Machine Vision and Applications manuscript No.
(will be inserted by the editor)

A Real-Time Vision Module for Interactive Perceptual Agents

Bruce A. Maxwell, Nathaniel Fairfield, Nikolas Johnson, Pukar Malla, Paul Dickson, Suor Kim, Stephanie Wojtkowski, Thomas Stepleton

Swarthmore College
500 College Ave.
Swarthmore, PA 19081
maxwell@swarthmore.edu

Received: date / Revised version: date

Abstract Interactive robotics demands real-time visual information about the environment. Real time vision processing, however, places a heavy load on the robot's limited resources, which must accommodate multiple other processes running simultaneously. This paper describes a vision module capable of providing real-time information from ten or more operators while maintaining at least a 20Hz frame rate and leaving sufficient processor time for a robot's other capabilities. The vision module uses a probabilistic scheduling algorithm to ensure both timely information flow and a fast frame capture. In addition, it tightly integrates the vision operators with control of a pan-tilt-zoom camera. The vision module makes its information available to other modules in the robot architecture through a shared memory structure. The information provided by the vision module includes the operator information along with a time stamp indicating information relevance. Because of this design, our robots are able to react in a timely manner to a wide variety of visual events.

1 Introduction

Interactive robotics is a demanding task for real-time vision systems. The motivation for our system came from two robot competition events–Hors d'Oeuvres Anyone? [HA] and Urban Search and Rescue [USR]–at the American Association for Artificial Intelligence [AAAI] annual conference. The Hors d'Oeuvres Anyone? event requires the robot(s) to serve food to the conference attendees at the main conference reception with over 500 people in attendance. The robots are evaluated based upon their ability to cover the room, manipulate food, and interact with the people they are serving. Robot-human interaction is weighted most heavily of the three.

The USR event requires the robot(s) to explore a test arena designed by the National Institute of Standards and Technology [NIST]. The test arena simulates the inside of a building after some type of disaster situation and contains three sections of increasing difficulty. The simplest level is navigable by a wheeled mobile robot, the other two sections require a tracked or 4-wheel drive robot. Robots are evaluated on their ability to explore the arena, detect humans–simulated by motion, heat, and skin-colored mannequins–and exit the area within the allotted time (25 minutes).

Both events benefit greatly from the use of visual input. The HA event, in particular, requires the robot to sense and react to people and various related features, such as colored conference badges in a timely and appropriate manner. The more information the visual sensing can provide, the more interactive and responsive the robot can be to the people it is serving. In the USR event, visual sensing gives the robots one method of detecting and labeling important features in the environment. In some cases it is the only sensing modality able to reliably detect the simulated people.

Rather than have custom vision processing and robot architectures for each event/platform, we use a single software architecture for all of our intelligent agents, with different controlling modules for the different events. Both the general architecture and the vision module are shared between different platforms–Nomad Super Scout II mobile robots, Magellan Pro mobile robots, and workstations acting as stationary agents. We have used all three in the HA event; we have used only the Magellan Pro robots in the USR event.

The robots work under significant processing constraints. The Nomad robots use a 266MHz Pentium MMX processor, and the Magellan robots use a 350MHz or 450MHz Pentium II. For the HA task, the Nomad's capabilities include vision, speech recognition, speech synthesis, visual processing, a sliding arm controller, navigation, control, facial animation, and inter-robot communication. The Magellan possesses similar software capabilities but does not attempt speech recognition or

possess a moving arm. The workstations tend to be better equipped computationally, but get correspondingly heavier loads. For example, a workstation may be simultaneously running the vision module with real-time facial animation and synchronization of the lips with speech synthesis.

Because of the wide array of capabilities demanding processor time, the vision module has to avoid monopolizing the processor while still capturing video in real-time and providing timely information. Furthermore, unlike most other real time vision systems, we ask the robot's vision system to undertake multiple tasks simultaneously, rather than dedicating the vision system to a single task such as navigation or skin detection. To manage multiple operators and still deliver real time performance, the vision system uses a stochastic operator scheduler, pipelines complex tasks, and integrates pan-tilt-zoom camera management with operator scheduling.

The combination of our overall architecture and the vision module achieved the goal of real time human-robot interaction using visual input. In part because of this design, at the 2000 AAAI robot contest our robots won both events as well as the Ben Wegbrait award for the integration of AI technologies. In 2001, we took 2nd in the HA event and garnered the highest point total in USR under a new set of rules. The remainder of this paper describes related work, gives a brief overview of the overall architecture, and a then presents a detailed description of the real-time vision module.

