AD-A282 697

DTIC
ELECTE
JUL 2 8 1994
S
F

Qualitative Vision and Action
Annual Report for ONR Grant N00014-93-1-0332
February 1993 to February 1994

R. James Firby and Michael J. Swain

University of Chicago
April 23, 1994

N
D
U
J:

B
D

94-12976

UNIVERSITY OF CHICAGO
DEPARTMENT OF COMPUTER SCIENCE

94 7 27 131

Qualitative Vision and Action
Annual Report for ONR Grant N00014-93-1-0332
February 1993 to February 1994

R. James Firby and Michael J. Swain

University of Chicago
April 23, 1994

For additional copies, write to:

Department of Computer Science
University of Chicago
1100 E. 58th Street
Chicago, Illinois   60637-1504
U.S.A.

Qualitative Vision and Action
Annual Report 1994

R. James Firby and Michael J. Swain

Artificial Intelligence Laboratory
Computer Science Department
University of Chicago
1100 East 58th Street
Chicago, IL 60637

Contents

Chapter 1

Introduction

The objective of the Animate Agent Project is to understand both the basic motor and visual skills required to support robot behavior in a mixed man-machine environment and the planning techniques required to combine and coordinate those skills. As mobile robots and other complex machines are deployed in the world, they will come into contact with human beings and will have to be able to function in environments designed primarily for human use. It is imperative that the machines be capable of interacting with these environments and the people within them effectively and cooperatively.

Our approach is built around a robot architecture that uses reactive planning, active perception and behavioral control. The RAP reactive execution system selects actions to execute at run time based on an agent's goals and an understanding of the immediate situation. Making action decisions after situations actually unfold ensures that the decisions are based on what is really happening and gives tremendous flexibility in the face of bad information, actuator failure, and unexpected events.

Within our architecture, actions are carried out using real-time skills. Skills are continuous control routines that can be reorganized at run time by the RAP system. All visual operators and behavioral routines are coded as such composable skills. Skills control the agent as continuous feedback loops but are constructed using discrete, symbolic instructions. This marriage of control, active sensing, and symbolic, reactive planning leads to an architecture in which the fundamental aspects of intelligent interaction with the world can be explored.

This architecture is particularly appropriate for robot control for a number of reasons. Skill sets represent a level of behavior description that is specific enough to actually implement on real hardware while remaining abstract enough to be portable across varying hardware platforms. Once a reasonable number of skills have been implemented, the ability of the system to reorganize their behavior will allow the reactive execution system to carry out an expanding number of goals without having to use an expanding number of skills.

1.1 Progress in 1993

The Animate Agent project has several different facets. There is the architecture being designed; the plans, control loops, and other representations required to implement specific robotic behaviors within that architecture; and the testing and evaluation of both the architecture and the behaviors. However, the primary goal of the project is to assemble all of these pieces into a single, integrated system to control a real robot in an ordinary indoor office environment. Building a complete system requires progress in the following areas:

- Hardware and software infrastructure.

- Basic algorithms for use in visual and behavioral skills.

- Experiments with basic algorithms.

- Generalization of basic algorithms into generic skills.

- Reactive plans for carrying out office goals using generic skills.

- Demonstrations of office behavior.

This year we have been concentrating primarily on the first three of these areas. A significant fraction of our work has gone into developing an experimental platform that integrates the entire Animate Agent architecture. While the hardware itself existed before the start of this contract, we have been developing software systems and languages that make it easier to use. In particular, we have developed the Vision Server for rapid prototyping of visual skills and the CRL system for programming, executing, and managing composable control skills. We have also adapted the RAP system to interface with the CRL system.

Concurrent with the implementation of the basic software infrastructure, we have been experimenting with vision algorithms that can be used in visual skills. Previously, we have done work in building simple behavioral skills for moving and turning the robot. These skills were designed to use visual feedback and in the last year we have been working on algorithms to give this feedback, such as motion and color-histogram tracking. We have also been exploring the use of face recognition to identify and track particular people, and we have been looking at ways for the robot to identify its current location. All of these skills together will form the building blocks for a variety of schemes for moving the robot from place to place.

1.1.1 Hardware and Software Infrastructure

The infrastructure for the Animate Agent project consists of both hardware and software. The hardware is a robot testbed that was assembled before the beginning of this grant. However, this past year our robot has gone through a significant overhaul and upgrade. The result is a mobile robot equipped with a binocular pair of cameras, sonar and infrared range sensors, an arm and jaw gripper, and on-board computers for gathering data from the sensors, controlling actuators, and executing the behavioral control routines. Vision processing is done off-board on a Datacube pipelined image processing computer with a Sun Sparc-10 workstation as host.

Images from the cameras are sent to Datacube via a tether or video transmission. The reactive control system also runs off-board on a Macintosh Quadra that communicates with the Sun via Ethernet and the robot via radio modem.

Basic Software Systems

On top of this testbed we have been implementing the software components of our architecture. The overall architecture consists of three primary software systems: the vision system, the skill control system, and the RAP reactive execution system.

The vision system manages the Datacube hardware and implements fast, active vision routines for extracting specific information about the world. To aid in building visual routines we have implemented a Vision Server to simplify interaction with the Datacube and modularize basic visual processing. In effect, the Vision Server defines a language for building visual routines. The Vision Server is described in more detail in two Animate Agent Project Working Notes [14, 13].

The skill control system implements basic actions onboard the robot such as moving while avoiding obstacles, orienting with respect to a particular object, reaching out to grasp an object, and so on. These basic actions are actually built up out of somewhat simpler control skills that use both vision routines and other sensor information as feedback. The skill control system also presents a single, unified interface between the robot's vision and control systems and the RAP execution system. Furthermore, the skill control system is spread across several different computer systems. This year we have defined the interface to this system, implemented the system itself, and defined a language for programming it. The result is CRL, a language for building control skills. CRL is described in more detail in an Animate Agent Project Working Note [8].

The RAP execution system existed as a reactive planning system before the start of this grant. However, the RAP system needed significant changes to the semantics of its interaction with the world to interface effectively with continuous visual and behavioral control processes. This year saw the design and implementation of these changes and the creation of a new, highly interactive RAP system. The new RAP system is described in University of Chicago Working Note [7]. The new semantics required for the RAP task network representation is described in a paper to appear in the Proceedings of the Second AI Planning Systems Conference [9]. More extensive papers are also in preparation.

