AI Magazine Volume 22 Number 1 (2001) (© AAAI)

REAPER: A Reflexive Architecture for Perceptive Agents

Bruce A. Maxwell, Lisa A. Meeden, Nii Saka Addo, Paul Dickson, Nathaniel Fairfield, Nikolas Johnson, Edward G. Jones, Suor Kim, Pukar Malla, Matthew Murphy, Brandon Rutter, and Eli Silk

■ This article describes the winning entries in the 2000 American Association for Artificial Intelligence Mobile Robot Competition. The robots, developed by Swarthmore College, all used a modular hybrid architecture designed to enable reflexive responses to perceptual input. Within this architecture, the robots integrated visual sensing, speech synthesis and recognition, the display of an animated face, navigation, and interrobot communication. In the Hors d'Oeuvres, Anyone? event, a team of robots entertained the crowd while they interactively served cookies; and in the Urban–Search-and-Rescue event, a single robot autonomously explored a section of the test area, identified interesting features, built an annotated map, and exited the test area within the allotted time.

In 2000, Swarthmore College entered robots in two events at the American Association for Artificial Intelligence (AAAI) Mobile Robot Competition. As any AI researcher knows, developing one robot for a real-world task such as serving hors d'oeuvres to a crowded room can be a significant undertaking. Our plan was to have a team of three agents participate in the hors d'oeuvres–serving contest and have one mobile robot attempt the Urban–Search-and-Rescue (USAR) event.

Even with 10 undergraduate students working on the project for 8 weeks, developing robots that could accomplish both events at the desired level of performance was difficult. What made it possible, let alone successful, was that each of the agents used the same overall software architecture for integrating navigation and control with perceptual processing. Furthermore, this architecture was largely platform independent and modular, permitting different agents, including nonmobile agents, to use different capabilities with few changes to the overall system.

Using the same architecture for each agent meant we could distribute our efforts and focus on common capabilities such as visual–information-processing modules and facial animation modules that could be used on several platforms. Thus, we were able to give each agent a wide range of abilities and integrate them effectively.

The unique aspects of our hors d'oeuvres entry included the integration of multiple sensors and modes of interaction in a single agent; a powerful, general purpose, real-time color vision module; fast, creative, entertaining, and robust human-agent interactions; facial animation, including tracking faces with the eyes in synchronization with spoken text; shirt-color detection and identification; fast, safe navigation in a crowded space using a reactive navigation algorithm; and communication and interaction between the agents.

The same architecture also managed our USAR entry. The only difference between MARIO the busboy robot and MARIO the rescue robot were the controlling modules. Otherwise, the vision, speech, and navigation modules were identical. The strengths of our USAR entry were completely autonomous function; a robust, reactive wander mode and a get-out mode using sonar and infrared sensors, providing a map built by the robot with connected annotated images; and the vision module, which could identify motion and skin color.

The theme for our 2000 Hors d'Oeuvres, Anyone? entry was based on our previous year's entry. In 1999, Swarthmore's waiter robot, ALFRED, won the Hors d'Oeuvres, Anyone? event. This year, ALFRED, graduated to Italian restaurant owner, changed his name to ALFREDO and went back to the competition with his sons SANTINO and MARIO. ALFREDO was the

Figure 1. Our Hors d'Oeuvres, Anyone? Entry.

Top: MARIO in his search-and-rescue uniform. Bottom: ALFREDO. Right: SANTINO ready to serve hors d'oeuvres.

maitre d', SANTINO the waiter, and MARIO the busboy (figure 1).

This year ALFREDO was not a mobile robot but a computer with a large monitor placed at the waiter's refill station. It had speakers and a video camera and would respond to different kinds of visual input. The monitor displayed a talking face, whose lips moved in synchronization with the speech. It had three special capabilities: (1) it could tell when you held your palm in front of the camera and would give you a palm reading, (2) it would comment on the color of your shirt (based on analysis of the video image), and (3) it would comment if you stayed in front of the camera too long. Otherwise, ALFREDO would talk about various things, responding to its visual input.

SANTINO, the waiter, was a Nomad SUPER SCOUT II, a medium-size mobile robot with an onboard computer. SANTINO was also outfitted with two cameras, a microphone, speakers, a 6-LCD display, and a mechanical arm that could raise a tray up and down. SANTINO used the two cameras to look for people, look for brightly colored badges, and check when the tray was empty. It would come up to a person, ask if he/she wanted an hors d'oeuvre, and then lift the tray if the person said yes. When its tray was empty, it would make its way back to the refill station. When SANTINO was happy, a face on the LCD screen would smile. When it was grumpy or angry, it would frown.

MARIO, the busboy, was a Real World Interfaces (RWI) MAGELLAN PRO, a short mobile robot with a camera and speakers. Its job was to provide entertainment by running around in the crowd. During the competition, it also had a plate of cookies on its back. In addition, it would shuttle back and forth between SANTINO and ALFREDO, attempting to strike up conversations with them. The two mobile robots could identify one another by a red, white, and green flag that each carried (one with the red side up, one with the red side down).

This year, Swarthmore not only competed in the Hors d'Oeuvres, Anyone? event but also in the Urban–Search-and-Rescue (USAR) event on a standard course prepared by the National Institute of Standards and Technology [NIST]. The robot MARIO explored one section of the course autonomously, built a map, and connected annotated 360° panoramic images of the scene to map locations. The annotations identified image areas of interest by highlighting motion and skin color. MARIO then made its way out of the course within the allotted time (25 minutes).

It's worth taking a look at what was under the hood, so the rest of this article examines

