Io, Ganymede and Callisto - a Multiagent Robot Trash-collecting Team*

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

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

e-mail:*tucker@cc.gatech.edu*
fax: (404)-853-0957

May 10, 1997

Abstract

Georgia Tech won the Office Cleanup Event at the 1994 AAAI Mobile Robot Competition with a multi-robot cooperating team. This paper 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.

1 Introduction

Georgia Tech's team of robots, *Io, Ganymede and Callisto,* (Figure 1) placed first in the "Clean Up the Office" event at the 1994 Robot Competition 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 ten 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, pickup and deposit trash in a waste basket. Unfortunately, the computational overhead was so great that Chip was only able to do this once in ten 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 "clean up the office" event, Rhino, is described in this issue [8]. Readers interested in the overall competition are referred to Reid Simmons' companion article [10].

**AI Magazine*,16(2):39-51, 1995.

[1]Georgia Tech's robots, Io, Ganymede and Callisto are able to grab and push trash, but not to lift it up to drop it in a wastebasket.

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 to test theories regarding multiagent reliability and cooperation. Implementation of this multi-robot system is interesting from several standpoints, including the design of:

- **Low Cost Hardware:** to permit construction of several robots.
- **Reactive Behaviors:** perceptual and motor processes used by the robots to collect trash.
- **Cooperative Behaviors:** perceptual and motor processes that provide for cooperation between robots.
- **Temporal Sequencing:** to coordinate transitions between distinct operating states for each robot and achieve the desired goal state.
- **Fast Vision:** to locate soda cans, wastebaskets, and robots.
- **Behavior and Hardware Definition Language (BHDL):** for description of robot hardware, specifying behavioral states, and transitions between them.
- **Realtime Executive**: to instantiate and execute processes at run-time according to a BHDL file.

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

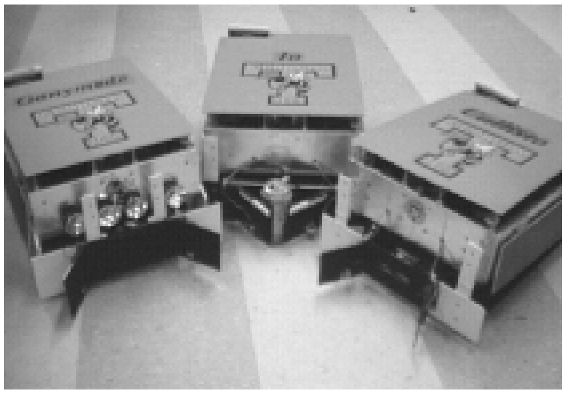


Figure 1: Ganymede, Io, and Callisto.

2 Hardware Design

The ten-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 system. 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 DC motor-powered treaded vehicle marketed for radio control model enthusiasts. Each robot base is driven by two separate motors and drive trains. Initially, off-the-shelf RC car controllers were used to drive the motors, but these were abandoned because non-linearities

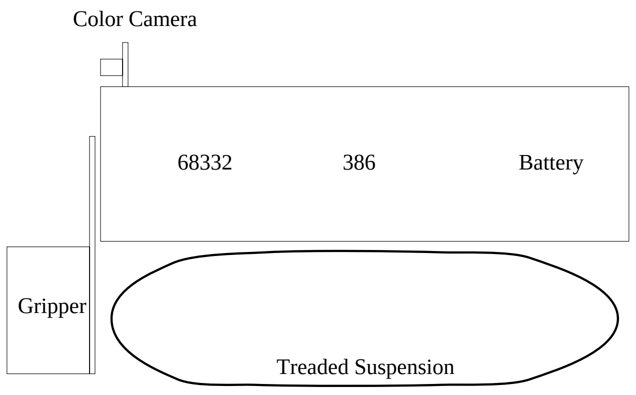


Figure 2: Hardware Design: each robot weighs approximately 10 pounds, stands 10 inches tall and measures 13 inches wide by 17 inches long.

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 trapozoidal velocity profiles. The BCCs were chosen for ease of programming (in C) and multiple digital I/O and timer channels. The microcontroller is also responsible for driving and detecting the gripper's IR 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 CPU with 4 Megabytes of RAM and a floppy disk. Small, low-power video cameras supply NTSC video directly to video capture cards. Upon 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, while 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.

