperating team. This article describes the design and implementation of these reactive trash-collecting robots, including details of multiagent cooperation, color vision for the detection of perceptual object classes, temporal sequencing of behaviors for task completion, and a language for specifying motor schema–based robot behaviors.

The team of robots from the Georgia Institute of Technology, IO, GANYMEDE, and CALLISTO (figure 1), placed first in the Office Cleanup event at the 1994 Robot Competition and Exhibition sponsored by the American Association for Artificial Intelligence (AAAI). The contest required competing robot entries to clean up a messy office strewn with trash. Wads of paper, Styrofoam coffee cups, and soda cans were placed by judges throughout the contest arena along with wastebaskets, where they hoped the robots would deposit the trash. The arena included ordinary tables, chairs, and desks as obstacles to provide realism. During competitive trials, each robot was to gather and throw away as much trash as possible in 10 minutes. Points were awarded for each piece of trash thrown away and penalties levied for collisions with walls or obstacles. The task proved difficult. Only one robot, CHIP, from the University of Chicago, was equipped to autonomously locate, pick up, and deposit trash in a wastebasket. Unfortunately, the computational overhead was so great that CHIP was only able to perform the task once in 10 minutes. The rules provided for robots with less capable—or no—manipulators by permitting *virtual manipulation*. If a robot was near an item of trash or a wastebasket, it could signal its intent to pick up or throw away trash and be credited for the pick up or drop off at a slight penalty.[1] Another top competitor in the Office Cleanup event, RHINO, is also described in this issue (see the article by Buhmann and his colleagues). Readers interested in the overall competition are referred to Reid Simmons's article, also in this issue.

Georgia Tech's approach differed from other entries in the event by emphasizing multiple, low-cost robots instead of a single (usually expensive) robot solution. This approach was motivated by several factors, including cost, but primarily by a desire to test theories regarding multiagent reliability and cooperation. Implementation of this multirobot system is interesting from several standpoints: (1) low-cost hardware permits construction of several robots; (2) reactive behaviors are used by the robots to collect trash; (3) cooperative behaviors provide for cooperation between robots; (4) temporal sequencing coordinates transitions between distinct operating states for each robot and achieves the desired goal state; (5) fast vision locates soda cans, wastebaskets, and robots; (6) behavior and hardware definition language (BHDL) describes robot hardware and specifies behavioral states and transitions between them; and (6) a real-time executive instantiates and executes processes at run time according to a BHDL file.

The robots' hardware design and vision processing are outlined in the next section. Later sections describe behavioral control, temporal sequencing, software architecture, and multiagent cooperation. The article closes with strategies used and lessons learned at the competition.

Hardware Design

The 10-pound robots were built using off-the-shelf components at a cost of approximately $1700 each. A 13- by 17-inch aluminum chassis encloses a PC-clone motherboard and floppy disk, a microcontroller, and the power

Figure 1. Ganymede, Io, and Callisto.

system (figure 2). The chassis sits atop a motorized base purchased as a radio-controlled–model kit. Each robot is equipped with bumper sensors, a miniature color camera, and a specially designed mechanical gripper. Off-the-shelf components were chosen for cost, flexibility, and reliability, enabling the design team to concentrate on software development and integration. Robot locomotion is provided by an inexpensive direct-current, motor-powered, treaded vehicle marketed for radio-controlled–model enthusiasts. Each robot base is driven by two separate motors and drive trains. Initially, off-the-shelf radio-controlled–car controllers were used to drive the motors, but these controllers were abandoned because nonlinearities in their output made accurate position control impossible. Instead, custom bipolar H-bridges were built, allowing smooth control at low speed.

Processing responsibilities are split between a 68332 business card computer (BCC) for low-level sensing and control and a 386 computer for high-level control and vision. Each robot's pair of H-bridge motor drivers is controlled by the BCC, which implements proportional, integral, and derivative (PID) speed control with trapezoidal velocity profiles. The BCCs were chosen for ease of programming (in C) and multiple digital input-output and timer channels. The microcontroller is also responsible for driving and detecting the gripper's infrared beam, driving the gripper's open-close servo, and reading the robot's eight bumper switches. The requirement for vision motivated the addition of a PC motherboard. In addition to providing processing speed and memory that is unavailable in the microcontroller, the PC enables the use of inexpensive, mass-market video cards. Each robot has a 33-megahertz 80386 central processing unit with 4 megabytes of random-access memory and a floppy disk. Small, low-power videocameras supply NTSC (National Television System Committee) video directly to video-capture cards. After booting from a floppy disk, the PC downloads software to the BCC over a serial link. Once the low-level BCC software begins execution, the serial link is used to exchange sensor and command information between the BCC and the PC. After boot up, the PC takes over vision and high-level motor-processing responsibilities, and the BCC serves as a low-level sensor and actuator device. Dividing the sensing and processing responsibilities between the PC and the BCC enforced modularity, isolated errors, and allowed parallel development.

Manipulator Design

It was immediately apparent to the design team that most commercial, general-purpose robot arms are too heavy, too slow, and too power hungry for small mobile robots. Task requirements were the starting point of the manipulator design: (1) capture empty soda cans, Styrofoam cups, and wadded sheets of notebook paper that might be in several orientations; (2) be lightweight; (3) use low power; and (4) withstand repeated collisions with immovable objects.

Other constraints affect the manipulator design as well. Because vehicle maneuvering is imprecise, the manipulator should maximize the capture area for the various types of trash. Also, trash objects are below the view of the camera when in manipulator range; so, the gripper should allow the robot to grasp objects it cannot see. Of the various types of trash, soda cans are the largest; so, the manipulator was designed for them and occasionally checked against Styrofoam cups and paper wads.