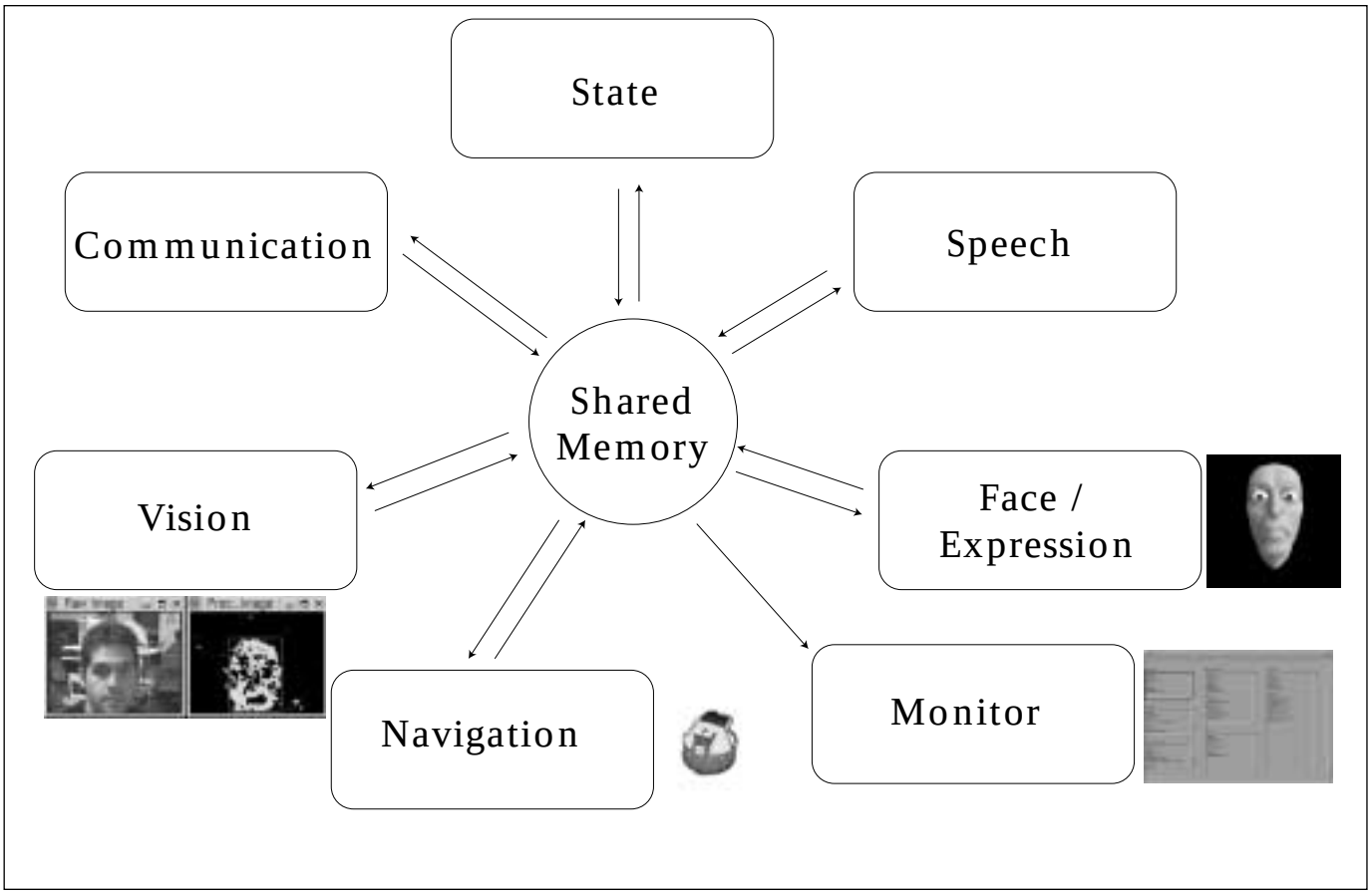

Figure 2. Logical Diagram of the REAPER Architecture.

the overall architecture and highlights the most important pieces.

REAPER: An Intelligent Agent Architecture

The system architecture, hereafter referred to as REAPER (reflexive architecture for perceptual robotics), is based on a set of modules. The purpose of each module is to handle one of the following tasks: sensing, reflexes, control, communication, and debugging (figure 2). The fundamental concept behind REAPER is that the central control module—whether it is a state machine or other mechanism—does not want a flood of sensory data, nor does it want to have to make low-level decisions, such as how fast to turn each wheel 10 times a second. At the same time, it needs real-time updates of symbolic information, indicating what the world around it is doing. The sensor and reflex modules gather and filter information, handling all the preprocessing and intermediate actions between high-level commands or goals. This approach is similar to the way our brain seems to deal with a request to pick up an object. Although we consciously think about picking up the object, our reflexes deal with actually moving our hand to the proper location and grasping it. Only then does our conscious mind take back control to decide what to do next. This approach has some biological justification (Bizzi et al. 1995). Each module takes input from, and writes its output to, the shared memory. The State module is the central controlling unit.

The two sensing modules handle all vision- and speech-based interaction. Their main task is to act as filters between the sensory data and the symbolic information required by the rest of the system. The reflex modules—navigation and face—handle the motion and appearance of the robot. The navigation module also incorporates sensing (sonar and infrared sensors), but its primary task is to control the motion of the robot, not to filter and interpret the sensory information for other modules. Central control of the robot is handled through a state module, and communication between robots is handled through its own module. Finally, the architecture has two modules for debugging purposes.

One—the monitor—shows text fields that represent all the information available to the system. The other—the visual monitor—is designed to show graphically the information being provided by the vision module.

The modules on a robot communicate through a shared memory structure, which provides a computationally efficient means of sharing information. They use a common framework for communicating and programming, including a handshaking protocol to ensure that information and commands are passed and read correctly. Communication between robots occurs through sockets between the agents' communication modules over a—possibly wireless—ethernet system.

The modules on a robot communicate through a shared memory structure, which provides a computationally efficient means of sharing information.

Central control of the robot is handled by a controller module, or *state module.* This module is started first, and it starts up all the other modules it needs, each of which is its own program. In our implementation, the state module then initiates a state-machine process that specifies how the robot will interact with the world, the sensing and interaction modalities it will use, and the navigational goals it needs to achieve. To specify what the other modules should do, it uses our handshaking protocol to send information and commands. The other modules, in turn, maintain blocks of output information that the state machine uses to determine what to do next and when certain actions are complete.

The state module is the most difficult to develop because of timing and synchronization issues. In particular, our state machine has to avoid switching between states too quickly. Because it does not include any of the low-level sensing or interaction, it iterates extremely quickly and can move between states before other modules have any chance to react to the previous state. Thus, it has to watch flags from the other modules to determine when actions are complete before moving on or making a decision. The strength of this approach is that the state machine can sit back and sample high-level information asynchronously, reacting to changes in the world smoothly and quickly.

The REAPER design methodology follows a number of the lessons learned about robot architectures over the past two decades. In particular, it follows the recommendations of Bryson (2000) that autonomous agent architectures need a modular structure, a means to control action and perception sequences for complex tasks, and a mechanism for reacting quickly to changes in the environment.

REAPER is related to behavior-based systems, such as those proposed by Brooks (1986), in that it distributes low-level interpretation and control of sensors and actuators to modules. These modules run in parallel, and those connected to actuators are designed to react to changes in the environment on their own. Within the modules, a subsumption approach and a weighted mixing of a set of behaviors are useful ways of generating appropriate low-level actions.

REAPER is not a pure behavior-based system, however, because it is designed to facilitate the mixing of symbolic and subsymbolic, or reactive strategies. In particular, the sensing and acting modules receive symbolic commands from a controller module that tells them their current set of low-level goals. Thus, REAPER also falls into the category of hierarchical architectures, which facilitate the development of action and perception sequences for complex tasks. Although our implementation of REAPER has so far been a two-level architecture, our fundamental approach can easily support a three-level architecture with a knowledge representation and reasoning system as the top level. The overall controller, in this case, could be either the middle level, as in Gat's (1991) ATLANTIS architecture or the top level as in Bonasso et al.'s (1997) 3T architecture.

Finally, REAPER's modular design facilitates cross-platform and multiplatform development and simplifies debugging and testing of the individual modules, which is particularly true with sensing modules, which we extensively tested on their own before putting them on the robot platform. As noted by Bryson (2000), modularity in design has become a staple of robot architectures for just these reasons.