We expect that future work will include significant refinement of all three of these systems as we gain more experience implementing actual visually-guided behaviors for the robot.

1.1.2 Visual and Behavioral Skills

In addition to implementing the basic software modules that make up the Animate Agent architecture, this year saw progress in the definition and implementation of specific visual and behavioral skills to populate that architecture. This work included two significant demonstrations of the utility of the integrated architecture in solving perceptual problems. In one case, we categorized the ways in which knowledge of the task and environment could be used to select the

appropriate tracking algorithm for a given situation, and demonstrated it in a working system that to chooses between color and motion tracking based on characteristics of the background. This work will appear in the Proceedings of the IEEE Workshop on Visual Behaviors [17].

In the other case, we showed that in a familiar man-made environment such as a grocery store, choosing an appropriate sequence of visual behaviors driven by the constraints that are known to hold in such environments can greatly facilitate searching for an object within that environment. Rather than transport the robot to the grocery store environment for the experiments, we created a virtual environment on videodisc of the store seen from a wide variety of positions and orientations. This work will also be presented at the Visual Behaviors Workshop [11].

We have been concentrating our initial efforts in implementing the basic skills needed for our robot to move around in the world. Virtually all of the ordinary office environment tasks our robot is designed to carry out involve moving from place to place. The robot must be able to find and approach particular desks, printers, and even people to carry out delivery and find and fetch tasks. The robot must also be able to follow people around to help move things. All of these navigation goals require the same set of underlying skills.

The basic building blocks for navigation tasks include moving while avoiding obstacles, orienting with respect to objects, tracking visual targets as navigation goals, and finding and identifying specific objects. This work draws on visual skills to localize the robot's position in known locales, detect obstacles, identify human faces, track motion, track color histograms, and match object boundaries.

1.2 Overview of this Report

The next section describes the two major new software modules in the system, the Vision Server and CRL, along with the RAP system changes required to interface with these modules. That section concludes with descriptions of progress made in defining specific navigation, visual tracking, and object verification routines. Finally, the report concludes with a discussion of future directions and planned research.

Chapter 2

Summary of Work in 1993-94

A considerable portion of the work done in the past year involved building the infrastructure needed for a complete robot system built along the lines of the architecture described in the first part of this summary. In one case (the new RAP interpreter) this has resulted in new research and publications. In more than one case (both for the RAP interpreter and the vision server) the code has been made publicly available and other institutions are using the code.

Another focus of our research work has been to discover how knowledge about the task and environment can simplify the perceptual problems confronting the robot. We have examined two problems: selecting a tracking algorithm given a task and an environment, and searching for objects (grocery items) in a particular environment (a grocery store).

This section of the report outlines the work done in extending the RAP system to interface with a modular, skill-based control system. It then describes the skill-based control system and vision server we have developed to support experiments on our robot. The control system, called CRL, is designed to make writing and controlling behavioral skills relatively easy, and the vision server is designed to simplify the creation of visual skills. Next, this section discusses some of the visual and behavioral skills that we are using to implement basic local navigation strategies for the robot to use in finding and approaching locations, objects, and people. Finally, this section finishes with a discussion of two experiments underway that explore the integration of a variety of visual skills into coherent, reactive plans.

2.1 The RAP System: Interfacing with Control Systems

An important part of the Animate Agent Project is interfacing the symbolic RAP reactive execution system with the CRL modular control system. The RAP system was originally developed using a simulated robot operating in a discrete-time domain in which actions were sequential and atomic. However, we are using it to assemble and coordinate reactive skills, and it has been extended to control concurrent, continuous processes. In a sense, what were primitive actions in the simulated domain will be more primitive on the real robot: instead of each action

corresponding to a goal to be achieved in the world, each is a command to enable a visual or behavioral routine.

There are two aspects of the original RAP system that make it difficult to use for controlling real robots: the assumption that control consists of sequencing atomic actions, and the assumption that only atomic actions update RAP memory.

Assuming atomic actions causes several problems. Atomic actions are not a good model for low-level robot control. It is often much easier to think of the low-level control system as a collection of concurrent, interacting processes. A robot takes a particular action by setting a subset of these processes in motion (some for sensing and some for acting) and allowing them to execute over a period of time. Within such a world view, the RAP system would be responsible for starting and stopping processes and, to a certain extent, monitoring their progress. Furthermore, real robots can often do more than one thing at a time; the RAP system should be able to pursue multiple goals concurrently as long as the hardware and control system will allow it.

The assumption that only primitive actions update memory is closely related to the assumption that the goals (and success and failure) of primitive actions are effectively independent of context. When a primitive action has a well defined goal and can detect its own success and failure, it can make changes in memory that reflect changes it makes in the world. It can also make inferences about the state of the world given particular primitive successes or failures. However, when the RAP system selects a set of concurrent processes for taking an action and detecting its success or failure, there is no notion within each process of a well-defined goal. Although the RAPs that select the processes form a hierarchy that contains subgoals at the right level of abstraction for updating memory, the processes themselves don't know what is really going on. In fact, memory is hierarchical by nature Rather than use primitive actions to try and keep the whole hierarchy consistent (as the basic RAP system does), it makes more sense for RAPs at different levels of abstraction to update memory themselves.

2.1.1   The Skill Model for Low-Level Control

Recently, AI researchers have proposed several different mechanisms for programming robots "reactively." These include collections of behaviors [2], schemas [1], routines [12], and reflexes [16]. Many details differ between these proposals, but they share the common idea that the actual behavior of the robot at any given moment is the result of a set of interacting processes acting on input from the environment. Thus, the behavior of the robot (i.e., its apparent immediate goal) can be changed by changing the set of active processes. This idea has been discussed by several authors and it allows some aspects of robot control to be described in terms of concurrent processes while other aspects are described in terms of discrete, symbolic steps that enable and disable those processes [12, 16, 6].