The final design (figure 3) is able to capture and hold soda cans in three orientations: upright, on side with the long axis perpendicular to the gripper, and on side with the long axis parallel to the gripper. Two short "hands" are hinged directly to a bumper that is designed to support and protect the hands. The servoactuator is attached to the rear of the bumper with control linkages connecting it to the hands. Construction materials are steel hinges, aluminum control horns, and plastic boards and fiberboards held together with pop rivets. An infrared beam in the gripper's hands alerts the robot of an object in range for grasping. The sensor helps the robot opportunistically grab trash that it encounters. The sensor is tripped by any intervening object, including obstacles; so, the robot must somehow discriminate between obstacles and trash. Whenever the infrared beam is broken, the robot closes the manipulator, then backs up. If the object encountered is immovable (for example, a chair leg), it slips

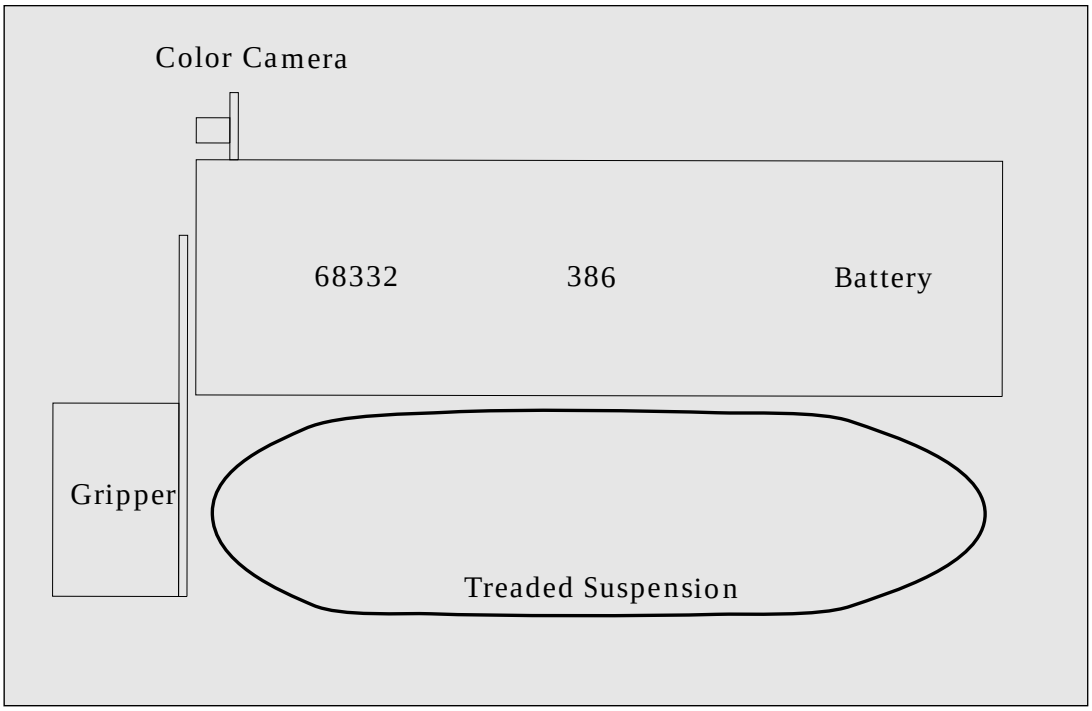

Figure 2.
Hardware Design: Each Robot Weighs about 10 Pounds, Stands 10 Inches Tall, and Measures 13 Inches Wide by 17 Inches Long.

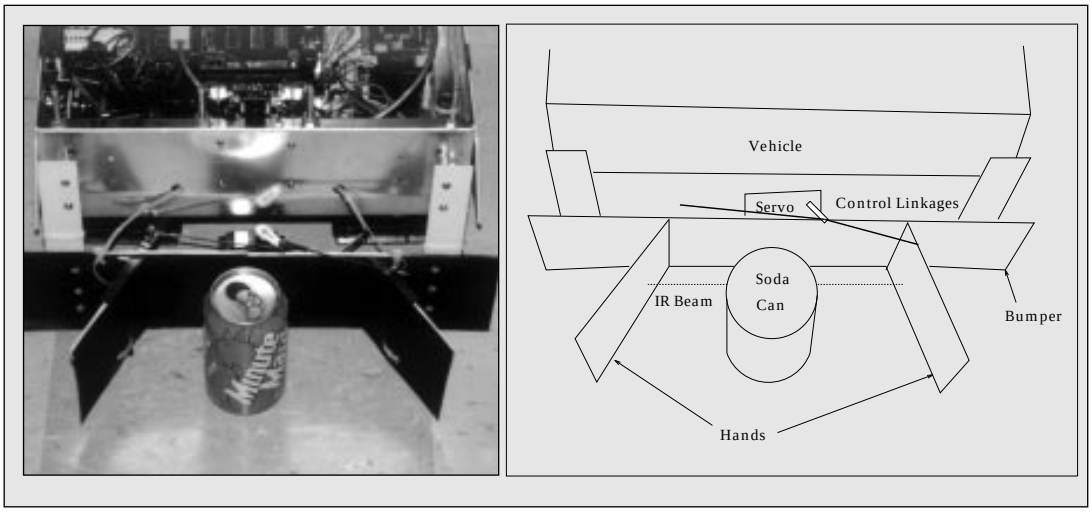

Figure 3. Closeup of Trash Manipulator.

out of the gripper, reestablishing the infrared beam. In this case, the robot infers that the object is an obstacle; otherwise, it is assumed to be trash. More details of the strategy are explained in Software Architecture.

Vision

Color vision is utilized to take full advantage of color cues for discriminating between various object classes important to the robots: trash, wastebaskets, and robots. The robots use miniature color cameras with wide-angle (73:5–degree field of view [FOV]) lenses and standard NTSC analog output. A low-cost frame grabber captures video images and makes them available in memory to software running on the PC.

The video boards support a chrominance-luminance model of color rather than the more familiar red-green-blue (RGB) standard.