2.1 Manipulator Design

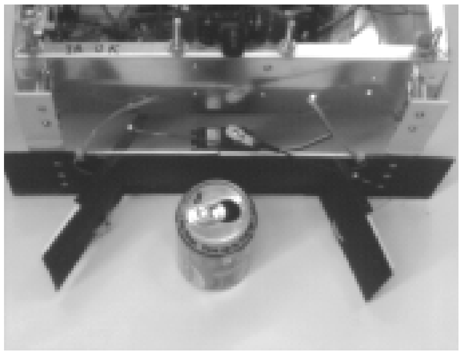
It was immediately apparent to the design team that most commercial general purpose robot arms are too heavy, too slow, and too power-hungry for small mobile robots. Task requirements were the starting point of the manipulator design:

- Capture empty soda cans, styrofoam cups, and wadded sheets of notebook paper that may be in several orientations.
- Light weight.
- Low power.
- Withstand repeated collisions with immovable objects.

Other constraints affect the manipulator design as well. Since 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 which is designed to support and protect the hands. The servo actuator is attached to the rear of the bumper with control linkages connecting it to the hands. Construction materials are steel hinges, aluminum control horns, plastic and fiber boards held together with pop rivets.

An IR beam in the gripper's hands alerts the robot of an object in range for grasping. The sensor helps the robot opportunistically grab trash it encounters. The sensor is tripped by any intervening object, including obstacles, so the robot must somehow descriminate between obstacles and trash. Whenever the IR beam is broken the robot closes the the manipulator, then backs up. If the object encountered is immovable (e.g. a chair leg) it will slip out of the gripper, re-establishing the IR 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 Section 4.



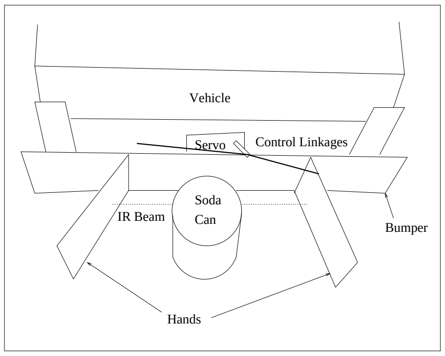


Figure 3: Close-up of trash manipulator.

2.2 Vision

Color vision is utilized to take full advantage of color cues for descriminating between various object classes important to the robots: trash, wastebaskets and robots. The robots use miniature color cameras with wide-angle (73.5 deg FOV) lenses and standard NTSC analog outputs. 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. 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 lead 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 to identifies 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 "super-components." Super-components for each color are

[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.

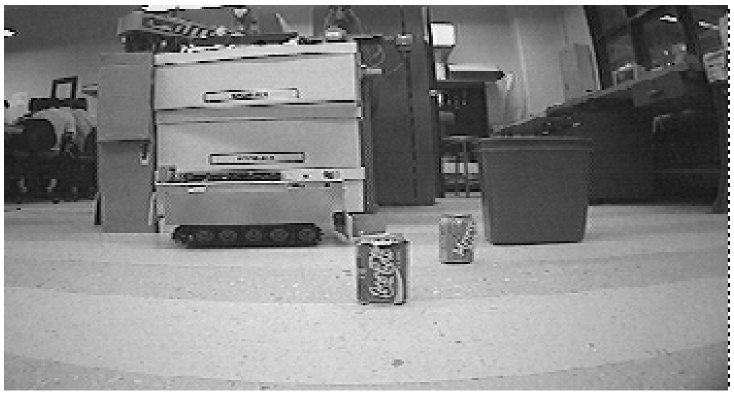


Figure 4: A robot's-eye view including another green robot (left), two red soda cans (middle) and a blue wastebasket (right).