At the most fundamental level, however, the REAPER approach evolved because it enables the agent to gather—or reap—information at a tremendous rate from a large number of sensors and present this information in a useful, symbolic form to the controlling module. Because our concern was real-time interaction between people and a robot integrating vision, speech, sonar, infrared, and bump sensors, it was essential that we offload the responsibility for sensor interpretation to individual modules. These individual modules could then plan and schedule processing according to their current goals, providing the most information possible given computational constraints.

Overall Module Structure

The modules other than the controller all contain the same basic program structure. After startup and initialization, each enters an event loop—initially in an idle state. Each time through the event loop, the module tests whether the controller has issued a new command. If so, the transition to executing that

command take places. Otherwise, the module processes the current command. When it completes the current command, the module transitions itself back to an idle state and indicates to the controller using a flag that it is in an idle state. In some cases, such as sensing commands, the module continues to process and update sensory information until told to do something else.

The goal of all the modules is to make the event loop as fast as possible. In the navigation module, our goal was to maintain a control loop of at least 10 hertz; in the vision module, our goal was to maintain 30 hertz, or real-time visual processing.

Reflexes: Navigation

The navigation modules on the SCOUT and MAGELLAN have to be platform specific because of the differences between the two robot's low-level interfaces. From the point of view of the controller modules, however, they appear similar. Different groups developed the navigation modules, so although they both use a reactive architecture, they differ in the specifics.

Scout Navigation

The navigation requirements for the SCOUT were fairly simple for the hors d'oeuvres event. It had to move slowly and safely, be able to get to a goal location, and be able to avoid obstacles on the way there. In addition, it had to have a mode where it actually stopped for an obstacle in case it was a person to serve.

The navigation module is set up as a two-layer reactive system. The sensors available to the navigation module are the sonar and bump sensors, including five bump sensors on a low front bumper we added to SANTINO. The bottom layer contains a set of behaviors that react directly to these input. These behaviors include the following: goal achieving, obstacle avoidance, wander, free-space finding, and front bumper reaction.

Each of these behaviors returns a fuzzy priority, speed, and heading. The controller layer combines the speed and heading values based on its mode and the currently active behaviors.

The modes-commands for the navigation system include Idle, Stop now, Stop slowly, Goto Avoid, Goto Attend (stop for obstacles), Arm Up, Arm Down, Wander, Track Attend, Track Avoid, and a set of commands for resetting the odometry and controlling orientation.

The most interesting of these modes is the track modes. Their purpose is to directly connect the vision system and the navigation system without controller intervention. They can be used to follow a judge's name-tag badge or track a target in real time. Once the vision module finds a badge or target, the controller initiates the mode in both the vision and navigation modules. Once initiated, the vision module continues to track the object and update the object's position. The navigation module, in turn, reacts as quickly as possible to the visual information and tries to orient and follow the target. The two modules continue to track the target until either the target is lost, the controller ends the tracking, or an obstacle appears (in the case of Track Attend mode).

MAGELLAN Navigation

The MAGELLAN PRO, MARIO is a small, round robot with symmetrically opposed wheels that allow it to rotate on its axis. The basic sensor array consists of 3 rings of 16 bump (contact), sonar, and infrared sensors mounted around the sides of the robot. In addition, MARIO has a Sony DV30 pan-tilt camera and external speakers. The onboard computer is a PENTIUM II running LINUX 2.2.10, and it communicates with the robot's RFLEX controller over a 9600-baud serial line.

Because of the lack of a low-level software library, we developed an interface for the MAGELLAN that we called MAGE. MAGE communicates directly with the RFLEX controller of the robot. The RFLEX accepts a simple set of motor-control commands and is also responsible for transmitting the sensor data of the robot back over the serial line. We were able to extract or deduce most of the protocol for this communication from some example code that RWI provides for updating the internal memory on the RFLEX. At our request, RWI sent us code snippets containing information relevant to the infrared sensors, which allowed us to enable and read the infrared range values. During this time, we also developed and integrated a controller for the Sony pan-tilt-zoom camera on the robot, which was controlled over a separate serial line.

In general, the MAGE application programming interface (API) closely resembles the API for the Nomad SUPER SCOUT—because we have extensive experience with the SCOUTS—although it has a simplified command set and uses thousandths of meters as the units of distance and thousandths of radians as the units of rotation (as opposed to tenths of an inch and tenths of a degree on the SCOUTS).

In keeping with the Nomad API, all sensor- and motor-control data are maintained in a large state vector. For example, the statement State[STATE_SONAR_0] returns the most recent value of the forward-pointing sonar sensor. This state vector is updated continuously by a thread that handles new data passed from the robot controller. Although the RFLEX controller supports a request-based protocol, the simpler

method is to ask it to continuously stream data from the sensors as fast as it can. This approach ensures that the sensor data are as up to date as possible. To send motor commands, the API includes a method that sets the contents of an output buffer. The same thread that handles incoming data also watches this buffer and transmits its contents to the RFLEX controller. As a note, these motor data are transmitted immediately if they change and then transmitted periodically to keep the RFLEX controller alive. Serial communication with the pan-tilt-zoom mount of the camera is implemented in the same way.

The navigation module sits on top of the MAGE API and is responsible for reporting the basic sensor data and actually getting the robot from point A to point B without running into obstacles. The MAGELLAN navigation module has several different modes, but they are all based on a reactive kernel. The robot decides how much to translate and rotate based on the following four lines of code:

```
Translate = Translate – Distance to nearest
object in front
Translate = Translate + Distance to nearest
object behind
Rotate = Rotate – Distance to nearest
object to the right (assuming clockwise
rotation)
Rotate = Rotate + Distance to nearest
object to the left
```

To make the robot wander, we give Translate a forward bias. To go to a goal point, we calculate the translation and rotation biases required to push the robot toward the goal point. To track an object, the navigation module monitors the relative position of the object—stored in the vision module—and feeds this information straight into the biases. This approach proves to be robust as long as the biases do not exceed the maximum repulsion of obstacles.

To build a map used in the USAR event, the navigation module uses an evidence-grid approach (Moravec and Elfes 1985). We integrate sonar readings into a probabilistic map that can then be classified into free space and obstacles for interpretation by a person. The evidence-grid technique worked well in our test runs, but in the actual event, small objects on the floor and tight paths between obstacles caused sufficient wheel slip to significantly throw off the odometry. Thus, local areas of the map were correct, but globally, it did not reflect the test situation. This problem is common with evidence grids and suggests that for this task, we either need alternative means of localization, or we need to take a topological approach to map building.

Reflexes: Face

Robot-human interaction is the key component that distinguishes the Hors d'Oeuvres, Anyone? competition from other robot competitions. The goal of creating a fully functional intelligent agent with the capabilities of any average human is far from realized. Our robot team this year began to make strides in developing our own synthetic characters to better solve the difficult task of the competition by incorporating an animated, three-dimensional graphic model of a human head with interactive capabilities.