We have chosen to use the composable routine model embodied in CRL. However, the RAP system assumes a model broad enough to include most of the programming models described above. This general model is based on the idea of interacting processes called "skills" [19]. The programming model for skills, the way that skills are interconnected, and whether or not there is a fixed number of skills is not important. What matters is that skills can be treated as independent entities and enabled and disabled at will. Most models for robot programming

can be phrased in terms of skills by using enable and disable to either turn on or turn off a hardwired process, subsume a process, instantiate or destroy a process, or alter the parameters of a permanent process.

Given that a desired robot action is obtained by enabling an appropriate set of skills, there is still the problem of telling when a particular goal has been attained or when a situation has arisen that prevents that goal from being attained. Sometimes the RAP system will monitor and detect these situations but often a skill will be best at detecting certain states of the world, particularly those states in which the skill is not functioning properly. Also, it will often be necessary to use skills to detect transient conditions reliably because skills will be able to react much faster than the RAP system. We assume that when skills detect various conditions, either good or bad, they will send asynchronous signals to the RAP system in the form of events.

2.1.2 Using RAPs to Control Skills: Skill Groups

Getting a robot to do useful things requires moulding skills and events into coherent groups that act to achieve a specific goal. For example, a skill for moving in a given direction while avoiding obstacles might be coupled with a skill for tracking a specific color to create a group that implements a "follow the color" behavior. Skill groups define control programs that have the property (often emergent) of achieving a semantically meaningful goal. By moving from one active group of skills to another, the robot can be moved through a series of 'ow-level goals. In effect, skill groups can be used to implement discrete primitive actions.

RAPs define a hierarchy of methods for carrying out discrete (although possibly concurrent) tasks in the world. The notion of discrete tasks and subtasks matches the way task plans are usually conceived, gives the RAP methods a clear semantics, and is crucial when RAPs and RAP tasks are used as planning operators. Skill groups are a way to map discrete RAP goals onto a continuous control system. When the RAP system refines goals into a sequence of skill groups, it is dividing behavior into segments that can each be implemented with a single configuration of the control system.

There are several aspects to defining a skill group that implements a reliable subgoal for the RAP system. First, a group must implement a well-defined goal so that the group can be used meaningfully in plans. This is not an onerous requirement because the RAP system can easily use groups that achieve a goal only under some circumstances. Second, when a group's goal has been achieved, it must signal success and transition to a mode of behavior that will maintain the goal until the RAP system has time to activate the next skill group. This aspect of skill group design is the most difficult to get right. Finally, a skill group must signal when it is stuck, diverging from its designated goal, or failing in some potentially dangerous way. While the RAP system is capable of monitoring the progress of skills (by monitoring sensor information) it will typically be operating much more slowly than the skill level and can supply only loose supervision. It is important that skill groups detect success, failure and inefficiency themselves. Only when skill groups are closed in this way under both success and failure can the RAP system use them with confidence.

Given a closed skill group, the RAP system can activate the constituent skills and events, and then wait for either a success signal or a failure signal. If the group is truly closed a signal is

guaranteed. This simple notion provides the model for the basic programming construct that connects the RAP and skill systems. A construct is provided for defining primitive RAP actions that activate a group of skills and then wait for an event to be signaled. When a signal is received the primitive action is assumed to be complete, a result is returned based on the signal received, and RAP execution resumes. Most activity can be expressed with this construct. However, under some circumstances, it makes sense to activate skills individually and watch for various events and sensor readings at the RAP level. This way of organizing and representing behavior is also supported by the same constructs.

2.1.3 RAPs and Concurrency

The RAP system must support concurrent processing of the RAP task agenda to enable RAPs to activate individual skills and events and then watch for changes in the world. The initial RAP system assumed that all activity consisted of individual, sequential primitive actions and that RAP processing could stop during the execution of a primitive without harm. Within the skill model, RAPs can activate and deactivate individual skills this way, but after an entire skill group is activated it must run for some extended period of time to accomplish a task. During this time, the RAP system can neither stop execution entirely, nor forge blindly ahead expanding tasks down to skill activations. Once a required set of skills is active, the RAP system must wait for it to succeed or fail before going on to the next subtask in that method but it should continue to execute tasks in other task families.

2.1.4 RAPs and Memory

Another important issue in porting the RAP system to manage a skill-based control system is updating memory in response to RAP activity. In the initial implementation of the RAP system, all memory changes were made by the primitive action and sensor handlers; RAPs and the RAP interpreter never changed memory themselves. This approach to memory maintenance is clearly too simplistic. When primitive actions exist and have a well defined semantics in the world, it makes some sense to put memory updating down at the level of primitive action execution because the interpretation of action results is independent (to a large extent) of the task hierarchy in which the action is executed. However, when primitive actions are synthetic constructions, their semantics are known only to the task that generated them. Thus, the RAP system must allow RAP tasks to update memory in response to events and sensor data generated by the skill system. The dynamic, RAP expansion hierarchy that enabled the skills involved holds the context required to properly interpret signals from the skills.

The memory update mechanism we are using is based on RAP activation and completion. Each time that a RAP task is activated (i.e., is selected for execution the first time) or completed (i.e., removed from the task agenda), a memory update rule specific to that task type is run. The rule can examine the current contents of memory and the result of the execution of the RAP and then alter memory any way that it chooses. This mechanism allows the results of task execution to be interpreted in the correct context and at the correct level of abstraction. Memory rules can also be defined to be run when a particular event occurs during the execution of a task.

Running memory update rules on RAP task activation, completion, and event detection allows arbitrary inferences to be made in response to RAP task execution. Unfortunately, this mechanism does not support a more global view that could interpret events that take place outside of the current execution context. Truly unexpected events will always go unnoticed unless they directly affect the tasks currently underway. A wider view requires a separate system dedicated to trying to understand what's happening in the world.

2.1.5 The New RAP System

The changes proposed in this discussion have been implemented in Common Lisp as the RAP system. The system is currently controlling the University of Chicago robot via the CRL modular control system. The new RAP system has also been made available to MITRE Corporation which is using it to control one of their research robots via a Gapps and Rex based control system organized into skills [19]. It has also been made available to Johnson Space Center which is using RAPs to control both a real mobile robot and a robot simulator.