Figure 4. A Robot's-Eye View,
Including Another Green Robot (left), Two Red Soda Cans (center), and a Blue Wastebasket (right).

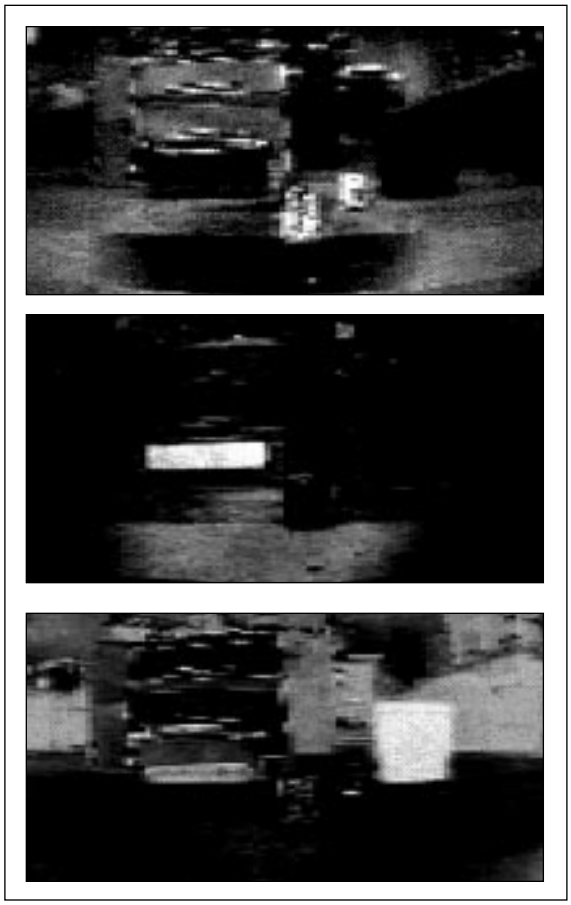

Figure 5. Superred (top), Supergreen (middle), and Superblue (bottom) Component Images of the Same Scene.

Software to convert these images to a normalized RGB format in three image planes is implemented in C++ on the PC. Additional image-processing software uses the RGB images to locate objects in the environment. Minimal, predictable memory usage, robustness, and low computational overhead were primary factors in the design of the vision software. These constraints led to the use of a simple blob-detection approach for vision processing. To simplify the problem, the team elected to take minor point penalties, according to the AAAI contest rules, for providing their own trash and wastebaskets.

This strategy enabled the team to select objects so that a separate primary color identified each important perceptual class of object: red for soda cans,[2] blue for wastebaskets, and green for other robots. These colors are easy to detect and distinguish even in a cluttered office environment. Rather than simply use red, green, and blue components directly, all of which are large for white objects such as glare spots, the software extracts supercomponents. Supercomponents for each color are computed by subtracting the values of the other two components at each pixel. Superred, for example, is computed as red 2 (blue + green). By subtracting other components, a simple hue-sensitive metric is formed. For example, a white blob has a low superred component, but a red blob has a bright superred component. Sample luminance and corresponding supercomponent

images are shown in figure 4 and figure 5.

The locations of colored objects in the environment are computed as follows: A thresholding operation on a supercomponent is followed by a blob-detecting pass. Azimuth to the object is calculated directly from the known visual field of view and relayed as a vector output or heading. Range is determined using the lowest pixel of the corresponding blob. Because all objects of interest are resting on the floor and because the camera is pointed horizontally, range can be estimated using the simple trigonometric relation

$$r = h * \arctan(theta) \ ,$$

where r is range, h is the camera height, and *theta* is the apparent angle from the center of the image to the bottom of the blob (computed in the same way as azimuth, using the known FOV). The range and heading data are converted to an object location in the robot's global coordinate system. Vision processing is completed in about one second. The accuracy of computed object positions is better than one inch when the objects are within three feet of the robot. Objects further away are usually within about a foot of the computed location. The results of vision processing are stored in global memory and are used by the behavioral processes described in Behavioral Control. The robot's position and the objects discovered visually are stored in a global coordinate system. An object's position relative to the robot can be computed even if the object is no longer visible. A reliability index associated with the location information for each object decays over a period of a minute if the object is lost from view, giving priority to objects that are currently visible. It also helps account for the approximate nature of dead reckoning and the possibility of misinterpreted visual data.

Behavioral Control

This section describes reactive behaviors developed for the competition at a conceptual level. Software that implements the approach is explained in the next section. The Office Cleanup task called for several separate steps to be accomplished in sequence: find trash, get trash, find trash can, move to the trash can, deposit trash, and so on. The approach used here is to develop separate reactive behaviors that accomplish each step. The individual behaviors are then triggered in an appropriate sequence by a simple behavioral manager (the technique is referred to as *temporal sequencing*). The behavioral paradigm used in this work and in other research at Georgia Tech's Mobile Robot Laboratory is motor schema–based control (Arkin 1989). Like other reactive systems (Brooks 1986), motor schemas offer low computational overhead; additionally, they provide for integration with temporal sequencing approaches (Arkin and MacKenzie 1994) and deliberative approaches (Arkin 1992b). Individual *schemas* are primitive behaviors that are combined to generate more complex emergent behaviors. Schemas are independent computational processes for sensing and acting that run in parallel. For example, consider the construction of the behavior for a robot that enables it to move to an item of trash while it avoids collisions with obstacles (the move-to-trash behavior). For this task, two perceptual schemas and two motor schemas are instantiated. The two perceptual schemas are (1) *detect-goal,* which uses vision to compute the location of the goal—a soda can, and (2) *detect-obstacles*, which detects and tracks obstacles in the environment using bumper switches. The two motor schemas are (1) *move-to-goal,* which generates a vector toward the goal detected by detect-goal, and (2) *avoid-static-obstacles*, which generates a vector away from any detected obstacles (magnitude varies inversely with range to the obstacles).