A growing amount of work has been dedicated to the creation of synthetic characters with interesting interactive abilities. Each year, the competitors in the robot contest find better ways to explicitly display complex interactions with humans. We considered a number of graphic models with the capability to display emotion and the flexibility to add increasingly more complex abilities. The DRAGON WING, for example, is a facial modeling and animation system that uses hierarchical b-splines for the generation of complex surfaces (Forsey and Bartels 1988). The technique provides an incredible amount of flexibility but was too complicated for our needs. Instead, we used a muscle model for facial animation and facial geometry data available on the web (Parke and Waters 1996). We ported the system to OPENGL (Neider, Davis, and Woo 1993) on LINUX.

The facial model is a simple polygon representation that uses 876 polygons. Only half the face is actually described in the input data file because symmetry is assumed between the right and left sides. Reading the data and rendering it is straightforward in OPENGL. The system permits the user to view the face data in a number of ways, including transparent, wire frame, flat shading, and smooth shading. In addition, the user can orient and rotate the face.

The model we used includes a simple muscle model to animate the face. The muscles are defined by specifying the beginning and ending points as well as a zone of influence. Each muscle can be relaxed or contracted, affecting all those vertexes within its specific zone of influence. The face module has a set of predefined expressions that consist of a set of contractions for each muscle in the facial structure. We move between expressions by interpolating the differences in the expression vectors. Our system uses a total of 18 different muscles and 6 unique expressions (figure 3).

Beyond the structure of the face, we added a couple of features to increase the interactivity of the system. First, the jaw has the ability to move to synchronize the mouth with speech genera-

tion. The jaw is able to rotate vertically by specifying jaw polygons and then rotating them about a central horizontal axis. The mouth is also able to move horizontally from puckered lips to a wide mouth through the addition of a virtual muscle that contracts the polygons of the mouth. Our speech-generation program, IBM's VIAVOICE OUTLOUD, generates a mouth data structure, containing mouth height and width, in parallel to the sound synthesis (IBM 1999). We pass this information to the face module and use it to update the mouth state in synchronization with the robot's speech.

The second capability we added was to give the face eyes, half-spheres colored appropriately with an iris and a pupil. In addition, the face module orients the eyes according to the output of the vision module, which simulates the effect of the eyes tracking people's faces or focusing on their conference badges.

We presented the faces on SANTINO and MARIO using color LCD displays at a resolution of 640 x 480 in 8-bit color. On ALFREDO, a dual-processor workstation, we presented the face on a 17-inch monitor with 8-bit color at a resolution of 800 x 600 pixels. The complete animation capabilities were only used on ALFREDO because of the more limited processing power on the mobile robots. On ALFREDO, with the full capabilities—and the vision module running simultaneously—the rendering system was able to run at approximately 9 hertz, which was at the low end of acceptable quality.

Overall, the facial animation system greatly enhances the interactive capability of the trio of intelligent agents. It gives people a central focus when interacting with the robots and helps to keep their interest throughout an interaction.

Senses: Speech

To serve people, a server must be capable of interacting with those being served. The server must signal his/her presence and offer the objects being served, the servee must be able to signal acceptance, and the server must serve. On SANTINO, we chose to make speech the main modality of communication and wanted SANTINO to be capable of asking people if they wanted an hors d'oeuvre and responding correctly to their response.

SANTINO's speech-generation and -recognition capability comes from commercially available development software. For recognition, it uses the VIAVOICE SDK for LINUX, which we also used in our 1999 robot entry (Maxwell et al. 1999a). For speech synthesis, the robots use VIAVOICE OUTLOUD, which provides the capability to adapt voices and intonation, in addition to being easy to integrate with the VIAVOICE recognition system (IBM 1999).

The major difficulty in using the VIAVOICE software is that it is designed to be used with a single person speaking clearly into a microphone in a mostly quiet room. The hors d'oeuvres competition does not meet these criteria. Instead, we expect several hundred people chatting among themselves and some people not knowing to speak directly into the microphone. Building on our experience in 1999, this year's module keeps recognition interactions brief and includes some additional filtering to remove unnecessary noise, as we describe later.

To complement the minimal recognition ability, the robots have as much personality as we can give them, including writing humorous and sometimes outrageous expositions, using different voices for each robot, and using the phrasing abilities of the OUTLOUD software. All three of the agents—SANTINO, MARIO, and ALFREDO—use the same basic software. ALFREDO and MARIO do not try to recognize speech but engage in a monologue that responds to different input. SANTINO engages in short interactions involving both recognition and synthesis.

SANTINO's Speech Module

SANTINO's speech module integrates both VIAVOICE and VIAVOICE OUTLOUD for recognition and synthesis, respectively. One of the major improvements on the speech system suggested by last year's hors d'oeuvres competition is that our robotic waiter agent can detect when the background noise exceeds a threshold, thus making speech recognition undesirable. When the environment is too noisy, the speech module shuts down its speech-recognition component and switches into a different mode that uses only speech synthesis. This noise-detection ability greatly improves speech-recognition rates because the robot attempts recognition only in reasonable environments.

The noise level–detection function calculates the average power of a 10-second sound recording from an omnidirectional microphone and compares it to a threshold value (Ifeachor and Jervis 1995). The threshold value is determined at the conference hall just prior to the competition. In determining appropriate threshold values, the function uses the peak power of a sound waveform as a guide to prevent us from specifying a threshold that would never be exceeded. The threshold value is set so that speech recognition will still occur with some background noise.

In addition to making the speech module more robust through the noise-level detector, the module uses a simple finite impulse

... the face module orients the eyes according to the output of the vision module, which simulates the effect of the eyes tracking people's faces or focusing on their conference badges.