2 Related Work

Building real time vision systems is an exercise in the art of compromise. The complexity, robustness, and performance of an algorithm must be balanced by both its latency and total processing time. Latency, in particular, is critical in robot systems that must react appropriately to their environment.

Until the advent of general purpose microprocessors able to run several hundred million operations per second, real time vision systems were built from dedicated hardware, such as Kanade's real-time stereo machine [10]. Even modern robot vision systems often use either a separate special-purpose vision processor [17] or send the video via a wireless connection to a stationary workstation or cluster of workstations [7], which may also have special purpose processing hardware [6].

Most systems that undertake real-time vision processing using general purpose computers only require the vision system to do a single task. One example of a stationary system in this category is the W4 person detection and tracking system [8]. Robot systems that follow this paradigm include Horswill's robots that use vision for navigation [9], and Minerva, the robot tour-guide, who used a camera to watch the ceiling and localize the robot [23]. Because these vision systems are only executing a single program, they have no need for independent scheduling mechanisms or general purpose communication paradigms.

The concept of dividing a set of vision tasks into separate modules, however, is useful both for optimizing processor usage and for distributing computation across multiple computers, if they are available. Cheng et. al., for example, have a robot vision system that consists of six separate modules–which they call agents–that each work on different aspects of the vision system [7]. They pipe the video off the robot and distribute the processing between multiple general purpose computers using point-to-point communication across ethernet. Load balancing is done manually through the initial assignment of modules to computers. Each process is running continuously, usually on its own dedicated processor.

Stasse and Kuniyoshi took a broader approach to the problem and developed a framework for designing distributed robotic software [22]. This framework includes a langauge and visual interface for describing processes, process communication, and network topology. Their test case vision system used seven general purpose PCs linked by an ATM switch for doing a stereo vision task. It is not clear, however, how this framework would apply to a set of processes running on a single processor.

In contrast to these related systems, our robot has a single processor that manages all aspects of the robot's interface, sensing, and control. Our overall system architecture is designed to facilitiate this by permitting fast communication between different processing modules and flexible selection of which modules are active on which robots. Within this overall system architecture, our vision module handles multiple independent and interdependent real time vision tasks, the scheduling of operators, communication with other modules, and control of pan-tilt-zoom capabilities of the camera.

3 REAPER: an Intelligent Agent Architecture

The overall system architecture, named REAPER [REflexive Architecture for PErceptual Robotics], is based on a set of modules. The purpose of each module is to handle one specific task, which include: sensing, reflexes, control, communication, and debugging. 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 like how fast to turn each wheel. 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 of the preprocessing and generating intermediate actions required to achieve high-level commands or goals. This is similar to the way our brain seems to deal with a request to pick up an object. While we consciously think about picking

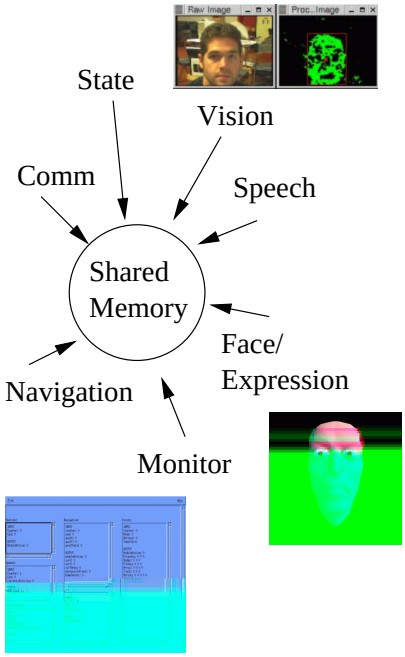

Fig. 1 Logical diagram of the REAPER Architecture. Each module takes inputs from and writes its outputs to the shared memory. The State module is the central controlling unit.

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 control to decide what to do next. This approach has some biological justification [1].

Two sensing modules, vision and speech, 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. Two reflex modules, navigation and face, handle the motion and appearance of the robot. 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 of 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 in the architecture communicate through a shared memory structure, which provides a computationally efficient means of sharing information. The overall framework is shown graphically in figure 1. 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 wireless ethernet system. Currently, we use the robots' communication abilities to execute humorous dialogues, although they could also be used to pass state information to one another.