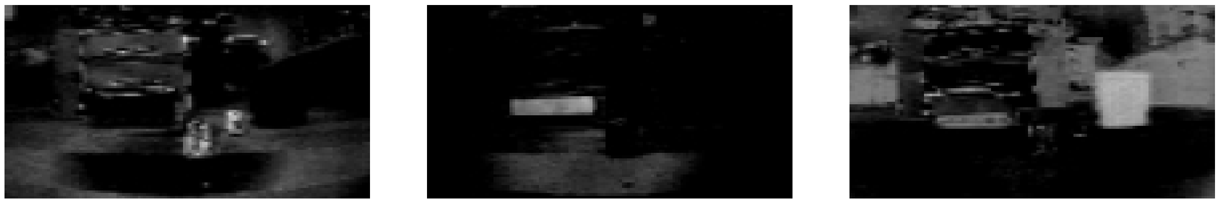


Figure 5: Super-red, super-green and super-blue component images of the same scene.

computed by subtracting the values of the other two components at each pixel. Super-red, for example, is computed as $red - (blue + green)$. By subtracting out other components, a simple hue-sensitive metric is formed. So, for example, a white blob has a low super-red component, while a red blob has a bright super-red component. Sample luminance and corresponding super-component 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 super-component 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. Since all objects of interest are resting on the floor, and since 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 field-of-view). The range and heading data is 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 behavioral processes described in Section 3. The robot's position, and the objects discovered visually are stored in a global coordinate system. An object's position relative to the robot may 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. This gives priority to objects currently visible. It also helps account for the approximate nature of dead reckoning and the possibility of misinterpreted visual data.

3 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 clean up task calls for several separate steps to be accomplished in sequence for completion: find trash, get trash, find trash can, move to the trash can, deposit trash, etc. 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 [1]. Like other reactive systems [7], motor schemas offer low computational overhead; additionally, they provide for integration with temporal sequencing [5] approaches (used here), and deliberative approaches [3] (not used here).

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. As an example, consider the construction of the behavior for a robot to move to an item of trash while avoiding collisions with obstacles (the $move-to-trash$ behavior). For this task, two perceptual schemas and two motor schemas are instantiated:

- $detect-goal$: a perceptual schema that uses vision to compute the location of the goal, a soda can.
- $detect-obstacles$ a perceptual schema that detects and tracks obstacles in the environment using bumper switches.
- $move-to-goal$: a motor schema that generates a vector towards the goal detected by $detect-goal$.
- $avoid-static-obstacles$: a motor schema that 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. Since some components of the task may be more important than others (e.g. 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 which appropriately emphasizes the various components of the task. Individual behaviors for each of the several steps in the "clean up office" task were developed and tested on the robots, and appropriate gain values were determined empirically. Readers are referred to [1] 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 [9, 5]. For example, when the robot is searching for trash, a different behavior is active than when it is moving towards a trash can to deliver the trash. Using the temporal sequencing technique, the designer specifies each operating state (behavior) and the perceptual triggers which 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. Section 4 describes how temporal sequencing is expressed in software.

3.1 Competition Strategy

Figure 6 shows pictorially the FSA constructed for the AAAI-94 competition event. States are denoted with circles and state transitions with 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 IR beam in its gripper is interrupted (*grab-trash*)