The vectors generated by each of the motor schemas are combined (added), then clipped to generate the overall movement vector, which is sent to the robot's actuators. Because some components of the task might be more important than others (for example, avoiding collisions is very important), the motor-schema vectors are multiplied by gain values before they are combined. The gain value for each schema allows the designer to custom configure a behavior that appropriately emphasizes the various components of the task. Individual behaviors for each of the several steps in the Office Cleanup task were developed and tested on the robots, and appropriate gain values were determined empirically. Readers are referred to Arkin (1989) for a more detailed description of the gains and parameters of specific motor schemas.

Sequenced coordination, or temporal sequencing, is the process by which the robot control system moves through a series of distinct behaviors (Arkin and MacKenzie 1994; MacKenzie and Arkin 1993). For example, when the robot is searching for trash, a different behavior is active than when it is moving toward a trash can to deliver the trash. Using

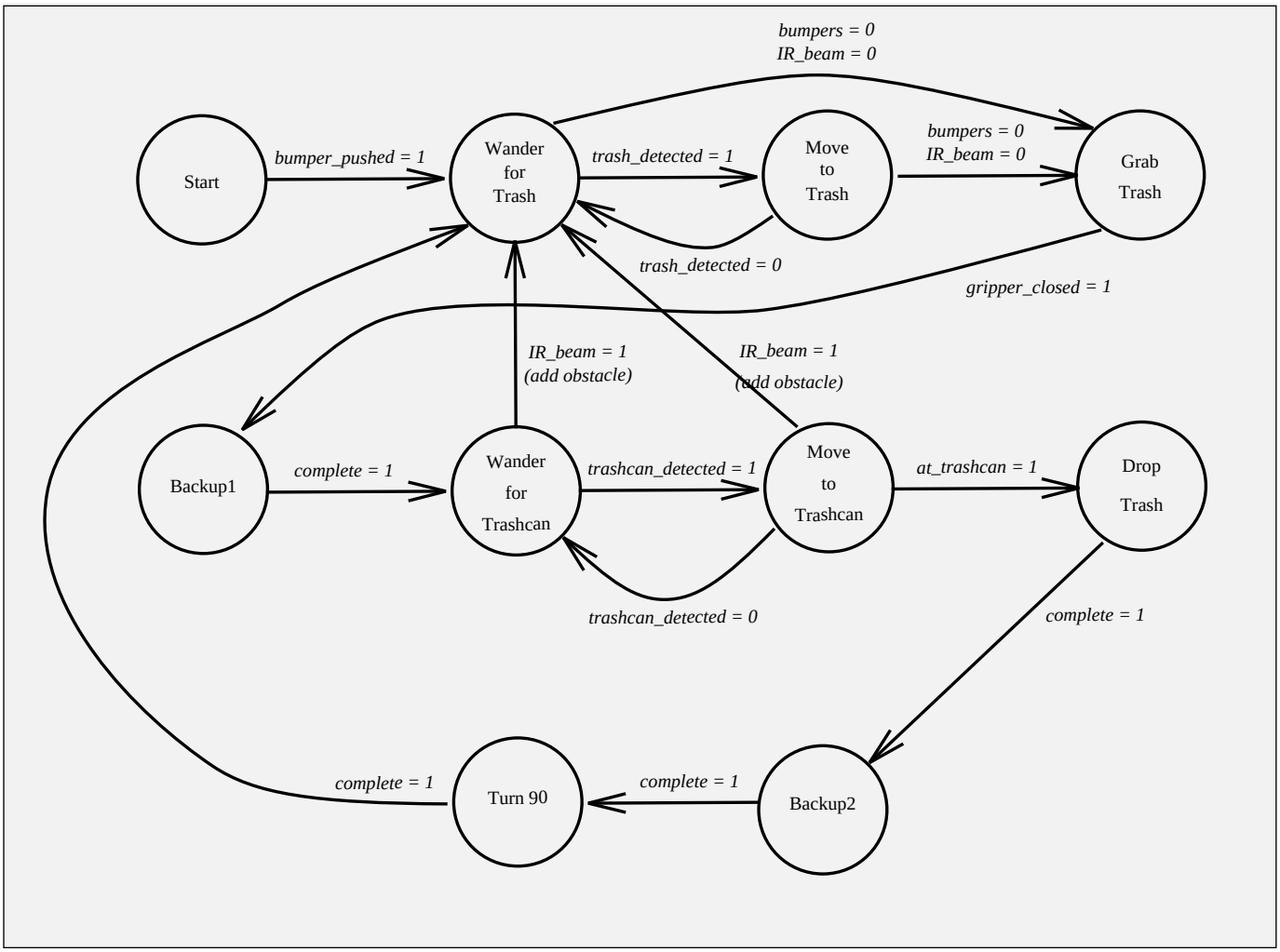

Figure 6. Robot Behavioral-State Diagram.

the temporal sequencing technique, the designer specifies each operating state (behavior) and the perceptual triggers that cause transitions between them. The resultant finite-state automaton (FSA) acts as a behavior manager, instantiating relevant behaviors based on the specified perceptual triggers. *Perceptual triggers* are specialized perceptual schemas that run in parallel with other schemas and signal the behavior manager when a state change is required. In Software Architecture, we describe how temporal sequencing is expressed in software.

Competition Strategy

Figure 6 shows the FSA constructed for the AAAI-94 robot event. States are denoted by circles and state transitions by directed edges labeled with the perceptual triggers causing the transitions.