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 that an autonomous agent architecture needs:

1. a modular structure,
2. a means to control action and perception sequences for complex tasks, and
3. a mechanism for reacting quickly to changes in the environment [4].

REAPER is related to behavior-based systems, such as those proposed by Brooks, in that it distributes low-level interpretation and control of sensors and actuators to modules [3]. REAPER is not a pure behavior-based system, however, because it is designed to facilitate the mixing of symbolic and subsymbolic, reactive strategies. Thus, REAPER is also related to hierarchical architectures, which facilitate the development of action and perception sequences for complex tasks [2].

The real strength of the system, however, is REAPER's modular design which facilitates cross-platform and multi-platform development and simplifies debugging and testing of the individual modules. This is particularly true in the case of the vision module, which we extensively tested on its own before putting it on the robot platform.

At the most fundamental level, the REAPER approach evolved because it enables the agent to gather information at a tremendous rate from a large number of sensors and present that information in a useful, symbolic form to the controlling module. This overall architecture for a perceptive agent is the framework within which we developed the real-time vision module. Our goals in designing the vision module were to avoid dominating the robot's processor, provide easy access to a wide variety of information about the world, and maintain as fast a frame rate as possible to avoid time lags between sensing and action. The remainder of the paper describes how we balanced these goals within an integrated robot system.

p i V f E

modules it is active. If a pan-tilt-zoom camera is present, it gets initialized and set to its home position. The module then enters an event loop–initially in an idle state. Each time through the event loop, the module tests if the controller has specified a new configuration mode. If so, the transition to executing that mode takes place. Otherwise, the module continues executing the current mode and operator set. The module continues to process and update sensory information until told to do something else.

In addition to running vision operators, the vision module also controls the pan-tilt-zoom aspects of the camera, if that capability exists. We currently support both the Sony EVI-D31 and the Canon VC-C4 PTZ cameras, both of which use serial port commands (but not the same ones).

4.1 Modes and operator scheduling

The vision module includes a rich set of operators for converting images into symbolic information. The three general classes of operators are: object detection, motion detection, and object characteristic analysis. Fields within a shared memory structure indicate the current mode and active operators. Information generated by those operators is placed back into the shared memory structure. Other modules within the REAPER architecture can scan 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. Based on the time stamp and the operator information, other modules can decide what information requires a response.

The set of operators provided by the vision module include:

- Person detection based on skin color and gradients
- Moving area detection across multiple frames
- Color blob detection, focused on colored conference badge detection
- P-similar pattern detection
- Red, white, and green flag detection
- Palm detection
- Green ring detection
- Shirt color analysis (dependent upon detecting a person)
- Person identification (dependent upon detecting a person)
- Calculation of how much food is on the robot's tray
- Panoramic image capture
- Conference badge detection and tracking, combined with text detection

The major modes of the vision module determine the resolution and select which camera is active. For example, in the 2000 competition we used a second camera to watch the tray of cookies in the HA event. In the 2001 competition we used a pan-tilt-zoom [PTZ] camera that could tilt down to check the tray. There are two possible cameras–camera 0 and camera 1–and two possible resolutions–120x160 and 240x320. The four combinations plus an IDLE and a QUIT state form the possible modes for the vision module. Each time through the event loop the vision module checks to see if its mode has changed. If so, it makes the necessary changes on the fly and goes back to processing the active operators.

Whatever camera or resolution, with the limitations on processing power there is clearly no way to maintain a high frame rate and execute all of the operators listed above on each image. Furthermore, we probably are not interested in running all of the operators simultaneously.

A more useful way to frame the problem is one of information gathering. Each of the vision operators represents an information stream. Each packet of useful information has an associated computational cost and relevance, which for our purposes is defined as elapsed time since the associated image was taken. Our goal is to maintain an arbitrary number of information streams while keeping both the overall computational load below a certain percentage of the processor's capability and the relevance of the information in each stream high. If the computation time for a single operator takes too long, then its relevance diminishes and the information is useless to the robot.

The task definition is complicated by the fact that different operators have different computation times, variable and unpredicatable computation times (although normally bounded), and different needs in terms of how often each operator needs to run. For example, a tracking operator needs to run on a significant percentage of the incoming frames in order to react appropriately. Other operators, however, need to run only a few times per second. In the extreme case, an operator may need to run for only a few continguous frames every couple of minutes.