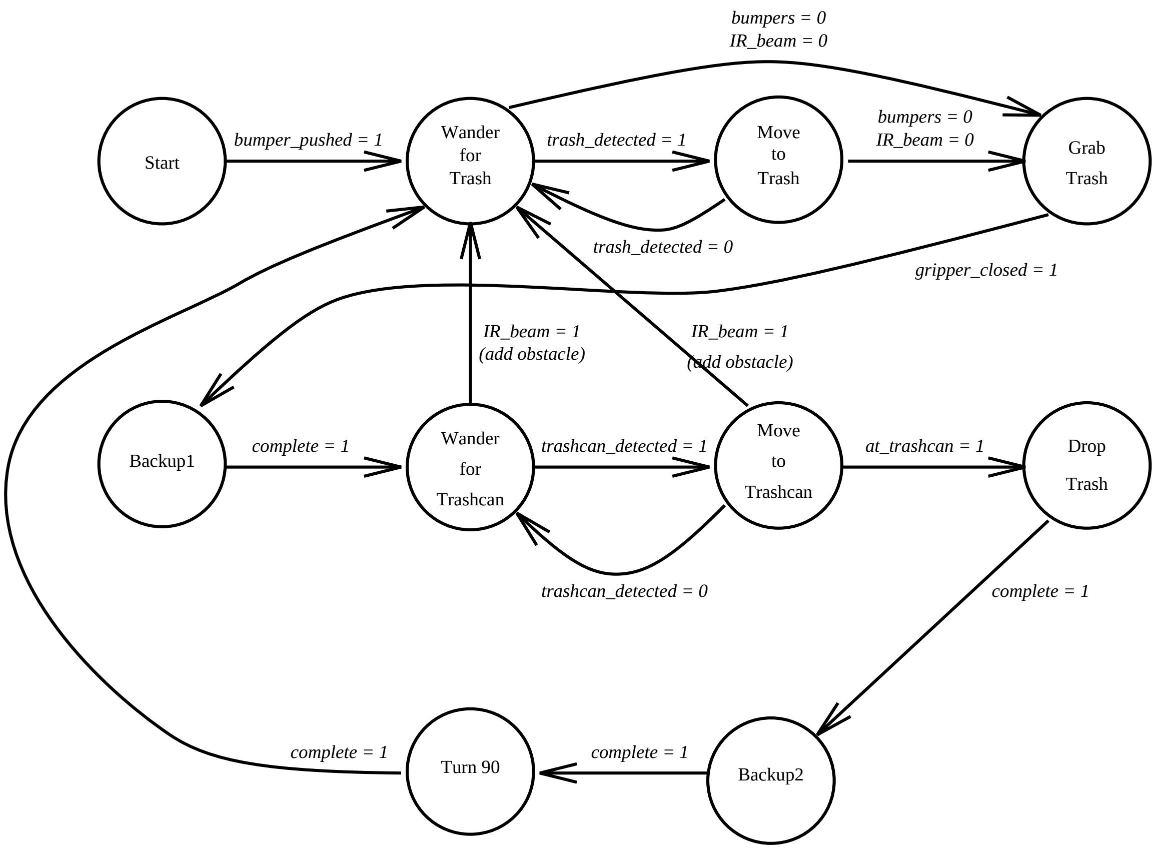


Figure 6: Robot Behavioral State Diagram

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 above), the robot moves towards the soda can using visual servoing until the IR gripper beam is broken (*grab-trash*). The *grab-trash* state closes the gripper and the *backup1* state moves the robot backwards 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 re-acquire the object. When a trash can is visually discerned, the *move-to-trashcan* state servos the robot towards 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.

3.2 Multiagent Cooperation

Recent multiagent robotics research at Georgia Tech has investigated tasks for robots similar to the "clean up the office" task [2, 4, 6]. In the work of Arkin [2] and Arkin, Balch and Nitz [4], 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 [6] additional tasks are explored, communication between robots is added, and the behaviors are validated on Denning MRV-2 mobile robots. The previous research[3] has established that for multiagent

[3]The retrieval task examined in [2, 4, 6] is slightly different from the competition task. In [2, 4, 6] several robots may cooperatively carry an object, and there is only one deposit zone. This differs from the "clean up the office" task because attractors

foraging tasks:

- For a given number of attractors (e.g. trash) more robots complete a task faster than fewer robots [2, 6].
- In many cases, performance is superlinear, i.e. N robots complete a task more than N times as fast as a single robot [6].

In hopes of transferring these results to the multiagent cleanup task, we modelled 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 inter-robot 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. Inter-robot 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 inter-robot repulsion only, explicit communication was not used. Inter-robot repulsion is implemented in two steps. First, an instantiation of $blob-detector$ for green blobs, locates robots in the camera field of view. Second, a motor schema, $avoid-static-obstacle$ generates a repulsive force away from the detected robot. Since 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.

4 Software Architecture

This section describes on-board software which implements the robots' behavioral control (described in the previous section). There are two major activities that the software architecture must successfully support to be useful. These are *behavior definition* and *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 Behavior and Hardware Description Language (BDHL). In this way, programmers may 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 may 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 resuability by inheritance.