The robot begins in the start state and waits for one of the bumpers to be pushed (a signal to the robot that the contest has begun). While in the wander-for-trash state, the robot randomly explores the arena until either it visually discerns a soda can (move-to-trash) or the infrared beam in its gripper is interrupted (grab-trash). The wander-for-trash behavior used at the competition was primarily a sit-and-spin behavior; the robot made a series of small turns so that it could visually take in the entire environment. In the move-to-trash state (outlined earlier), the robot moves toward the soda can using visual servos until the infrared gripper beam is broken (grab-trash). The grab-trash state closes the gripper, and the backup1 state moves the robot backward to see if the object it has grabbed is movable and, therefore, trash. While in the wander-for-trashcan state, if the gripper drops trash it was carrying, the robot returns to the wander-for-trash state to reacquire the object. When a trash can is visually discerned, the move-to-trashcan state directs

the robot toward it. When the robot arrives at the trash can, it drops the trash (drop-trash) and backs away (backup2). It then executes a right turn so that it will search a different area and returns to the wander-for-trash state to locate more trash.

Multiagent Cooperation

Recent multiagent robotics research at Georgia Tech has investigated tasks for robots similar to the Office Cleanup task (Balch and Arkin 1994; Arkin, Balch, and Nitz 1993; Arkin 1992a). In the work of Arkin (1992a) and Arkin, Balch, and Nitz (1993), simulated robots are tasked to collect "attractors" in the environment and return them to a home base. Attractors are items of interest scattered randomly about the environment, analogous to trash in the cleanup task. In later work by Balch and Arkin (1994), additional tasks are explored, communication between robots is added, and the behaviors are validated on Denning MRV-2 mobile robots. The previous research has established the following for multiagent foraging tasks:[3]

First, for a given number of attractors (for example, trash), more robots complete a task faster than fewer robots (Balch and Arkin 1994; Arkin 1992a).

Second, in many cases, performance is superlinear; that is, *N* robots complete a task more than *N* times as fast as a single robot (Balch and Arkin 1994).

With the hope of transferring these results to the multiagent cleanup task, we modeled the cooperative behaviors of GANYMEDE, IO, and CALLISTO on the robot behaviors used in the earlier investigations. Even though communication was shown to improve performance, quantitative results demonstrated efficient cooperation between robots without explicit communication. A key to cooperation without communication is the addition of interrobot repulsion during the initial search, or foraging step, analogous to the look-for-trash step in office cleanup. The repulsion causes robots to spread out and scan the environment efficiently. Interrobot repulsion is lowered in other phases of the task but kept at a high enough setting to prevent collisions between robots. Cooperation is implemented on IO, GANYMEDE, and CALLISTO using interrobot repulsion only; explicit communication was not used. Interrobot repulsion is implemented in two steps: First, an instantiation of blob-detector for green blobs locates robots in the camera FOV. Second, a motor schema, avoid-static-obstacle, generates a repulsive force away from the detected robot. Because quantitative data have not yet been gathered on these robots, the extent of cooperation in IO, GANYMEDE, and CALLISTO has not been established. Qualitative behavior of the robots, however, seems to confirm our earlier results.

Software Architecture

This section describes on-board software that implements the robots' behavioral control (described in the previous section). The software architecture must successfully support two major activities: (1) behavior definition and (2) schema development. *Behavior definition* consists of specifying the robot behavior in terms of states and state transitions and describing which perceptual and motor schemas should be active during each state. The software architecture allows programmers to specify the robot's behavior in a text file using the BHDL. In this way, programmers can revise behaviors simply by editing the text file. Schema development consists of writing new code to create new schemas, which then can be incorporated into a library of available schemas that programmers can use to specify more complex robotic tasks. The software architecture is written in C++ and allows programmers to write new code efficiently and reliably by strictly enforcing information hiding and supporting reusability by inheritance. Objectives of the software architecture are the following:

Flexibility allows programmers to easily specify, test, and change behavior descriptions to adapt the robots to new tasks or environments.

Expressivity allows programmers to represent a wide range of behaviors common to robots operating in complex environments.

Reliability ensures basic schemas execute appropriately and consistently, as prescribed in the BHDL file.

Performance ensures efficient execution of motor schemas and minimal resource consumption.

Ease of debugging verifies the proper execution of individual schemas and robotic tasks.

The architecture is diagrammed in figure 7. Primary elements of the system are devices, schemas, variables, and states. *Devices* interface with the robot's hardware to isolate details of hardware and configuration (for example, frame_grabber, communication_port). *Schemas* are the units of execution of high-level software. Two types of schema are implemented: (1) *perceptual schemas*, which