Researchers have developed a number of useful results in the field of scheduling processes for real time systems [21]. However, few of them apply to a real time vision system. In addition to the variability of the tasks, the tasks required for a single frame must be completed before the next frame can be captured. Since it is clear that not all operators can be completed on each frame, but that we want information from all operators, then we have to interleave operators among frames in a manner that is appropriate for the characteristics of each operator. Something like a deadline monotonic policy applied at the frame level would only run the highest priority tasks–tasks that required access to most of the frames and, therefore, had the shortest period–and never get to the others [11].

Rather than trying to develop an optimal scheduling policy for each frame or set of frames, our system is built to provide good average characteristics over long peri-

ods of time. To both limit the computational load and interleave operators, the vision module uses a scheduling algorithm that applies at most two operators to each frame, selected stochastically based on a priority rating. The premise of this approach is that running two operators per frame will not reduce the frame rate below an acceptable level or unduly tax the processor. This puts an upper bound on operator complexity, although we get around the limitation by pipelining complex operators.

A stochastic operator selection mechanism works because, as noted above, the robot usually doesn't need to know that there is a pink blob in view–or whatever other object–30 times per second. Frame rate is usually a lot faster than the robot can react to things since reactions often involve physical actions or speaking. Likewise, many of the operators do not benefit from continuous application in an interactive situations. Since the module supplies a time stamp with each piece of information, the information persists and the controller can decide based on the time stamp whether a piece of information is recent enough to warrant a response.

While the short term characteristics of this algorithm may not be optimal in terms of providing relevant information in each stream, the long-term behavior of the system maintains relevant information on multiple information streams and delivers proportionately more frames and computation time to operators with high priority.

To deal with the heterogeneous nature of the vision operators, the main controller can label an operator as being either *stochastic* or *fixed* in its timing. A *stochastic* operator has a priority associated with it. If all of the operators are *stochastic*, then the process for selecting two is straightforward. Through random number generation, the system picks two processes, with higher priority processes having a correspondingly higher probability of being picked. The probability of any one operator being picked to run on any given frame is its priority divided by the sum of the priorities of all active operators. An operator can only take one slot, however, so if there are only two active operators they will both run on every frame. Over time, an operator runs on average proportional to its relative priority. Using relative priorities makes it easy for a person to specify the system configuration.

Since not all operators are applied on each frame, there can be a time lag between operator applications. To communicate the freshness of data to other modules, the vision module supplies a time stamp with each piece of information. The information persists until the next application of the operator, and the controller can decide based on the time stamp whether it is recent enough to warrant a response.

The need for *fixed* timing operators grew out of our use of the PTZ camera to look at the cookie tray in the 2001 HA event. The PTZ camera had to look down to see the tray, but we wanted the camera facing up most of the time. It is not necessary for the PTZ camera to look down at the tray of cookies more than once every minute or so, and even then it needs to look only for a short period of time (10 frames or less of analysis). The desired behavior, therefore, is for the cookie tray analysis operator to run for only a few frames every 30 seconds.

An operator with a *fixed* timing label, therefore, is only put in the queue when it needs to run. When it activates a *fixed* operator, the controller specifies how many seconds should exist between runs. When the current time *mod* the timing value is less than a few seconds–currently set at four–the operator is given a high priority and put into the stochastic queue along with all of the other active operators. Thus, a fixed operator has a high likelihood of running at evenly spaced intervals, where the intervals can be anything longer than four seconds.

In some situations, such as tracking, it is important for an operator to run every frame. As noted above, if there are at most two active operators, it will guarantee that both are executed every frame. With more operators, it is possible to get close to every frame execution of one operator by making its priority much higher than all of the other priorities combined. Coordinating the visual tracking with physical motion within the REAPER architecture is straightforward to do because the navigation module can look directly at the vision module output and adjust its control of the robot accordingly. One of our Magellan Pro robots used this ability to follow pink badges in the 2000 HA event.

The scheduling algorithm and overall structure were a successful way to manage our robot vision system. Even with all of the robot modules running, the vision module was able to maintain a frame rate of at least 20Hz during the competition. Information updates occurred often enough that the robots could attend to multiple aspects of their environment with real-time reactions.

4.2 Pan-tilt-zoom camera controller

With the development of more affordable pan-tilt-zoom [PTZ] cameras, they are becoming more common on interactive mobile robotics. Effective use of the PTZ capabilities requires tight integration with the vision processing. One method of achieving this integration is to have a single vision operator be responsible for the PTZ capabilities. This method is too inflexible, however, given the number of operators that may want to control some aspect of the camera. Instead, we developed a generic framework within which any operator has the potential to control the camera and operators must share control of the camera as appropriate.