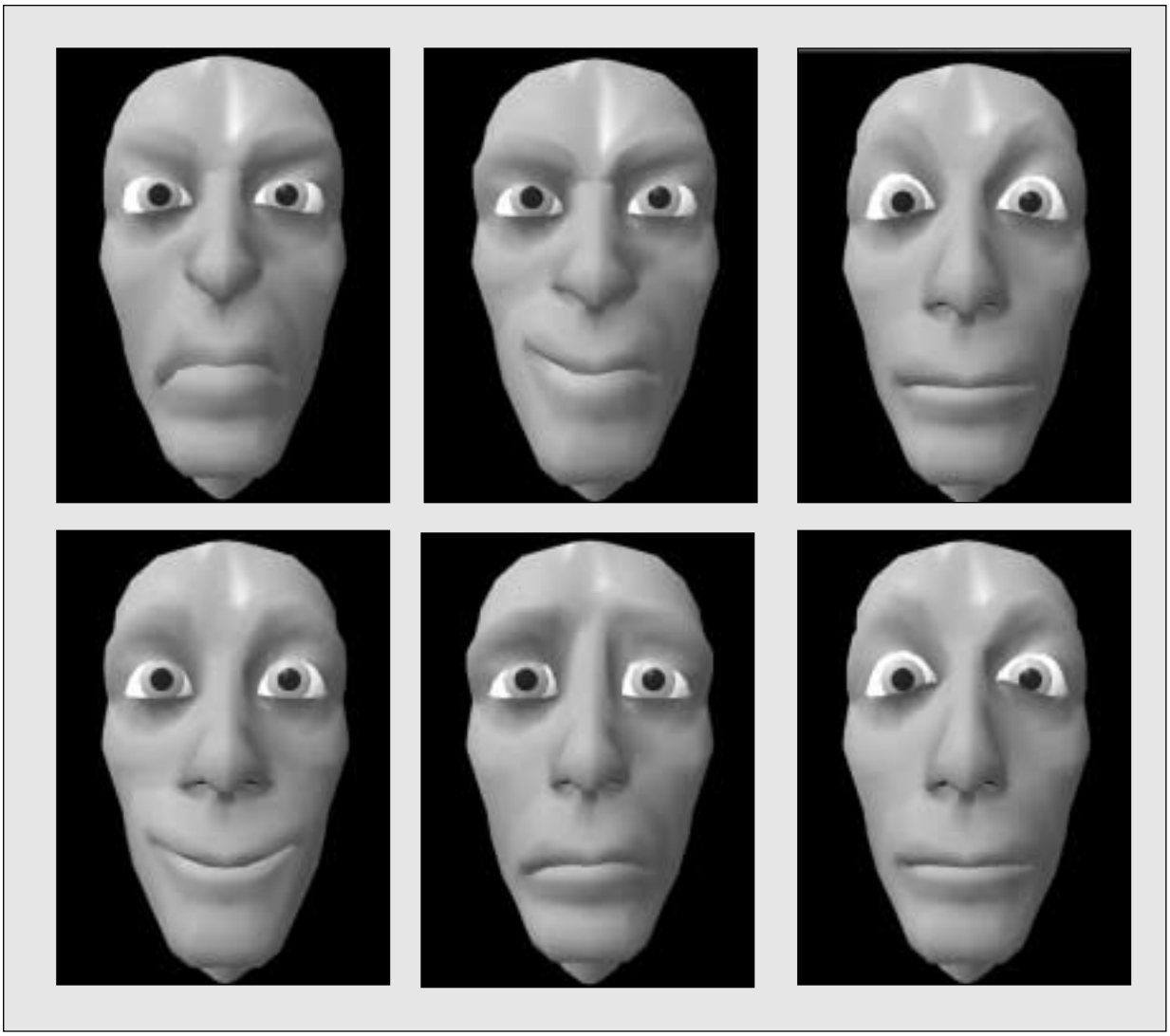

Figure 3. The Faces of SANTINO.

Top (left to right): anger, disgust, fear. *Bottom (left to right):* happy, sad, surprised.

response band-pass filter to eliminate frequencies that are beyond a specified range (~200 hertz to 2 kilohertz) (Ifeachor and Jervis 1995). Mechanical objects—such as ventilation fans in a conference hall—mainly produce low frequencies, but high frequencies occur from electrical interference in the sound card, which is integrated on the single-board computer on the robot. To ensure module independence and speed, the band-pass filter is integrated into the VIAVOICE speech-recognition audio library, thus bypassing the necessity to first record the speech utterance to a pulse-code–modulated (PCM) wave file, perform filtration, and then pass the output to the recognition engine.

The most important part of the competition for SANTINO was interacting with a person during a serving scenario. There are several classes of spoken phrases used during each serving scenario. When the state machine signals speech to begin an interaction, it says something that asks the person if they would like something to eat, usually in an interesting and occasionally rude way. When the robot finishes speaking, the recognition engine gets control of the sound device to record the response of the person. If the recognition engine registers a yes or no response, the speech module reports the response to the controller module, which then instructs speech to respond appropriately and end the interaction.

If the recognition fails, the robot will either say something about the color of the person's shirt—if vision has managed to detect shirt color—or something noncommittal. SANTINO will then ask the person again if he/she want an hors d'oeuvre and listen for a response. A

second failure causes speech to say something to just get out of the interaction, and state starts to look for someone else to serve. If the robot hears nothing at all, the speech module will comment that the person is probably a box being mistakenly served and move on.

During the competition, we found this shortened interaction to be better than the long interactions we used in the 1999 competition. People did not lose interest or become confused about what to say, which seemed to make the robot more personable and desirable to interact with.

When SANTINO is not in an interaction, it mutters, which is a running commentary about whatever the robot is doing at that moment. When the robot is in the GOTO_SERVE state and not serving anyone, it mutters about all the food that it has to give out. In the GOTO_REFILL state, it mutters and tells people to not bother it; there is no food to be had. To get this functioning properly on the actual robot, we had to make synchronous calls to both VIAVOICE programs telling them to stop controlling the audio device to deal with a slow turnaround time switching from input to output on the sound card.

The speech module acquitted itself very well at the competition. Recognition rates in the crowded hall were fairly high, at about 70 to 75 percent, which included misrecognitions of people not talking into the microphone or saying something with absolutely no resemblance to a yes-no response. Given the loudness and the large numbers of people, the robot did just a little worse than a human might have in the same circumstance. Its worst mistakes were when it appeared that a variable was not getting properly cleared, causing the robot to respond to a no response as if it were a yes response, but this mistake only occurred once or twice. We attribute the strong performance of the speech module to the fact that we could run it and debug it as a stand-alone program prior to incorporating it into the overall REAPER architecture, which, in itself, is one of the strengths of the REAPER approach.

MARIO's Speech Module

Because MARIO does not attempt speech recognition, its speech module is a simplified version of SANTINO's. The speech module serves a mainly diagnostic function, encoding information about the internal state of the robot into natural-sounding phrases, as well as a means for the robot to communicate its goals and interact with humans. The speech output is expressed as strings and then we render the speech using IBM's VIAVOICE OUTLOUD. Although the speech module does have the functions to read and speak a phrase directly from the state module, we often used a more flexible mutter mode. In the mutter mode, the speech module monitors the shared memory information fields and makes its own decisions about what to say. Once properly configured, the mutter mode picks an appropriate phrase out of a pool of possibilities every few seconds. To a practiced ear, the robot's muttering is informative about the robot's internal state, but at the same time, it reduces the risk of hearing the same boring phrase over and over.

Senses: Vision

Being able to sense the visual world gives numerous advantages to a robot, especially one involved in human interaction. Visual capability allows the robot to find and locate objects, detect motion, and identify visual object characteristics. One of our goals in both contests was to make the robots react to their world as quickly as possible. Thus, the navigation module maximized the number of times a second it executed the control loop. Likewise, our goal in the vision module was to maximize the frame rate and still provide a rich array of information.

When SANTINO is not in an interaction, it mutters, which is a running commentary about whatever the robot is doing at that moment.

The structure of the vision module is similar to the others. After initialization, the event loop checks if there is a pending command from the controller. If so, it transitions to the new state according to the command. Otherwise, it continues to execute the current command set.