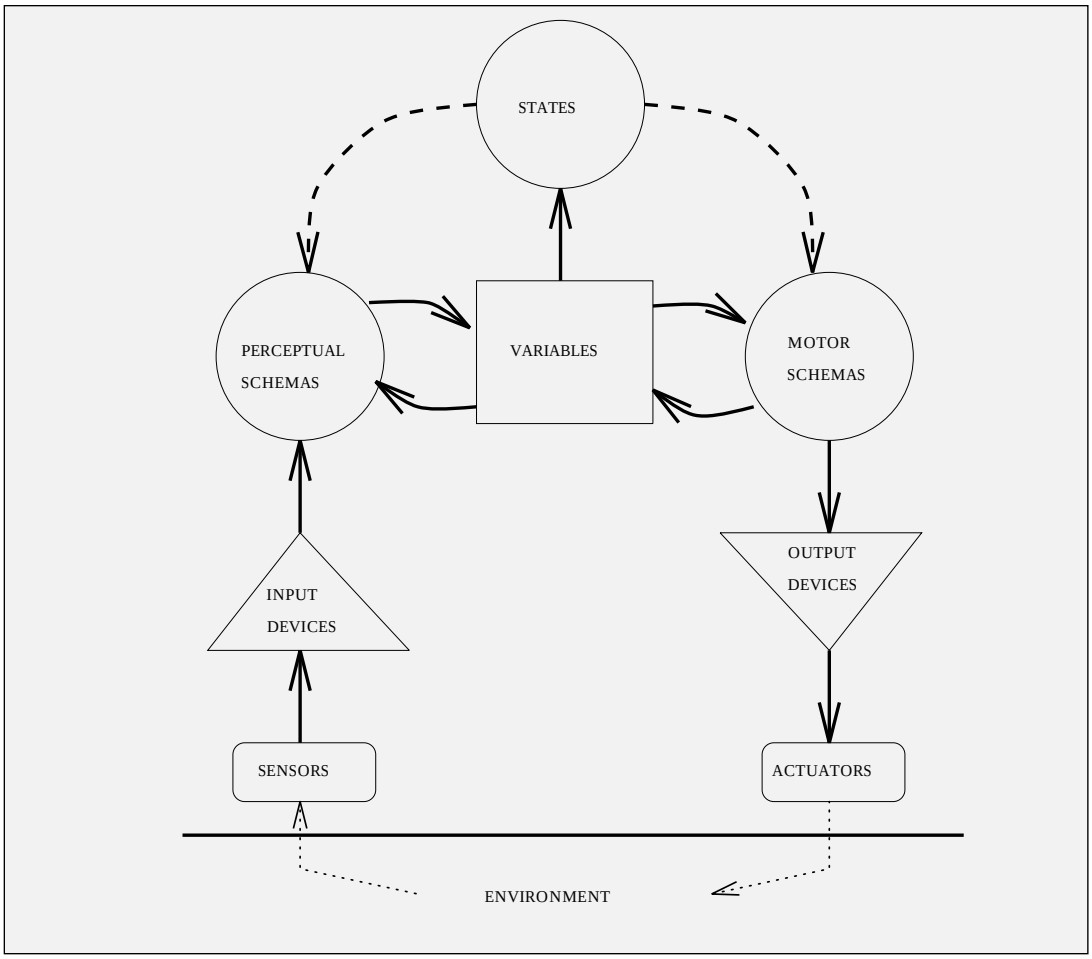

Figure 7. High-Level Software Architecture.

process information from the robot's sensors (for example, detect_obstacle), and (2) *motor schemas*, which command the robot's actuators (for example, close_gripper). *Variables* are public high-level objects accessed by the schemas to share information on robot status and sensor data (for example, green_image, bumper_switches_status). *States* encode control knowledge used to activate appropriate schemas according to prespecified conditions and current status (for example, wander_for_trash).

BHDL allows programmers to specify and configure instances of each class of component to meet the requirements of the robotic task according to the available hardware. Each component class consists of a library containing specific components that perform a particular job. Programmers specify in BHDL which of these specialized components to instantiate. Programmers can also configure each component instance individually, according to the details of the robotic task. In this way, a programmer can specify a robotic behavior on a given hardware platform quickly and easily without recompiling the source code.

This design allows great flexibility and expressiveness because the components are generic and useful across different tasks. For example, a blob_detector schema can be instantiated to process a green_image variable (that is, a green_blob_detector). Similarly, a move_to_goal schema can be instantiated to go to the location where the green blob was detected. The result is a robot that moves to green objects! However, a robot that moves away from red objects could be specified just as easily by switching the green image with a red one and using an avoid_static_obstacle motor schema instead of move_to_goal. The design is also reliable because different instances of the same component execute the same code. If a piece of code is buggy, it will fail in all its instances. Such an anomaly is easily detected so that corrective measures

can be taken. Because the software is implemented in C++, programmers can develop code that follows strict enforcement of information hiding through encapsulation. This approach allowed several programmers on our team to develop code in parallel. Additionally, reliable code is reused by inheritance, which allows programmers to develop new schemas as specializations of existing ones. The following subsections describe devices, variables, perceptual schemas, and motor schemas used at the AAAI-94 Robot Competition and Exhibition. The reader is referred to figures 8 through 10, which list fragments of a BHDL file to help illustrate the syntax and structure of BHDL.

Devices

Devices are used by schemas to communicate with hardware components in the robot. Programmers can specify what hardware components to use by specifying the appropriate devices in a BHDL file. Each component has particular attributes to customize its operation (figure 8). Devices include the following: *Frame_grabber* is used to send commands to the frame grabber card and to read digitized color images. *Communication_port* is used to send commands and receive status information from the microcontroller through the serial port. *Motor_drive* is used to send steering and translation commands to the motor drivers.

Variables

Variables are objects that hold relevant information shared by schemas. Each component in this library has particular attributes that depend on the type of information that it holds, but two attributes are common to all variables: status and certainty. The *status attribute* is a bit that indicates if the content of the variable is valid or not. The *certainty attribute* measures the reliability of the content of the variable. The certainty factor is a number between 0 (not reliable) and 1 (most reliable) (figure 9). Variables include the following: *Shaft* stores the robot's position and orientation in global coordinates. The information is estimated by dead reckoning on the left and right motor shaft encoders. *Image* stores 352 3 186 eight-bit images. *Locate* keeps track of environmental object positions in global coordinates. The environmental objects considered by the system are trash, baskets, obstacles, and other robots. *Bits* keeps track of binary events such as gripper closed and bumper switch active. *Force* stores vectors output by motor schemas.

```
;
;  Input devices.
;

input_devices:
    comm_port:
timeout: 10
end:
    frame_grabber:
    end:

;
;  Output devices.
;  output_devices:
    motor_drive:
forward_velocity: 100 ; in ticks per sec.
stright_dead_zones: 200 200 –200 –200 ; in ticks
turning_velocity: 100 ; in ticks per sec.
turn_dead_zones: 330 330 –330 –330 ; in ticks
acceleration: 10 ; int ticks per sec^2.
turn_radius: 18.5 ; in cms.
cms_constant: 3.25 ; ticks to cms.
timeout: 30 ; in secs.
end:
    end:
```

Figure 8. Declaration of Devices in a BHDL File.