Three observations form the basis for the PTZ control algorithm. First, the PTZ should only be controlled by a single operator at any given time. This ensures that there will be no conflicting commands. It also ensures that the needs of at least one operator are met–the one

3. Flag detection (red/white/green and blue/red): timing = *stochastic*, ptz = *default*
4. Cookie tray analysis: timing = *fixed*, ptz = *fixed*

The resulting behavior was nicely appropriate for the task. Most of the time, the camera was searching for the first three items. When no badges were visible, it would return to its home state of pointing slightly up, continuously running the first three operators. Approximately every 30 seconds, the PTZ would tip down and stare at the cookie tray for a few seconds. Then it would return to the idle state looking slightly up. When the camera caught sight of a badge, it would orient on the badge and try to zoom in and read the name tag. This combination of actions both provided the robot with real-time information about its environment and made the robot look perceptive and natural in its motion.

A brief description of some of the operators follows. While none of these are particularly fancy, they do display the range of real-time processing that the system is capable of running. The capabilities we developed for the 2000 competition include a new face detector, a short-term person identifier, a shirt-color identifier, a vertical Italian flag detector, and the ability to estimate how much food was left on the robot's tray. For the 2001 competition we focused on reading name tags, which required location and tracking of the name tags, identification of text blocks, optical character recognition and name generation. For more complete descriptions of other capabilities, see [14], [13], and [12].

4.3 Motion-based person detection

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 [13] [12].

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 complete–with the first three capturing images and calculating the Sobel results. To ensure the motion algorithm is called frequently enough, it has a high weight in the probability distribution. On average, the motion algorithm completes 3-4 times per second and the overall vision module maintains a frame rate of at least 20Hz. When the motion operator is active, it is usually selected as one of the operators.

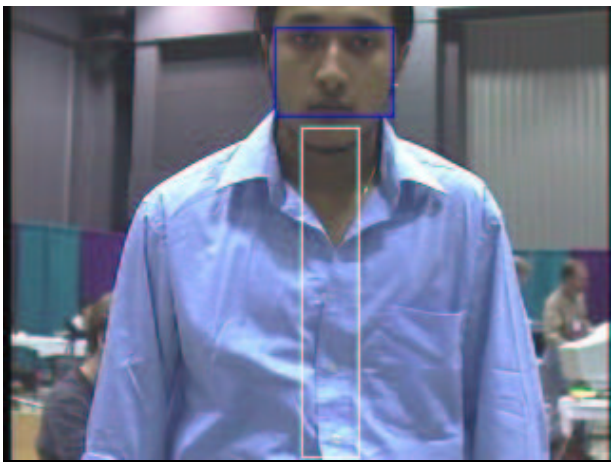

Fig. 3 Successful face detection and the corresponding box used for shirt color and person identification.

4.4 Skin-based person detection and identification

We used two independent techniques, motion and face detection, to detect people. Our motion detector is described above, but we took a somewhat novel approach to face detection that resulted in a robust technique in the HA domain.

The basis of our face detection system is skin-color blob detection. The key to skin detection is effective training (or good color constancy, which is hard), since lighting conditions strongly affect colors. We developed a fast, interactive training program 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 1.

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

A fuzzy histogram is a histogram with entries in the range [0, 1] that indicate membership in the colors of interest. To create a fuzzy histogram we generate a standard histogram and divide each individual bucket by the maximum bucket value [24].

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 several training images and then keep only pixels with membership values above a specified threshold. To get blobs we run a 2-pass segmentation algorithm on the binary image and keep only regions larger than a certain size [16].

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 HA competition areas in 1998 and 1999 were flesh-tones. While this was not the case in 2000, there were other sources of skin-color besides people in the environment.

Our solution for distinguishing between background flesh-tones and true faces is to multiply the gradient magnitude of the image with the skin-color probability image prior to segmentation. The gradient magnitude image, however, is pre-filtered to remove high gradient values (i.e. strong edges). The result is a gradient image where mild gradients are non-zero 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 mid-range gradient values effectively eliminates walls–which are flat and tend to be featureless–but leaves faces. We found the combination to be robust and it reduced our false positive rate to near zero for the event while still reliably locating people. Figure 3 shows an example face detection (upper box).