2.2 Software Infrastructure

A considerable portion of the work done in the past year was to build the infrastructure needed for a complete robot system built along the lines of the Animate Agent Architecture. In particular, we have needed to design and implement a control system consisting of composable processes and a vision server that supports multiple independent visual routines.

We have also designed and implemented a language for writing behavioral control routines. Composable routines are required to make the behavioral control system flexible enough to generate the wide variety of behaviors an agent requires in the real world. However, unrestricted composability is an ideal. If routines were all self-contained composable units then the behavioral control system could be completely reorganized at any time. In practice, routines will often depend on each other and arbitrary configurations will not make sense. The control system we propose will use a network of routines within which only restricted, but sufficiently flexible, reconfigurations will be possible. The language have designed for encoding such routines can be used to control a simulated robot while debugging behaviors and can then be translated to C for running on the robot's onboard control computer.

Another important software development has been the design and implementation of a vision server that permits rapid switching between visual operators on the pipelined image processor.

2.2.1 The Chicago Robot Language

CRL (The Chicago Robot Language) is a system designed to meet three primary needs:

- CRL is a simple, modular, behavior-based control system distributed over multiple computers.

- CRL is a language for implementing modular control routines, or skills, within the CRL system.

- CRL is a foundation for more advanced explorations of modular control when that becomes necessary.

There are many different ways to implement control systems on mobile robots and it may seem like CRL is unnecessary. However, there are several aspects to the Animate Agent project that make our development of CRL expedient. First, our primary research interest is studying how behavioral skills can be modularized and then mixed and matched in various ways to achieve different tasks. Thus, we need a language for writing control routines that emphasizes modularity and a system for executing those routines that enables them to be turned on and off under a planner's control. CRL is specifically designed to support this level of abstraction.

Second, our control system must be distributed over several different machines using different languages. In particular, some of the control routines for our robot will be running on-board on a 68000 embedded controller, some will be running off-board on a Macintosh in Common Lisp, and many of the vision routines will be running on an off-board Sparc 10. CRL is designed to allow reasonably transparent distribution of routines across these machines. In particular, CRL includes translators for turning CRL routines into both Lisp and C code and the CRL interpreter moves information between systems as required by the currently active routines.

Finally, CRL is designed to make various control problems explicit so that they can be evaluated and studied. For example, a critical real-time control problem is making sure that routines run often enough to keep up with the world effectively. CRL maintains an explicit execution schedule for active routines so that we can explore different mechanisms for managing the schedule and supplying execution time guarantees. Similarly, CRL makes the explicit the notion of default routines, constraints between routines that use the same robot resources, and routine subsumption. By taking these concepts seriously in the CRL language and implementation, we hope to be able to start out with a relatively simple control system and elaborate it as the situation dictates.

Control and operating system research are not the main focus of the Animate Agent project however, we believe that CRL will be a good basis for incorporating research from those areas as it is required to make the project work.

The Skill Control Model

The basic modular control system modeled by CRL consists of *skills*, *signals*, and *channels*. Skills are control routines that can be enabled and disabled independently. Skills communicate with one another and with the robot sensors and actuators through channels. Active skills can also generate signals when significant events occur in the world.

Externally, the CRL system looks like a collection of skills that can be enabled and disabled, and that occasionally generate asynchronous signals. Channels are not currently visible outside of the system. If an external system wants to know the value of a channel, it must activate a skill that will send it a signal that includes the value of the channel.
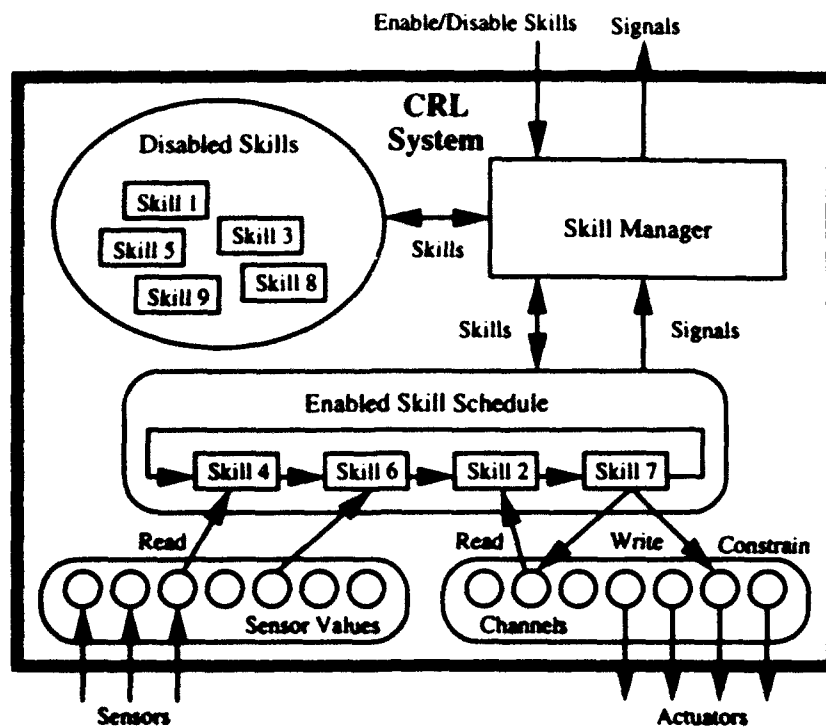
CRL System

Disabled Skills

Skill 1

Skill 5   Skill 3

Skill 9   Skill 8

Enable/Disable Skills   Signals

Skills

Skill Manager

Skills   Signals

Enabled Skill Schedule

Skill 4 → Skill 6 → Skill 2 → Skill 7

Read   Read   Write   Constrain

Sensor Values   Channels

Sensors   Actuators

Figure 2.1: The CRL System

A schematic view of the CRL system is shown in Figure 2.1. When skills are enabled, they are moved to the schedule of actve skills. As they are disabled, they are removed from the schedule. The CRL system is continually cycling through the schedule of skills running each in turn. Each skill can examine the current state of all sensors and channels and decide whether and how to alter the value of a channel. Sensor values come directly from sensors (or sensor drivers) and some channels are connected to actuators (or actuator drivers). A single skill can include a tight feedback loop from sensors to actuators. Feedback loops can be built using multiple skills that communicate through channels.