Objectives of the software architecture are the following:

- **Flexibility:** to allow programmers to easily specify, test, and change behavior descriptions to adapt the robots to new tasks or environments.
- **Expressivity:** to allow programmers to represent a wide range of behaviors common to robots operating in complex environments.
- **Reliability:** to ensure basic schemas execute appropriately and consistently as prescribed in the BHDL file.
- **Performance:** efficient execution of motor schemas and minimal resource consumption.
- **Easy to debug:** to verify the proper execution of individual schemas and robotic task.

(trash) are light enough for individual robots to easily carry them, and there may be several deposit zones (wastebaskets) available.

The architecture is diagramed 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 (e.g., *frame_grabber, communication_port*). *Schemas* are the units of execution of high-level software. Two types of schemas are implemented: perceptual schemas, which process information from the robot's sensors (e.g., *detect_obstacle*); and motor schemas, which command the robot's actuators (e.g., *close_gripper*). *Variables* are public high-level objects accessed by the schemas to share information on robot status and sensor data (e.g., *green_image, bumper_switches_status*). *States* encode control knowledge used to activate appropriate schemas according to prespecified conditions and current status (e.g., *wander_for_trash*, see Section 4).

The Behavior and Hardware Definition Language allows programmers to specify and configure instances of each class of components 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 may also configure each component instance individually according to the details of the robotic task. In this way, a programmer may specify a robotic behavior on a given hardware platform quickly and easily without recompiling the source code.

This design allows great flexibility and expressiveness since the components are generic and useful across different tasks. For example, a *blob_detector* schema may be instantiated to process a *green_image* variable (i.e., a green_blob_detector). Similarly, an *move_to_goal* schema may be instantiated to go to the location where the green blob was detected. The result is a robot that moves to green objects! But a robot that moves away from red objects could be specified just as easily by switching the green image by a red one and by using an *avoid_static_obstacle* motor schema instead of *move_to_goal*. The design is also reliable since different instances of the same component execute the same code. If a piece of code is buggy, it will fail in all of its instances. Such an anomaly is easily detected so that corrective measures can be taken. Since the software is implemented in C++, programmers can develop code following strict enforcement of information-hiding through encapsulation. This 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.

Following is a brief description of devices, variables, perceptual schemas, and motor schemas used at the AAAI-94 Robot Competition. 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**

  These components provide the interface 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 (see Figure 11 for an example).

  - Frame_grabber: used to send commands to the frame grabber card and to read digitalized color images.
  - Communication_port: used to send commands and receive status information from the micro-controller through the serial port.
  - Motor_drive: 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 there are two attributes that 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 on the content of the variable. The certainty factor is a number between 0 (not reliable) to 1 (most reliable). See Figure 8 for an example.

  - 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 352x186 8-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, bumper switch active, etc.
  - Force: stores vectors output by motor schemas.

- **Perceptual Schemas**

  Perceptual schemas are specialized processes that are responsible for reading information from input devices and updating appropriate variables accordingly (see Figure 9 for an example).

  - 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 a 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 timed has elapsed.
  - Forget variables: keeps track of the last update time of a variable and decreases the certainty of that variable at a prespecified rate.

- **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. (see Figure 9 for an example).

  - Move_robot: this schema sends a move command to the robot according to a force variable.
  - Combine_forces: this schema computes the weighted average of a list of force variables and stores the result in another force variable.
  - Avoid_obstacle: this schema computes a repulsive force from an obstacle represented by a locator variable and the robot. The result is stored in a force variable.
  - Noise: this schema computes a force with a random direction after every prespecified time period. The result is stored in a force variable.
  - Move_to_goal: this schema computes an attractive force from a position represented by a locator variable and the robot. The result is stored in a force variable.
  - Move_gripper: this schema opens or closes the gripper according to the value in a bit variable.