4.5 Short-term person identification

In the 1999 competition our Nomad robot, Alfred, tried to remember people based on texture and color histograms from a fixed portion of the image. Unfortunately, this relied on the person standing directly in front of the camera, which was rarely the case. This year we integrated the person identification with face detection and shirt color identification, described below. We did not store a permanent database of persons, but instead used a 100 entry buffer to 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 the memory algorithm extracts a bounding box around the person's body based on the location of their face, as shown in figure 3. It then extracts a short feature vector from that box to represent that person's identity. The feature vector is the top five buckets in an rg histogram–as defined in 1–the top five buckets in an IB (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.

To compare two keys, we use standard histogram intersection with the top five buckets in the rg and IB histograms, and then percentage difference in the average edge strength, number of strong edge pixels, and number of significant colors. Appropriate normalization provides a single number which indicates the degree of match.

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 will send the matching ID to an output filter. If it finds no match, it will add the key to the data base and then call the output filter with the new ID value. The output filter simply returns the most common key identified in the past 10 calls. However, if no single key has at least three matches in the past 10, it returns a null result (no match). The output filter guarantees that, even in the presence of a moving person and schizophrenic face detection results (jumping between people), if a person is standing in front of the camera for an extended period of time their key will register consistently.

We used this capability in 2000 with the maitre'd workstation, Alfredo. If a person was standing in front of it for a minimum period of time, it would comment that they should go do something else. Clearly there are other applications, but we did not pursue them in the competition setting.

4.6 Shirt color identification

Like the memory operator, the shirt color operator waits for a successful person detection (face or motion) and then extracts a section of the image that corresponds to the likely location of the person's shirt, as shown in figure 3 (lower box). The algorithm then generates a histogram of this region and selects the dominant color. The difficult aspects of this task are selecting the histogram space, and attaching color labels to regions of that space.

Based on experimentation, we selected the rgI histogram space to represent color, where

$$I = \frac{1}{3}(R + G + B) \tag{2}$$

I is intensity, and r and g are normalized color as defined by 1. (R, G, B) are the raw pixels values returned by the camera. 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^{TM} color chart prior to each situation in which the robot interacts–i.e. once prior to the competition. 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 chart.

After identifying the region of interest–the shirt region–the system identifies the most common color in the rgI histogram. The system then finds the closest color centroid in a Euclidean sense and returns its text color label as the output. The robot running the shirt color algorithm correctly identified numerous shirts during the competition, including Dr. Maxwell's mother, who was wearing a purple shirt. It made the computer appear cognizant of its surroundings in an engaging manner and provided it with something interesting to talk about.

4.7 Food tray analysis

The food tray analysis is a simple, but effective algorithm. We use a light colored tray and dark colored food. A reasonably selected threshold permits the vision system to count the percentage of the tray covered by food.

Fig. 4 Successful flag detection.

Fig. 5 Example of text detection on a name tag.

In the 2000 HA event we used a second camera dedicated to watching the tray. In 2001, we had the PTZ camera look down on the tray at regular intervals.

The interesting feature of the food tray analysis is its use of the fixed timing feature and fixed PTZ position in the vision module. We set the time period for this operator to 30 seconds and the position to be pointed down. These settings caused the operator to run with high probability for only a few seconds, spaced 30 seconds apart. Before running, the operator would capture the PTZ capabilities and cause the camera to look at the tray. In both years, this feature worked well and enabled the robot to respond to how much food was left on its tray by heading towards its food refill station in an appropriate manner.

4.8 Vertical Italian flag (red-white-green) detection

In the 2000 competition, we gave the robots in 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 can "see" one another, which means they must be able to visually recognize each other. Since our theme was an Italian restaurant, we used the italian flag colors–red, white, and green–as our identifying feature. The Nomad robot had a flag draped vertically from his serving tray, and the Magellan had one placed on an antenna about 4 feet above the ground. To differentiate the two mobile robots we reversed the order of the colors for one of them.

The technique we use for flag recognition is based on traversing columns in the image with a state machine that tracks the order and color 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 reversed order). Each color has to be mostly continuous and contain at least a specified 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 that particular recognition and resets the state machine. This method, since it is based on single columns, is robust and easy to execute in real time. The recognition system worked well both in test runs and in the competition. An example flag detection is shown in figure 4.

We used the same method to robustly detect red and blue flags worn by all the robot entrants in the 2001 HA competition.