Enabling and disabling skills alters the members of the active schedule, altering the active feedback loops and hence altering the robot's behavior.

2.2.2 The Vision Server

Another important software development has been the design and implementation of a vision server that permits rapid switching between visual operators on the pipelined image processor. The server implements a number of elementary low-level visual operations at near frame rate (30 times per second), such as real-time motion detection, a multiresolution Gaussian pyramid, and color histogram creation and backprojection. These last three computations can be restricted to a region of interest at a given level of resolution, allowing the system to attend to different regions of the scene. Restricting the focus of attention can also prevent data transfer to the host from being

a bottleneck in the vision system. The other primitive operations we are planning to implement on the pipelined image processor are a Laplacian pyramid, and Freeman's directionally-selective steerable pyramid. Again, only a specific region of interest and level of resolution need be sent to the host, saving resources.

The server frees most programmers from having to learn the specialized control language for the pipelined processor. It also allows routines implementing visual operators to run on different processors, even on a different computer networked to the host. Since we plan to run a number of different operators concurrently, we will be able to take advantage of this natural parallelism to speed visual processing. The operators will sometimes give complementary information to guide the same task, such as in the use of color and motion for tracking, and other times be used for different tasks that are executed concurrently, such as avoiding obstacles using stereo cues while following a tracked object using color cues.

To minimize computational overhead, all of the DataCube routines have been compiled into a special format called a PAT (PipeOp Altering Thread). This enables us to switch between different visual operators implemented by the server within a frame time.

Why not just program the DataCube directly?

We have found the three most limiting aspects of the DataCube machines to be:

- Any process using the DataCube must be running on a single designated DataCube host machine.

- Only a single process can use the DataCube at a time.

- Pipelined programs are hard to program.

The DataCube server alleviates all three of these problems. Processes connect to the DataCube server via TCP-IP. Only the DataCube server must be running on the DataCube host. Clients can run on any machine that has a network connection with the host. This can be especially useful when either the fastest available machine is not the DataCube host, or when a machine other than the datacube host has special hardware that must be used.

The DataCube server allows multiple concurrent connections so that many routines can use the DataCube simultaneously. It is often the case that off-DataCube processing is required in addition to the processing done on the DataCube (especially in the development stages). By providing multiple processes access to the DataCube, non-pipelined processing can be done in parallel. Furthermore, since clients can connect from different machines non-pipelined processing can be distributed to many processors.

The DataCube becomes easier to program in three ways. First, most routines will not need to have any DataCube code written at all; the server will provide all the functions needed. Second, when DataCube code must be written, a C++ interface to imageflow is provided that sets up commonly used paths (from the camera to a DataCube memory, for example), automates memory allocation, and keeps track of volatile elements on the DataCube that need be reset only when some other process has corrupted them (such as look up tables). And third, a standard

12

structure is provided that simplifies the integration of multiple pipelined routines into a single program, thus allowing multiple routines to use each other's results without shipping them to the host and back. This software has been made available via anonymous ftp; researchers at Hughes Research Labs (Malibu, CA) and the University of Rochester have obtained the code and added to it for their own uses.

2.3 Skills for Simple Activities

One of the central goals of our research is to define a small set of skills that can be used in a variety of combinations to carry out a large number of everyday tasks. Our initial work in this area has been to look at skills for reliable local navigation including such tasks as finding and approaching a work area, finding and approaching a person, and following a person. These tasks form the basis for a number of more complex tasks we will address such as fetching and delivering objects and helping a person to move a collection of objects from one place to another.

In realistic environments, local navigation tasks require frequent checking of the relative position of the goal location with respect to the robot. This checking eliminates errors that build up due to dead reckoning inaccuracies, obstacles, and possible movement of the goal itself. We are particularly interested in doing this sort of goal tracking visually. As a result, all of our local navigation tasks break down into three subtasks:

1. *Acquiring the goal visually.* When given the task of moving to a goal, the robot must first identify the goal visually. This may simply require turning to the face the goal, it may require searching for the goal, or it may require traveling to intermediate goals before the final goal can be brought into view.

2. *Visually tracking the goal.* As the robot moves towards its goal, it must continually check the relative position of the goal. This tracking must be done in different ways (i.e., using different visual routines) in different circumstances.

3. *Moving while avoiding obstacles.* While the robot is keeping tracking of its goal, it must also move toward that goal. While moving it must avoid both static and moving obstacles. Currently we are doing obstacle avoidance using sonar range data.

2.3.1 Active Visual Skills

In order to support the visual acquisition and tracking of navigation goals, we have been implementing a variety of visual skills that build on the low-level algorithms incorporated into the vision server. These skills include localizing the robot within a room, locating and recognizing human faces, and real-time tracking using motion and color cues.

Robot Localization

A difficult problem faced by any robot is figuring out where it is when extrapolation from where it has been in the past fails, or would introduce too much uncertainty. In particular, we would

like our robot to be able to tell where it wakes up when it is first turned on. After determining this initial position and orientation, our robot may also have to make further periodic location checks from time to time when it determines that there is enough uncertainty in its estimate of its current location.

We are willing to accept a reasonably large error in the position estimation. We assume that robot dead reckoning will be crude at best and cannot be counted on for navigation over extended distances (more than a couple of meters). A central idea in the robot control plans we devise is that the robot must get a visual fix on targets it would like to approach or follow. Thus, all we need from a localization procedure is to get close enough to the right answer so that the robot can turn to face the place it wants to go. Knowing the robot's position within centimeters and fractions of a degree is not necessary.

We elected to use an approach to robot localization based on image signatures. Basically, an image signature is a compressed encoding of an image that can be matched against other image signatures very quickly. The robot localizes its position by taking an image, computing its image signature, and then matching that signature against a large database of images taken at known locations. The best match tells the robot where it must be.

The aim is simply to match the image seen by the robot with the images stored in the database, taken at closely spaced intervals in location and orientation. Because we need to store and match representations of a large number of images, the representation must be extremely compact. Nelson [15] developed what he termed *image signatures* as a solution to such a problem. The image is cut into an array of sub-images (8 by 8, for example), and in each subimage a function is computed to summarize its content. The summarizing function should be invariant under any different lighting conditions that are expected to occur. One good choice is dominant (*i.e.* most frequent) edge orientation. If the orientation is quantized into 8 directions, an image can be represented by 64 bytes.