Perceptual Schemas

Perceptual schemas are specialized processes that are responsible for reading information from input devices and updating appropriate variables accordingly (figure 10). Example perceptual schemas follow: *Bumper_scanner* updates the value of a bumper bit according to the status of a bumper switch. *Obstacle_detector* updates a locator variable corresponding to an obstacle according to the status of a bumper bit. *Look* reads the frame buffer and updates the image variables for blue, red, and green images. *Blob_detector* scans an image variable for blobs of specific size and intensity and uses the blob's centroid to update the information of locate variable. Green, red, and black blobs provide the location of other robots, trash, or wastebaskets, respectively. *Gripper_scanner* reads the status of the gripper and the infrared beam to detect the presence of an object in the gripper. *Time detector* keeps track of how much time has elapsed since its activation and updates a bit whenever a prespecified time has elapsed. *Forget variables* keeps track of the last update time of a variable and decreases the certainty of the variable at a prespecified rate.

```
variables:

;
;  Variables that store sensor data.
;

shaft:
    name: “shaft” position: (0,0) heading: 0 status: “valid” certainty: 1
bits:
    name: “bumpers” value: 0 status: “invalid” certainty: 0
image:
    name: “red” status: “invalid” certainty: 0.0
image:
    name: “green” status: “invalid” certainty: 0.0
image:
    name: “blue” status: “invalid” certainty: 0.0

;
;  Variables that store motor schema output.
;

force:
    name: “avoid-obstacle” value: <0,0> status: “invalid”
force:
    name: “noise” value: <0,0> status: “invalid”
force:
    name: “back” value: <0,–1> status: “valid”
```

Figure 9. Declaration of Variables in a BHDL File.

Motor Schemas

Motor schemas are specialized processes that are responsible for suggesting a direction of motion according to the information of relevant variables. Some motor schemas send commands to output devices as well (figure 10). Motor schemas include the following: *Move_gripper* opens or closes the gripper according to the value in a bit variable. *Move_robot* sends a move command to the robot according to a force variable. *Combine_forces* computes the weighted average of a list of force variables and stores the result in another force variable. The remaining motor schemas store their results in force variables: *Avoid_obstacle* computes a repulsive force away from an obstacle toward the robot. *Noise* computes a force with a random direction after every prespecified time period. *Move_to_goal* computes an attractive force from a position represented by a locator variable and the robot.

States

Temporal sequencing, described in Behavioral Control, is implemented in BHDL using state components. States are steps in an overall behavioral sequence. For each state, the behavior at that step in the sequence is specified as a list of motor and perceptual schemas to be activated. Transitions are enumerated for each state to specify conditions and corresponding next states that the robot should follow. The states and transitions between them describe a high-level finite-state automaton, which, when executed, leads to task completion. Unlike the previous libraries, this one does not contain any specialized components. All the states are instances of state_template, which contains empty lists for perceptual schemas, motor schemas, and state transitions. Programmers fill these lists in appropriately in the BHDL file (figure 11).

```
;
;  Perceptual schemas.
;

percp_schemas:
    obstacle_detector:
        name: “bumper0”
        bits_variable: “bumpers”
        bit_index: 0
        translation: <0.0,22.0>
        obstacle_variable: “obstacle0”
        end;

;
;  Motor schemas and their parameters.
;

motor_schemas:
    move_robot:
        name: “move”
        input_force: “next_move”
        shaft_variable: “shaft”
        mask_variable: “bumpers”
        max_distance_magnitude: 15.0 ; in cms.
        safety_range: 30 ; in degrees
        end:
    avoid_obstacle:
        name: “aso0”
        shaft_variable: “shaft”
        obstacle_variable: “obstacle0”
        min_sphere: 30.0
        max_sphere: 80.0
        max_repulsion: 100.0
        result: “aso0”
        end:
```

Figure 10. Specification of Input and Output to Perceptual and Motor Schemas in a BHDL File.

Results and Lessons Learned

The robots competed in three trials and the final competition at AAAI-94. Each trial provided new information that the team used to refine the robots' software between runs. The most significant problem concerned the use of vision for wastebasket detection. The office environment at the competition site consisted of tables with floor-length fabric skirts attached. These were not just any skirts; they were blue skirts! Clearly, using blue table skirts made it impractical for the robots to distinguish wastebaskets by blue color. The competition wastebaskets were distinctly black; so, the blob detector was modified to reach threshold on overall intensity less than a given value. This approach worked reasonably well, except that some items in the environment were incorrectly classified as wastebaskets, including distant dark areas in the arena ceiling, robot treads and other dark areas under a robot chassis, and shadowy areas in the folds of the skirts.

The team focused on correcting these misidentifications: Ceiling areas were not a problem because they were rejected easily by their height in the image. The areas under the other robots were eliminated by noting their relation to the green panels on the side of each robot. The shadowy areas in the skirts were handled in part by eliminating them on the basis of their geometry, but they ultimately were the largest limitation of the vision

```
;
;  The wander-for-trash state
;

state:
    name: “wander_for_trash”
    perceptual_schemas: ; active perceptual schemas.
        “trash_detector”
        “trash_finder”
        “robot_detector”
        “gripper_scanner”
        “bumper_scanner”
        end:
    motor_schemas: ; active motor schemas.
        “aso0”
        “move”
        end:
    condition: ; conditions for state change.
        bits_variable: “gripper”
        bit_index: 0
        bit_value: 1
        bit_update: “same”
        new_state: “backup1”
        reset_variables: “trash”
    end:
end:
```

Figure 11. Definition of States.

system. During the competition, the robots often deposited cans under the tables instead of next to the wastebaskets. If there had been more time, it would certainly have been worthwhile to fully investigate more alternative vision strategies. Moving beyond a basic blob-detection approach to region segmentation would probably provide more effective object detection and discrimination.