4.9 Name tag tracking and text detection

Our primary goal in the 2001 HA competition was to have the robot be able to recognize and say people's names by reading their name tags. This involved three separate steps: 1) finding the name tags, 2) finding and extracting the text on the name tags, and 3) optical character recognition to recognize the letters and pass them to the speech synthesizer. After significant experimentation, we found that we had to use color features to find the name tags at a reasonable distance, then zoom in and use text features to center and extract the text. This dichotomy is supported by the four-state state machine used for PTZ control.

To find the name tags, the system used a columnar state machine, similar to the one described above, to find a colored bar followed below by white. If the system found a name tag, it would center it in the image and then zoom in on the text. Once zoomed, it would use a text detector to make boxes around the text. The text detector was loosely based on that of Smith & Kanade, and uses a horizontal gradient operator, followed by a dilation step and a segmentation step [19]. The system filters the resulting regions using minimum height and height/width ratio requirements. It then passes any boxes that pass this requirement to the optical character recognition software. This system was extremely successful at locating badges and extracting

the text boxes. Figure 4 shows an example of the text detection on an actual badge.

4.10 Optical character recognition

Given a region that is believed to contain text which has been extracted from the captured image, such as the boxed areas of Figure 4, the vision module filters the region to provide better input to the template matcher. First, the isodata method is used to find an appropriate threshold for the image [20]. This method works well since the intensities in the image are centered around two values, that of the text and that of the background. Using the filtered image, the primary axis of the text is calculated. Then, this information is used to rotate the original image so that the primary axis is horizontal. The rotation is applied to the original image and not the thresholded one to minimize the amount of aliasing in the rotation. Finally, the rotated image is thresholded to provide a clean, well-oriented input to the template matcher.

The template matcher scans the prepared image from left to right and determines the ten letters most likely to begin at each column. First the height of the letter possibly beginning at the current x location is determined by finding the top and bottom of dark regions a few columns ahead. Pre-built letter bitmaps are scaled to this height and compared to the actual image, and the ten best matches and their widths are stored at the current x value in an array the size of the width of the image. Finally, a recursive procedure seeks the highest scoring path through the array by adding the value of a letter at the current column to the score returned by the procedure called on the array element K columns ahead, where K is the width of the current letter. To prevent spurious results, this procedure is constrained by a list of actual names compiled from the U.S. Census Bureau Name List [5].

This procedure worked well in pre-contest tests. However, increased illumination in the contest setting washed out the badges due to saturation, and we did not have a chance to recalibrate the camera settings. Future work in badge reading includes developing an automatic calibration procedure so that illumination changes do not so drastically affect the results.

4.11 Stereo panoramic image capture

As a final capability for our 2001 entry in the USR competition, we gave our robots the ability to take stereo panoramic images with a single camera using the theory described in [15] and [18]. The theory states that if you move a camera in a circle, taking images continuously, then concatenating a strip from the left and right side of each image will generate a right and left stereo pair. The disparity of the stereo pair is controlled by the radius of

Fig. 6 Left and right panoramic images taken during the 2001 Urban Search & Rescue Competition (view images after rotating them clockwise)

the circle the camera traverses and the strips selected for concatenation.

In our system we used 120x160 images and a strip starting ten pixels from the center of the image. The strip width is determined automatically by the vision module based on the expected frame rate and the turning speed of the robot. To capture a panoramic stereo pair, the controller starts the robot turning and tells the vision module the turning rate. It then tells the vision module to start capturing a panoramic image. The panoramic image operator has *stochastic* timing and a *fixed* PTZ position (straight forward). The vision module then fills up left and right image buffers and sets a flag when it completes the capture. After writing the images to a file it is ready to capture another panoramic image. Figure 6 shows example left and right images taken during the 2001 USR event. In addition, from the stereo pair we can generate an anaglyph image–a red-blue stereo pair– by properly overlaying the two images. This creates a depth effect that gives the user a sense of the robot's environment. The ability to capture panoramic images is one more example of the generality of the vision module framework.

5 Summary

The vision module described here, is a product of our experience that we are continuing to use on multiple robot and stationary vision systems. The vision module, combined with the overall REAPER architecture, provide us with a 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 debugging tools that permit us to track all of the information and messages that pass between modules during execution. For us, this infrastructure is the real outcome of this project.

In particular, we have continued to use and develop the vision module for research purposes. It is straightforward to add capabilities to the module and use it for a variety of tasks. The probabilistic scheduling algorithm and wide variety of capabilities make it ideal for a mobile robot that is processor limited but must continuously monitor and react to multiple aspects of its environment.