We follow Engelson [5] in applying the image signature approach to robot localization. Like Engelson, we combine the result of more than one measure in each subwindow, in our case dominant edge orientation, *texturedness* and *dominant hue*. The image signature approach is based on a form of correlation matching. In the image signature approach, the spatial resolution is heavily reduced, to save storage and computation, and to make up for it, rich features are computed at each location for matching.

Locating and Recognizing Faces

One of the things we would like our robot to do is interact with human beings. Our short term goal is to be able to find and then approach or follow people. The most reliable way to identify an a particular person is to recognize the person's face. Using color information, we currently have ways of differentiating individuals, but only as long as they don't change their clothing! Faces are much less subject to change, and a significant research community has developed algorithms and code. The best algorithms work well, if not perfectly. We hope that our ability to acquire large amounts of data by observing someone over a number of seconds will allow us tolerate some error. In addition, we can influence the person's behavior, and convince him or her to look at the robot by motioning or talking to the person. In this way, we can lessen the need to recognize the face from many points of view.

The recognition method we have employed is based on the paper 'Eigenfaces for recognition' by Turk and Pentland, and the code by Turk and Starner from MIT. It consists of taking a set of images, representing faces, finding the average image, and then building the subspace of the differences using their correlation matrix and its eigensystem. This lowers the dimensionality of the search space from MxN (size of input image), to less than n-1, where n is the number images in the initial set. The original images are then projected on the so determined subspace and form the initial classes of faces. A class consists of a face vector and radius of the ball defining a neighborhood of the class. Classification is done by projecting any new difference image, obtained after subtraction from the mean face image, and then comparing it to the existing classes. If it is within the class's span, it is recognized as belonging to this class.

We have the ability to obtain many images of the subject, as we can track the face over time. This allows us to tolerate a certain number of misidentifications, but nonetheless, we do not yet have enough accuracy for our needs, and will add two improvements:

- better registration of the image and model. This can be done, for example, by locating the eyes in the image, as is done in [3].

- better learning algorithms, that can more accurately model the class of representations that correspond to the face seen over many observations.

2.3.2   Active Behavioral Skills

Along with the visual skills, our robot must also have behavioral skills for moving around in the world. Some of the necessary skills are quite simple, like turning to face a particular direction or moving the pan/tilt head. However, moving from one place to another is more complex because it must deal with continually changing obstacles.

Moving Toward a Goal

The primary skill our robot currently uses for local navigation is moving to a goal location while avoiding obstacles. To be a useful step in higher level plans, this skill must be well behaved. It must:

- Move the robot predictably towards a selected goal position.

- Move the robot quickly yet avoid collisions with both static and moving objects.

- Signal reliably when the robot gets stuck due to a blocked path, closed doors, too many moving objects in the way, etc.

The inputs to the "move-to-goal" skill are current sonar readings and a desired heading. The desired heading can be changed any time and represents the direction that other skills (such as a visual tracker) would like the robot to move. Sonar readings are taken continually by the robot hardware. A simple algorithm is used to combine these inputs to generate turning and motion velocities for the robot.
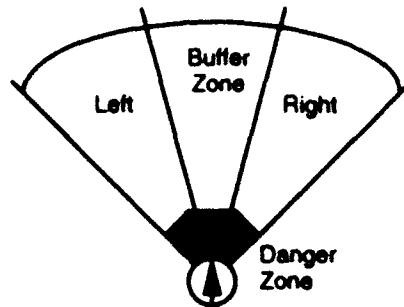
Figure 2.2: Obstacle Avoidance Zones

Sonar readings are combined to give five values: left and right danger flags, and left, right, and center blocked flags (see Figure 2.2). A danger flag is set when a sonar reading suggests that the robot is in imminent danger of colliding with an object if it continues to move in the current direction. This calculation compares each sonar reading pointing forward with a danger threshold value. If the reading is too close the appropriate danger flag is set. Blocked flags are similar to danger flags but use thresholds that are typically farther away. The blocked threshold represents a buffer distance that the robot tries to keep between itself and any objects around it. The space ahead of the robot is divided into three zones: the forward zone is 30 degrees wide centered straight ahead, and the left and right zones are 30 degrees on either side. If a sonar reading gives a reading nearer than the buffer distance in a zone, the corresponding blocked flag is set. The buffer is a parameter to the obstacle avoidance behavior.

The "move-to-goal" skill cycles continuously, calculating danger and blocked flags from the latest sonar data. It then chooses turn and motion velocities for the robot in the following manner. If both the left and right danger flags are set, then the robot cannot move to the right or the left and robot is stopped. The skill also signals that it is stuck. If only one danger flag is set, then the robot stops moving forward and begins turning away from the dangerous direction. If neither danger flag is set then the robot is free to move. However, blocked flags may indicate obstacles that need to be maneuvered around. If the front is not blocked but the left and right sides are, the robot moves forward (i.e., it needs to squeeze through a gap between obstacles). If only the sides is not blocked then the robot must turn in that direction. If only one side is blocked then the robot is free to move forward or turn towards the desired heading if it is away from the blocked side. If just the front is blocked then the robot turns in the direction nearest to the desired heading. If all three zones are blocked, then the robot stops moving forward and signals that it is stuck. If no zones are blocked then the robot turns towards the desired heading.

This algorithm for choosing rotation and motion velocities is very quick and simple and it relies only on the most recent direct sensor data. It is essentially a very robust feedback loop. It recovers from spurious sensor readings very quickly and adapts to a new desired heading as quickly as it is changed. In addition, the algorithm does not require a map, so the robot can travel through both mapped and unmapped terrain.

This algorithm is fast and robust because it looks only at directly available local data. However, it is easy for the robot to get stuck. When both danger flags or all three buffer zones are blocked, the robot does not move. This can happen if obstacles move into the robot's way or if

the robot wanders into a cul-de-sac. Getting the robot out of such difficulties requires a more global view of the world — it requires a map and a path planner. We have done some initial work on incorporating a map into the robot's local navigation (see [10]) and this will be a major direction in future work.