The vision module includes a rich set of operators for converting images into symbolic information. The three general classes of operators are (1) object detection, (2) motion detection, and (3) object characteristic analysis. Each command to the vision module indicates a general mode and the set of operators that should be turned on. The controller can then scan the relevant output fields of the vision module for positive detections, motion, or object characteristics. Each output field includes information about where an object was detected in the image and when it was detected as determined by a time stamp. The controller can then decide what information requires a response.

The set of operators provided by the vision module include (1) person detection based on skin color and gradients; (2) motion detection across multiple frames; (3) color-blob detection, focused on conference badge detection; (4) *p*-similar pattern detection; (5) red, white, and green flag detection; (6) palm detection; (7) orange-arrow detection; (8) shirt-color analysis (dependent on detecting a person); (9) person identification (dependent on detecting

a person); (10) calculation of how much food was on the robot's tray (using the tray camera); and (11) the taking of a panoramic image (on MARIO only).

Which operators are available depends on the mode the controller selects. The modes relevant to the competition are IDLE, LOOK, TRAY, and PANO. The LOOK mode is the primary mode of operation and permits all but the last two operators to be active. The TRAY mode activates the second-camera input and analyzes how much of the tray is filled. The PANO mode works with the pan-tilt-zoom camera on MARIO to generate a 180-degree panoramic image that concatenates 8 frames together while it applies the motion- and person-detection operators.

In the LOOK mode, there is clearly no way that we can maintain a high frame rate and execute all these operators on each image. To solve this problem, the vision module includes a scheduling algorithm that only applies a few operators to each frame. This solution works because the controller usually doesn't really need to know that there is a badge in view—or whatever other object—30 times a second. Frame rate is usually a lot faster than the robot can react to things because reactions often involve physical actions or speaking. Likewise, most of the other operators do not benefit from continuous application. Because we supply a time stamp with each piece of information, the information sticks around, and the controller can decide based on the time stamp whether a piece of information is recent enough to warrant a response.

Our scheduling algorithm is based on the premise that running two operators for each frame does not reduce the frame rate. This puts an upper bound on operator complexity, although in the case of motion analysis, we get around the limitation by pipelining the process. In the standard LOOK mode, the module randomly selects two of the active operators based on a probability distribution. To create the probability distribution, each process has a weight associated with it, with processes requiring higher frame rates receiving higher weights. Most of the vision operators have small, relatively equal weights. Once selected, the module executes the two operators and updates their information. On average, each operator is executed according to the probability distribution.

The motion-detection operator is the most difficult operator to integrate within this framework because it requires multiple frames—at least three for robust processing—and requires a significant amount of processing for each frame. Our algorithm uses Sobel gradient operators to calculate edge images and then subtracts adjacent (in time) edge images to locate edges that moved. It then locates the bounding box of areas of motion that exceed a certain threshold. We have found this algorithm to be quite successful at locating people in the hors d'oeuvres event (Maxwell et al. 1999a, 1999b; Moravec and Elfes 1985).

To avoid breaking the overall structure of the vision module, we pipeline the algorithm across multiple event loops. The motion algorithm takes five event loops to calculate a result, with the first three capturing images and calculating the Sobel results. To ensure the motion algorithm is called frequently enough, we give it a high weight in the probability distribution. On average, the motion algorithm produces a result 5 to 6 times a second, and the vision module maintains a frame rate of greater than 20 hertz. When the motion operator is active, it is usually selected as one of the two scheduled operators.

A secondary mode within the LOOK mode permits tracking using one operator in addition to looking for other objects. To engage tracking, the controller specifies a single tracking operator and the regular list of other active operators. The vision-module scheduler then puts the tracking operator in one of the two execution slots and randomly selects the other operator from the active list. Thus, it is guaranteed that the vision module will look for the object being tracked every frame, providing the fastest update rate possible. As noted earlier, in the tracking mode, the navigation module can look directly at the vision module output and adjust its control of the robot accordingly. MARIO used this ability to follow badges during the competition.

The scheduling algorithm and overall structure were extremely successful as a way to manage a robot vision system. Even with all the other robot modules running, the vision module was able to maintain a frame rate of at least 20 hertz during the competition. Information updates occurred regularly enough that each agent was able to attend to multiple aspects of its environment with real-time reactions.

The interesting new capabilities and algorithms we developed this year were person detection and identification, shirt-color identification, food-tray analysis, and Italian flag detection. For details on the motion, color-blob, and P-similar pattern detection, see Maxwell et al. (1999a, 1999b) and Scharstein and Briggs (1999).

Person Detection and Identification

Person detection is one of the most important capabilities for an interactive robot to possess.

We used two independent techniques to achieve this capability: (1) motion and (2) face detection. Our motion detector was straightforward and was described earlier, but we took a slightly novel approach to face detection that resulted in a fairly robust technique in the hors d'oeuvres domain.

The basis of our face-detection system is skin-color blob detection. The key to skin detection is effective training because lighting conditions can strongly affect the appearance of colors. We developed a fast, interactive training algorithm that gives the user direct feedback about how well the system is going to perform under existing conditions. The output of the training algorithm is an *rg* fuzzy histogram, where *r* and *g* are defined as in equation 1.

$$r = \frac{R}{R+G+B} \quad g = \frac{G}{R+G+B} \tag{1}$$

A fuzzy histogram is a histogram with entries in the range [0, 1] that indicate membership in the colors of interest. You can create a fuzzy histogram by taking a standard histogram, which counts the occurrences of each *rg* pair, and dividing each bucket by the maximum bucket value in the histogram (Wu, Chen, and Yachida 1999).

We use fuzzy histograms to convert standard images into binary images that contain only pixels whose colors have high fuzzy membership values. For skin-color blob detection, we train the fuzzy histogram on skin-color regions of some training images and then keep only pixels with membership values above a specified threshold. To get blobs, we run a two-pass segmentation algorithm on the binary image and keep only regions larger than a certain size (Rosenfeld and Pfaltz 1966).

The result of the blob detection is a set of regions that contain skin color. In previous competitions, we ran into trouble using just blob detection because the walls of the hors d'oeuvres competition areas in 1998 and 1999 were flesh tones. Although this was not the case in the 2000 competition, there were other sources of skin color in the environment besides people.

Our solution to this problem is to multiply a gradient image with the skin-color probability image prior to segmentation. The gradient image, however, is prefiltered to remove high-gradient values (that is, strong edges). The result is a gradient image where mild gradients are nonzero, and all other pixels are zero or close to it. Faces are not flat and contain mild gradients across most of their surface. However, they do not tend to contain strong edges. Thus, including the midrange, gradient values effectively eliminate walls, which are flat and tend to be featureless but leave faces. We found the combination to be robust, and it reduced our false positive rate to near zero and still reliably located people.

In the 1999 competition, our robot ALFRED tried to remember people based on texture and color histograms. This approach worked okay at the competition, but it relied on the person standing directly in front of the camera, which was rarely the case. In 2000, we decided to integrate the person identification with the face detection and shirt-color identification. We also decided not to store a permanent database of persons but instead to only recall people for a short time period. The purpose, therefore, of the person identification was to discover if a particular person was standing in front of the robot-agent for an extended period of time.