As shown by the motion operator, the vision module is not limited to simple operations. We can use more complex operators by pipelining the processing across multiple event loops. Ultimately we are limited by the processing power of the robot, but clearly operators of moderate complexity are feasible within the real-time vision module approach.

Finally, the PTZ control and tracking capability gives the module the ability to make intelligent use of a pan-tilt zoom camera. The combination of fast image capture, the application of operators in a probabilistic manner, pipelining of complex operations, and the tight integration with camera control, avoids overloading the robot's processor and permits it to respond appropriately to the visual world.

References

1. E. Bizzi, S. Giszter, E. Loeb, F.A. Mussa-Ivaldi, and P. Saltiel. Modular organization of motor behavior in the frog's spinal cord. *Trends in Neuroscience*, 18:442–446, 1995.
2. R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiments with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2/3):237–256, 1997.
3. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
4. J. Bryson. Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190, 2000.
5. U.S. Census Bureau. U.s. census bureau name list, July 2001.
6. G. Cheng and A. Zelinsky. Real-time visual behaviors for navigating a mobile robot. In *Proceedings of Int'l Conf. on Intelligent Robots and Systems*, pages 973–980. IEEE, 1996.
7. T. K. Cheng, L. Kitchen, and Z.-Q. Liu. Multi-agent real-time 3d tracking in robot vision. In *Proceedings of 18th Australasian Computer Science Conf [ACSC'95]*, pages 80–89, 1995.
8. I. Haritauglu, D. Harwood, and L. S. Davis. W4s: A real-time system for detecting and tracking people in 2 1/2 d. In *Proceedings of European Conference on Computer Vision*, pages 877–892, 1998.
9. I. Horswill. Polly: A vision-based artificial agent. In *Proceedings of 11th Nat'l Conf. on Artificial Intelligence*, pages 824–829. AAAI Press, 1993.
10. T. Kanade, H. Kato, S. Kimura, A. Yoshida, and K. Oda. Development of a video-rate stereo machine. In *Proceedings of International Robotics and Systems Conference (IROS '95)*, volume 3, pages 95–100, August 1995.
11. J. Leung and J. Whitehead. On the complexity of fixed priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.
12. B. Maxwell, S. Anderson, D. Gomez-Ibanez, E. Gordon, B. Reese, M. Lafary, T. Thompson, M. Trosen, and A. Tomson. Using vision to guide an hors d'oeuvres serving robot. In *Proceedings of IEEE Workshop on Perception for Mobile Agents*, pages 37–43, June 1999.
13. B. A. Maxwell, L. A. Meeden, N. Addo, L. Brown, P. Dickson, J. Ng, S. Olshfski, E. Silk, and J. Wales. Alfred: The robot waiter who remembers you. In *Proceedings of AAAI Workshop on Robotics*, July 1999.
14. B. A. Maxwell, L. A. Meeden, N. Addo, P. Dickson, N. Fairfield, N. Johnson, E. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, and E. Silk. Reaper: A reflexive architecture for perceptive agents. *AI Magazine*, 22(1):53–66, 2001.

15. S. Peleg and M. Ben-Ezra. Stereo panorama with a single camera. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 495–401, June 1997.
16. A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *ACM*, 13:471–494, October 1966.
17. R. Sargent, B. Bailey, C. Witty, and A. Wright. Dynamic object capture using fast vision. *AI Magazine*, 18(1), 1997.
18. S. Seitz. The space of all stereo images. In *Eighth Int'l Conf. on Computer Vision*, pages 26–33, June 2001.
19. M. A. Smith and T. Kanade. Video skimming and characterization through the combination of image and language understanding techniques. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 775–781, June 1997.
20. M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. PWS Publishing, Pacific Grove, 1999.
21. J. Stankovic, M. Spuri, and M. D. Natale. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, 1995.
22. O. Stasse and Y. Kuniyoshi. Predn (parallel real-time event and data driven network) : A framework for real-time distributed robotic software system. In *Proceedings of 17th Annual Conf. of the Robotics Society of Japan*, volume 3, pages 845–846, Sep 1999.
23. S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, , and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research (IJRR)*, 19(11):972–999, 2000.
24. H. Wu, Q. Chen, and M. Yachida. Face detection from color images using a fuzzy pattern matching method. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(6):557–563, June 1999.