Monitoring Progress

An additional skill is required to make sure that the overall behavior of the robot is well behaved while it is controlled by the "move-to-goal" skill. The obstacle avoidance process will signal reliably that it is stuck when its path is blocked. However, there are situations when the robot might wander very far from its goal, or when it might wander back and forth between two cul-de-sacs without ever making progress.

The "monitor-progress" skill watches for these situations, and any other situation that might crop up to confuse the robot, using two simple heuristics. First, the monitor keeps track of the robot's closest approach to its goal. If a significant period of time passes without getting closer, then the monitor signals that no progress is being made. Second, the monitor keeps track of the difference between the robot's current heading and the direction to the goal. If the current heading deviates too far from the goal direction, then the monitor signals that no progress is being made.

The progress monitor serves to guarantee that the robot always notices when things are not working out. Without a reliable signal of potential failure, local navigation routines cannot be used as reliable actions in a plan.

2.4 Matching the Task to the Environment

One of the focal points of our research has been to discover how knowledge about the task and environment can simplify the perceptual problems confronting the robot. We have examined two problems: selecting a tracking algorithm, given a task and an environment, and searching for objects (grocery items) in a particular environment (a grocery store).

2.4.1 Task and Environment-Sensitive Tracking

The utility of different cues for distinguishing an object from its background, and therefore as the basis for tracking, can vary tremendously. For example:

1. Motion cues are more effective from a stationary platform than one that is translating or rotating.

2. Color cues are insensitive to platform motion.

3. The reliability of color cues can be predetermined by comparing the model to the background.

Image-tracking Algorithm	Unacceptable Conditions	Favorable Conditions
Motion	background motion∗ motionless target• self-rotation•	target always moving•
Color	false positives∗ target not in scene• no way to acquire model•	individual target model•
Correlation	changes in view• fast-moving target∗•	still target•
Zero-disparity	can't verge on target∗ busy background• false positives∗	

Table 2.1: Favorable and unacceptable conditions for image tracking algorithms. Conditions that can be determined without testing are marked with •; those that are measured are marked with ∗.

4. Motion cues are useful for animate objects, since they are rarely completely still.

5. If tracking a small stationary object from a moving platform, larger proximal objects can be used as landmarks.

6. Stereo cues are more reliable if the target is well textured or forms a sharp boundary with the background and the background does not contain too many confusing edges.

Applying this knowledge varies in difficulty. For example, in a mobile robot, it is easy for the system to keep track of the motion commands sent to the base and, subject to some uncertainty in the finishing times of the actions, know if it is undergoing self-propelled motion or not. The target may be described to the robot as a person, for instance; so the robot will know the target will often moving. Landmarks to head for in a homing task, such as those described in item 5, are usually known to be stable (for example, a pop can on a desk), so we can select an image-tracking mechanism that might be inappropriate for moving or changing targets. Since this sort of knowledge is so easy to apply, it is extremely useful. However it is also useful to make the choice of tracking algorithm based on information sensed about the target and background. In this case, which tests to make, and how to interpret the results of the tests, is more problematic.

Our solution is to label each algorithm with a set of *unacceptable* conditions along one or more dimensions, and with another set of *favorable* conditions (see Table 2.1).

When the robot is given a tracking task and target, it considers each algorithm available to it. If the conditions for any algorithm are unacceptable, either on the basis of prior knowledge or measured characteristics, that algorithm is not used. Of the remaining algorithms, the one with the most favorable conditions satisfied is selected.

We have successfully demonstrated how examination of the background can be used to choose between a color tracking algorithm and a motion tracking algorithm, as is shown below.
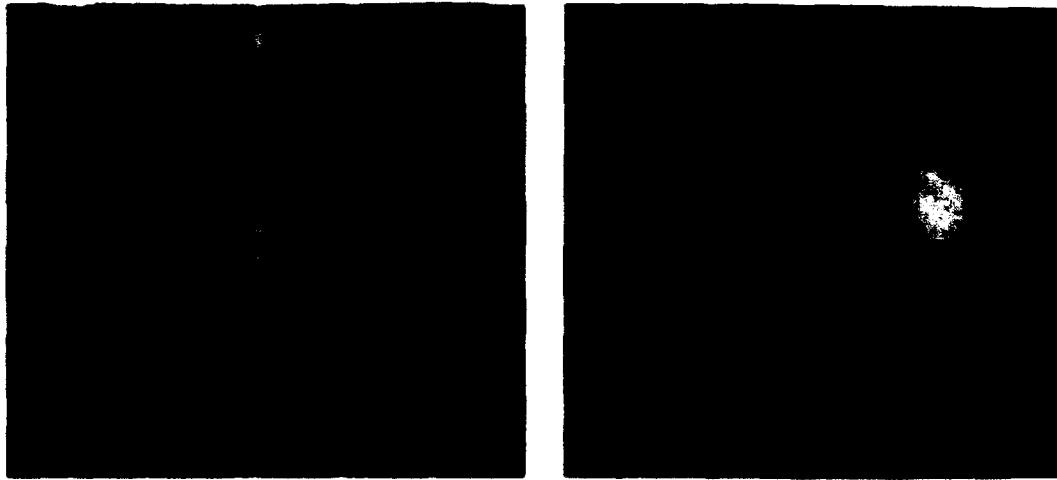
Figure 2.3: Left: A portion of the 360 degree background evaluated for responses to target colors. Right: Strong background responses rule out the use of color tracking for this target (Roger).

Demonstration

Use of a decision rule is illustrated in this example: The tracker incorporates color and motion tracking algorithms, and has a color histogram model of the model it wishes to track. Because the robot is not moving and the object to track is known to be a person, both color and motion are considered viable tracking options. The background distractor test is applied for color tracking only, because the robot had prior knowledge that only one person would be moving in the room. Not knowing this, the robot would scan the room for background motion. The color test consists of matching the colors of the target over a 360 degree sweep of the room; if there are any locations that receive a response higher than a threshold, then color tracking is disqualified. Of course, this test as stated must be used when the object to be tracked is not present. If it is possible the model is present, the tracker must apply a verification algorithm to determine whether or not the peak corresponds to the object.