Another challenge at the competition concerned a change in the strategy for robots to find trash. Originally, robots were programmed to move in a random direction while they looked for trash or wastebaskets (refer to the wandering states in figure 6). Because there was so much trash provided in the arena, it seemed that a sit-and-spin approach would be more efficient. The robots' behavior was modified by introducing two additional states. The robot alternated between a state where it rotated for a fixed period of time and a state where it moved in a random direction. The sit-and-spin behavior continued until the robot detected the desired target. The modification was made easily because it only involved editing the BHDL file and did not require any recompilation. The strategy worked well: Robots were usually able to find a piece of trash or a wastebasket after rotating in place for a short time. In other respects, the system worked surprisingly well.

During competitive runs, the robots aggressively located the brightly colored soda cans and consistently avoided collisions with other robots. The range estimation was quite successful, with the robots almost always making their final movement toward a can within an inch of the optimal gripping location. Once this final movement is made, the camera loses sight of the bottom of the blob; so, an additional backup-and-acquire step is required if the trash does not trip the infrared gripper sensor. Once trash was acquired, it was occasionally dropped by mistake at the foot of another robot. To observers, this mistake looked like a clever hand-off maneuver when another robot completed the delivery.

Acknowledgments

This research is funded by the CIMS/AT&T Intelligent Mechatronics Laboratory at Georgia Tech. AAAI provided a generous grant for travel to the competition in Seattle. Ronald C. Arkin supervised this work and provided many helpful suggestions. Many volunteers contributed their time and expertise to this project. Erik Blasch helped assemble the robots and designed the bumper switches. Ray Hsu contributed vision software for blob finding. David Huggins and Claudia Martinez helped assemble and solder power-supply components. Knut Hybinette photographed the robots for this article. We also thank those at AAAI who organized this competition, particularly Reid Simmons, Erann Gatt, David Kortenkamp, and Rick Skalsky. Simmons also reviewed this article and offered many useful comments.

Notes

1. Georgia Tech's robots, IO, GANYMEDE, and CALLISTO are able to grab and push trash but not lift it up to drop it in a wastebasket.

2. In an empirical robot "taste test," we discovered that orange Minute-Maid cans used in the competition have a much brighter red component than red Coca-Cola cans.

3. The retrieval task examined in Balch and Arkin (1994), Arkin, Balch, and Nitz (1993), and Arkin (1992a) is slightly different from the competition task. In these publications, several robots can cooperatively carry an object, and there is only one deposit zone. This approach differs from the Office Cleanup task because attractors (trash) are light enough for individual robots to easily carry them, and several deposit zones (wastebaskets) might be available.

References

Arkin, R. C. 1992a. Cooperation without Communication: Multi-Agent Schema-Based Robot Navigation. *Journal of Robotic Systems* 9(3): 351–364.

Arkin, R. C. 1992b. Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. In *Designing Autonomous Agents*, ed. P. Maes, 105–122. Cambridge, Mass.: MIT Press.

Arkin, R. C. 1989. Motor Schema–Based Mobile Robot Navigation. *International Journal of Robotics Research* 8(4): 92–112.

Arkin, R. C., and MacKenzie, D. C. 1994. Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation. *IEEE Transactions on Robotics and Automation* 10(3): 276–286.

Arkin, R. C.; Balch, T.; and Nitz, E. 1993. Communication of Behavioral State in Multi-Agent Retrieval Tasks. In Proceedings of the 1993 IEEE Conference on Robotics and Automation, 678. Washington, D.C.: IEEE Computer Society.

Balch, T., and Arkin, R. C. 1994. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots* 1(1): 27–52.

Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1): 14.

MacKenzie, D. C., and Arkin, R. C. 1993. Formal Specification for Behavior-Based Mobile Robots. In Proceedings of the SPIE Conference on Mobile Robots VIII, 94–104. Boston: Society of Photo-Optical Instrumentation Engineers.

Tucker Balch received a B.S. from the Georgia Institute of Technology in 1984 and an M.S. from the University of California at Davis in 1988, both in computer science. He is currently pursuing a Ph.D. in autonomous robotics at Georgia Tech. From 1984 to 1988, he was a computer scientist at the Lawrence Livermore National Laboratory. His research interests include integration of deliberative planning and reactive control, communication in multirobot societies, and parallel algorithms for robot navigation.

Gary Boone received a B.S. and an M.Eng. in electrical engineering from Cornell University. He is currently pursuing a Ph.D. in computer science at the Georgia Institute of Technology. Previous research areas include genetic algorithms and reactive control in robotics. His current research focuses on strategy acquisition and transfer in complex dynamic tasks.

Thomas Collins is a senior research engineer at the Georgia Institute of Technology. His research interests include robotics, image processing, simulation, and other applications of high-performance computing. He has written several papers on parallel computer architectures for embedded applications in intelligent machines and on other topics, ranging from low-noise CCD cameras to radar signal processing. He received his Ph.D. in electrical engineering from Georgia Tech.

Harold Forbes is a graduate research assistant in the Information Technology and Telecommunications Laboratory at the Georgia Institute of Technology Research Institute, where he was named a graduate fellow in 1994. In 1977, he received a B.S. in computer science from Trinity University. He received an M.S. in computer science from Georgia Tech in 1988 and is currently a Ph.D. candidate in the College of Computing at Georgia Tech. His research focuses on operating system support for real-time, knowledge-based control systems, of which mobile robots are a particularly demanding example.

Doug MacKenzie received a B.S. in electrical engineering in 1985 and an M.S. in computer science in 1989, both from Michigan Technological University. He has been a Ph.D. student in the College of Computing at the Georgia Institute of Technology since 1990. His research interests center on AI as it applies to mobile robotics. While at Georgia Tech, he has been working as a graduate research assistant in the Mobile Robotics Laboratory.

Juan Carlos Santamaría is a Ph.D. candidate in AI at the Georgia Institute of Technology. He received a B.S. in electronic engineering from Simon Bolivar University in Venezuela in 1989 and an M.S. in computer science from Georgia Tech in 1993, where he is also pursuing an M.S. in industrial engineering. His research interests include learning and adaptation in autonomous agents, intelligent control, and theories of perception and action selection.