- **States**

  Temporal sequencing, described in Section 3, 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

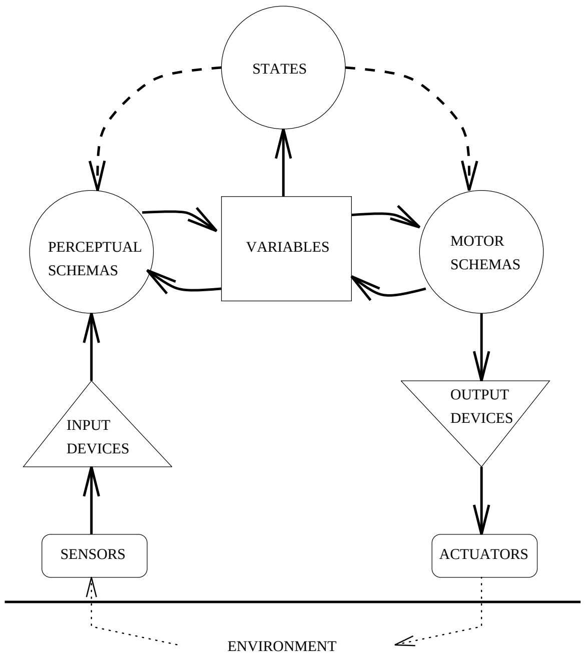


Figure 7: High-level software architecture

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 (FSA) 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 a *state_template* that contains a empty lists for perceptual schemas, motor schemas, and state transitions. Programmers fill these lists in appropriately in the BHDL file (see Figure 10 for an example).

5 Results and Lessons Learned

The robots competed in three trials and the final competition at AAAI-94. Each trial provided new information the team used to refine the robots' software between runs. The most significant problem concerned use of vision for wastebasket dectection.

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, this 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 threshold on overall intensity less than a given value. This 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.
- Shadowy areas in the folds of the skirts.

The team focused on correcting these mis-identifications: Ceiling areas were not a problem, since they were easily rejected 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

```
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 outputs.
    ;

    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 8: Declaration of variables in a BHDL file.

partially handled by eliminating them on the basis of their geometry, but they ultimately were the largest limitation of the vision system. During the competition the robots often deposited cans under the tables, instead of next to 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 descrimination.

Another challenge at the competition concerned a change the strategy for robots to find trash. Originally, robots were programmed to move in a random direction while looking for trash or wastebaskets (refer to the "wandering" states in Figure 6.) Since 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 alternates between a state where it rotates for a fixed period of time, and a state where it moves in a random direction. The "sit-and-spin" behavior continues until the robot detected the desired target. The modification was made easily, since 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 gripper IR sensor. Once trash was acquired, it was occasionally mistakenly dropped at the foot of another robot. To observers, this looked like a clever handoff maneuver when another robot completed the delivery.

```
;
; 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 9: Specification of inputs and outputs to perceptual and motor schemas in a BHDL file.

```
;
; 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 10: Definition of states.

```
;
; 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 11: Declaration of devices in a BHDL file.

6 Acknowledgments

This research is funded by the CIMS/AT&T Intelligent Mechatronics Laboratory at Georgia Tech. The AAAI provided a generous grant for travel to the competition in Seattle. Dr. 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.

We also thank those at AAAI who organized this competition, particularly: Reid Simmons, Erann Gatt, David Kortenkamp and Rick Skalsky. Reid Simmons also reviewed this paper and offered many useful comments.

References

[1] R.C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

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

[3] R.C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In P. Maes, editor, *Designing Autonomous Agents*, pages 105–122. Bradford-MIT Press, 1992.

[4] R.C. Arkin, T. Balch, and E. Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *Proceedings 1993 IEEE Conference on Robotics and Automation*, page 678, Atlanta, GA, 1993.

[5] R.C. Arkin and D.C. MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 10(3):276–286, 1994.

[6] T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1), 1994.

[7] R. Brooks. A robust layered control system for a mobile robot. *IEEE Jour. of Robotics and Auto.*, RA-2(1):14, 1986.

[8] J. Buhmann, W. Burgard, D. Cremers, D. Fox, Hofman T., F. Schneider, J. Strikos, and S. Thrun. The mobile robot rhino. *AI Magazine*, 15(2), 1994.

[9] D.C. MacKenzie and R.C. Arkin. Formal specification for behavior-based mobile robots. In *SPIE Conference on Mobile Robots VIII*, pages 94–104. SPIE, September 1993. Boston, MA.

[10] R. Simmons. The 1994 aaai mobile robot competition. *AI Magazine*, 15(2), 1994.