After a successful face detection, if the memory feature is activated and called, then the memory algorithm extracts a bounding box around the person's body based on the location of his/her face. It then extracts a short feature vector from the box to represent the person's identity. The feature vector is the top five buckets in an *rg* histogram, as defined in the top five buckets in an (intensity, blue) histogram, the average edge strength as determined by *X* and *Y* Sobel operators, the number of strong edge pixels, and the number of significant colors in the *rg* histogram. These 12 numbers provide a nice key with which we can compare people's appearance.

Once the system extracts a key, it compares the key to all other keys recently seen. The system stores the 100 most recent unique keys. If it finds a probable match, then it sends the matching identification to an output filter. If it finds no match, it adds the key to the data base and then calls the output filter with the new identification value. The output filter simply returns the most common key identified in the past 10 calls. However, if no single key has at least 3 matches in the past 10, it returns a null result (no match). The output filter guarantees that, even in the presence of a person's motion and schizophrenic face-detection results (jumping between people), if a person is standing in front of the camera for an extended period of time, his/her key will register consistently.

We used this capability with ALFREDO. If a person was standing in front of ALFREDO for a minimum period of time, it would comment that he/she should go do something else. Clearly, there are other applications, but we did not pursue them for lack of time.

Shirt-Color Identification

Like the memory operator, the shirt-color operator depends on a successful person detection

Figure 4. Examples of the Vision Module in Action.

Top: Successful face detection and the corresponding box used for shirt-color and person identification. Center: Successful flag detection. Bottom: Training system for face-detection system.

(face or motion). Once a person is detected, the algorithm selects a section of the image that corresponds to the person's shirt, as shown in the top photograph in figure 4. The algorithm then analyzes a histogram of this region to determine the dominant color. The difficult aspects of this task are selecting a histogram space to use and attaching color labels to regions of the space (figure 5).

Based on experimentation, we selected the *rgI* histogram space to represent color, where *I* is intensity, and *r* and *g* are the normalized coordinates defined by equation 1. (*R*, *G*, *B*) are the raw pixel values returned by the camera for a given pixel. The benefit of using the *rgI* space is that the color, represented as *rg*, is then independent of the intensity, represented in the *I* axis. We used 20 buckets in each of *r* and *g*, and 4 buckets in *I*.

Because different camera settings and different lighting affect where a color sits in the *rgI* space, we calibrate the system using a MACBETH color chart prior to each situation in which the robot interacts. Using a picture of the color chart under the appropriate illumination, we identify the centroid in the *rgI* space for each of the 24 colors on the color chart.

After identifying the region of interest, that is, the shirt region, the system identifies the most common color in the rgI histogram. The system then finds the closest, in a Euclidean sense, color centroid and returns its text color label as the output. ALFREDO used this system to great effect during the competition. It correctly identified numerous shirts, including Bruce Maxwell's mother, who was wearing a purple shirt. It made the computer appear cognizant of its surroundings in an engaging manner.

Food-Tray Analysis

The food-tray analysis is a simple but effective algorithm. We have two cameras on the robot connected to an Osprey 100-frame grabber card with multiple composite video input. On entering the TRAY mode, the vision module switches to analyzing the input from a small grey-scale camera mounted on top of the tray looking across it. During the competition, we used a white napkin to cover the tray and served dark brown or black cookies.

The tray-analysis algorithm works on the middle half of the image, where the tray dominates the scene. The operator simply counts the number of dark pixels and calculates the percentage of the visible tray that is full. Using precalculated minimum and maximum values, the operator sets a flag that specifies full, empty, or a percentage in between, which turns out to be a good proxy for how many cookies remain on the tray. Because the small camera

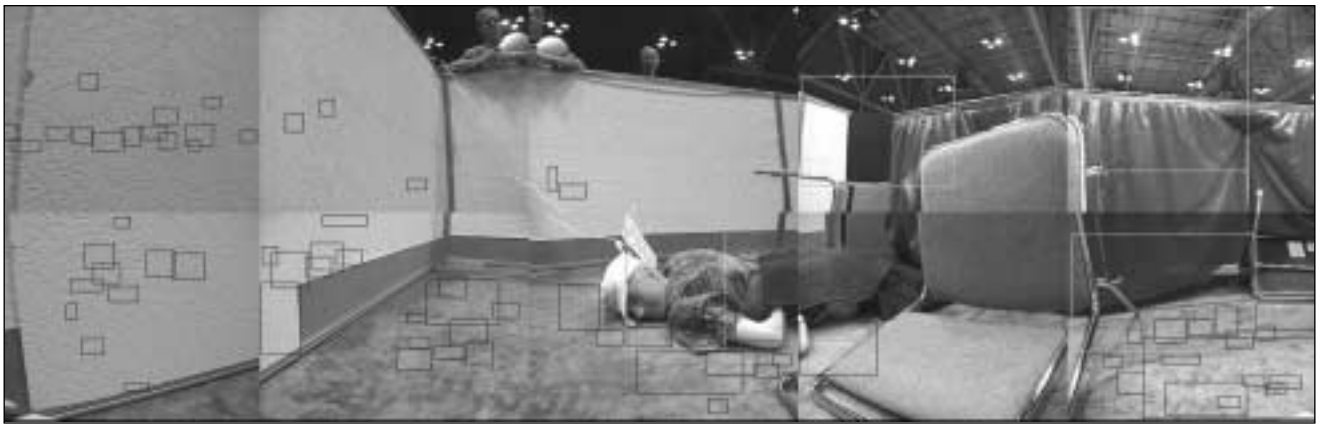

Figure 5. Panoramic Image Projected from the Vision Module During the Urban–Search-and-Rescue Contest.

The boxes indicate possible motion and skin color, respectively. The skin color on the mannequin arm, on which we trained, is grey, which is why the floors and wall get highlighted.

includes an auto-gain feature, this method works even when someone blocks the direct lighting by leaning over the tray or standing so it is in shadow.

Based on the percentage-full values returned by the vision module, the controller can smoothly transition from pure serving to serving while heading toward the refill station to heading directly to the refill station because the tray is empty. This feature worked well during the competition.

Vertical Italian Flag (Red-White-Green) Detection

Finally, we gave the robots for the hors d'oeuvres event the ability to strike up conversations with one another. To make this capability realistic, it should only happen when the robots are near to each other, which means they need the ability to visually recognize each other. We originally considered putting *p*-similar patterns—easily recognizable targets—on each robot. However, this method would have detracted from the robot's appearance, which was something close to formal dress.

Because our theme was an Italian restaurant, we used the Italian flag colors—red, white, and green—as our identifying feature. SANTINO had a flag draped vertically from his serving tray, and MARIO had one placed on an antenna about four feet above the ground. ALFREDO could also initiate conversations when it saw one of the mobile robots in its camera. To differentiate the two mobile robots, we reversed the order of the colors for MARIO and SANTINO from top to bottom.

The technique we use for flag recognition is based on traversing columns because the colors are arranged vertically. Along each column, a state machine tracks the order of the pixels. The state machine only outputs a positive identification if it finds a vertical series of red, white, and green pixels (or in reverse order). Each color has to be mostly continuous and contain a sufficient number of pixels. The state machine allows a certain number of invalid (not red, white, or green) pixels as it traverses the colors. However, too many invalid pixels invalidates a particular recognition and resets the state machine.

This method, because it is based on single columns, turns out to be robust and easy to execute in real time. The recognition system worked well both in test runs and in the competition. Because SANTINO was almost continuously engaged in serving during the competition, however, it was never able to respond to MARIO during the competition. For us, watching the robots engage one another prior to the competition was one of the highlights of the experience.

Lessons Learned and Looking to the Future

The products of our experience that we will continue, and are continuing, to use are the overall REAPER architecture, the navigation modules, the face module, and the vision module. All these provide us with generic scaffolding on top of which we are building other capabilities and systems. All of them are extendable and easily integrated with one another. We also now have excellent debugging tools that permit us to track all the information and messages that pass between modules during execution. For us, this infrastructure is the real outcome.

What we also learned is that designing the controller module is still more art than science. From a practical point of view, if we continue to use the state-machine approach, we need to build a set of standard techniques for manag-

ing and passing information around the system. We have already started some of this building, but it needs to be approached in a more formal manner. One alternative is to start building a generic state controller that uses a knowledge management system and a set of rules to determine its actions. This method would implement a true three-layer architecture where the controller sits between a reactive system and a deliberative symbolic system (Kortenkamp, Bonasso, and Murphy 1998). The REAPER architecture is ideal for this situation because the perceptual information is provided to the controller module in a symbolic and real-time manner.

Looking to the future, if the Hors d'Oeuvres, Anyone? event continues, then the challenge is to push the envelope. On the interaction front, one challenge is to develop a more generic speech-interaction system that can engage in and follow conversations, albeit within a limited domain. A second is to fully implement an emotional subsystem that can affect the whole range of robot behaviors. A third is to more closely link visual recognition of features, such as shirt color, with the interactions in a natural manner. We came close to that goal this year, but to be smooth, it must be integrated with a more generic speech-interaction system.

On the navigation front, coverage of the serving area has only been achieved by MARIO, mostly because it never stopped to talk. Combining MARIO's ability to move in a crowd with a more effective SANTINO will be difficult because at some point the robot has to take the initiative, stop interacting, and move on.

Finally, the multirobot system proved to be both entertaining and successful at solving the task. Future competitions should encourage multiple-robot interaction; two teams attempted it this year. They will have to deal with the fact that it is difficult for the robots to get to one another, but it should be possible.

In the USAR task, the challenge is clear. The autonomous entries covered only a small amount of the test area, mostly because of limitations in their ability to sense and interpret the realities of the situation. The teleoperated entry, however, did not give much responsibility to the robots. Building meaningful maps, correctly flagging important features or injured people, and simply getting out of the test area within the time limit should be minimal goals for future entries. We believe the techniques exist to accomplish these goals, but their integration in a single package has yet to be completed.

References

Bizzi, E.; Giszter, S.; Loeb, E.; Mussa-Ivaldi, F. A.; and Saltiel, P. 1995. Modular Organization of Motor Behavior in the Frog's Spinal Cord. *Trends in Neuroscience* 18:442–446.

Bonasso, R. P.; Firby, R. J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiments with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(2–3): 237–256.

Brooks, R. A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* 2(1): 14–23.

Bryson, J. 2000. Cross-Paradigm Analysis of Autonomous Agent Architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 12(2): 165–190.

Forsey, D. R., and Bartels, R. H. 1980. Hierarchical B-Spline Refinement. *Computer Graphics (SIGGRAPH '88)* 22(4): 205–212.

Gat, E. 1991. Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots, Ph.D. thesis, Department of Computer Science, Virginia Polytechnic Institute and State University.

IBM. 1999. IBM VIAVOICE OUTLOUD API Reference Version 5.0. Available at www-4.ibm.com/software/speech/dev/ttssdle_linux.html.

Ifeachor, E. C., and Jervis, B. W. 1995. *Digital Signal Processing. A Practical Approach*. Reading, Mass.: Addison Wesley.

Kortenkamp, D.; Bonasso, R. P.; and Murphy, R., eds. 1990. *Artificial Intelligence and Mobile Robots*. Menlo Park, Calif.: AAAI Press.

Maxwell, B.; Anderson, S.; Gomez-Ibanez, D.; Gordon, E.; Reese, B.; Lafary, M.; Thompson, T.; Trosen, M.; and Tomson, A. 1999a. Using Vision to Guide an Hors d'Oeuvres Serving Robot. Paper presented at the IEEE Workshop on Perception for Mobile Agents, 26 June, Fort Collins, Colorado.

Maxwell, B. A.; Meeden, L. A.; Addo, N.; Brown, L.; Dickson, P.; Ng, J.; Olshfski, S.; Silk, E.; and Wales, J. 1999b. ALFRED: The Robot Waiter Who Remembers You. Paper presented at the AAAI Workshop on Robotics, 23 July, Orlando, Florida.

Moravec, H. P., and Elfes, A. E. 1985. High-Resolution Maps from Wide Angle Sonar. In Proceedings of the IEEE International Conference on Robotics and Automation, 116–121. Washington, D.C.: IEEE Computer Society.

Neider, J.; Davis, T.; and Woo, M. 1993. *OPENGL Programming Guide: The Official Guide to Learning OPENGL*. Reading, Mass.: Addison-Wesley.

Parke, F. I., and Waters, K. 1996. *Computer Facial Animation*. Wellesley, Mass.: A. K. Peters.

Rosenfeld, A., and Pfaltz, J. L. 1966. Sequential Operations in Digital Picture Processing. *Journal of the ACM* 13(4): 471–494.

Scharstein, D., and Briggs, A. 1999. Fast Recognition of Self-Similar Landmarks. Paper presented at the IEEE Workshop on Perception for Mobile Agents, 26 June, Fort Collins, Colorado.

Wu, H.; Chen, Q.; and Yachida, M. 1999. Face Detection from Color Images Using a Fuzzy Pattern-Matching Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21(6): 557–563.

Bruce Maxwell is an assistant professor of engineering at Swarthmore College. He received a B.A. in political science and a B.S. in engineering, with a concentration in computer science, from Swarthmore College; an M. Phil. from Cambridge University; and a Ph.D. in robotics from the Robotics Institute at Carnegie Mellon University. His primary research interests are computer vision, robotics, computer graphics, data mining, and visualization.

Lisa Meeden is an associate professor of computer science at Swarthmore College. She received a B.A. in mathematics from Grinnell College and a Ph.D. in computer science, with a minor in cognitive science, from Indiana University. Her primary research interests are evolutionary robotics and robot learning.

Nii Saka Addo, Paul Dickson, Nathaniel Fairfield, Nikolas Johnson, Edward G. Jones, Suor Kim, Pukar Malla, Matthew Murphy, Brandon Rutter, and Eli Silk are all undergraduate students at Swarthmore College.