There are two people to be tracked; one (Dan) is wearing a purple jacket, the other (Roger) is wearing a white shirt and beige pants that blend with the white walls in the background. In figure 2.3, the color tracker produces strong responses to the background when checking for Roger's colors, and is rejected on that basis. The result of motion tracking is shown in figure 2.4 (left). When the background was searched for Dan's colors, there was essentially no response (not shown). Figure 2.4 (right) shows the tracker using the color algorithm to track Dan against the clean background.

2.4.2 Vision in Man-Made Environments

People are able to act efficiently in environments like grocery stores, libraries, and other man-made domains. They do so because they have useful knowledge about the way these domains are organized. In this project we have shown that an everyday domain, a grocery store, exhibits
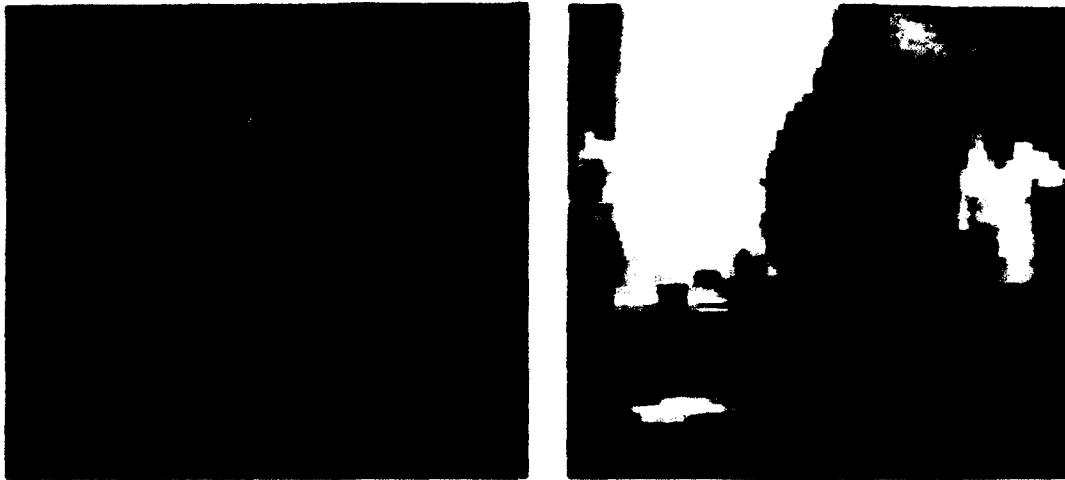
Figure 2.4: Left: The robot tracks Roger by motion, after deciding that his colors blend in with the current background. The marker, near the middle of the shirt, is unfortunately barely visible here. Right: The robot uses color to track Dan, whose coat was determined to be distinct from the background.

useful regularities an agent can use both to simplify visual tasks and to perform efficiently. We do this by first identifying the organization principles around which grocery stores are structured, identifying the visual routines involved, and using these principles and routines to find items in GROCERYWORLD, a simulated grocery store.

Grocery store shopping is a common task everyone does at least occasionally. Since everybody is able to accomplish their shopping needs fairly quickly, we are interested in what functional knowledge a planner needs in order to shop as efficiently as possible. Since all grocery store managers presumably want customers to find items without much trouble, they place and index items in some consistent manner. They do so according to the features they deem the most functional in terms of satisfying their customers' needs, and their own needs for selling as much food as possible.

A customer intending to leave in a reasonable time has to know how his food will be used. For example, a customer who wants to bake a cake might need cake mix and cake frosting. He'll find the cake frosting near the cake mixes. He also might find the cake mixes near the flour, sugar, and baking tins. This arrangement is anything but accidental: it's *intentional*. The cake mixes, as well as the rest of the items in the store, are indexed according to the features most useful in serving the needs of customers and stores.

To study perception and action in such environments, we have built the SHOPPER system, an integrated system for the task of grocery store shopping. The SHOPPER agent works in a simulated grocery store called GROCERYWORLD. In designing the simulator, we wanted to build a world which offers the same challenges and opportunities as a real grocery store. On the other hand, we wished to be able to study these issues before the robot was capable of such activity, and without the difficulties of transporting it to the domain whenever we wished to run an experiment.

The GROCERYWORLD simulator satisfies these design criteria. By using video footage from an actual store, we are able to base our simulator on real images of a grocery store. The simulator is complete in that the entire store (excluding checkout counter areas) is modeled by the simulator. Any object in the image database is accessible by moving through the world.

Regularities in grocery stores

Because a moderately-sized grocery store can stock at least 10,000 items, grocery stores need to organize their food items in consistent ways so customers can easily find them. In this section we illustrate the different ways in which stores organize their goods. Below we list the regularities we have identified so far:

- **Type**
  Items that either serve nearly the same function are nearby each other. This is a most basic organization principle under which many items fall under; e.g. McIntosh apples are near Rome apples; a jar of Gerber baby food will be found with other baby foods; a tomato clustered with other vegetables; an apple placed with other fruits; coffee is near tea.

- **Brand**
  Within a section of a specific type, the makers of the food will also be clustered together. For example, in a typical grocery store aisle, soups of the same brand (e.g. Campbell's, Progresso) will be clustered with each other no matter how similar a soup is to another brand's. Campbell's vegetable soup is not placed adjacent to Progresso vegetable soup.

- **Counterparts**
  Items that complement each other are placed together. For example, salad and salad dressing, pancakes and maple syrup, pasta and tomato sauce, etc.

- **Physical Constraints**
  Perishable or bulky items that require special storage considerations like orange juice, eggs, and frozen entrees, are placed together.

- **Ethnic foods**
  For items commonly associated with other countries or cultures (e.g. soy sauce, curry, matzah, water cress) tend to be placed nearby each other in an "ethnic" section.

- **Packaging**
  Bulk items such as bags of oranges, apples, and potatoes will be placed separate from their individual versions.

In addition, perception relies on some simple assumptions we make about the domain:

- Items are placed on shelves.

- Items are displayed in a consistent manner: e.g. cereal boxes are upright with the front of the box facing outward.

Using a specialized shelf detector, color histogram matching, and an edge-matching routine for verification, Shopper is able to effectively locate items in its simulated environment.

Chapter 3

Future Plans

This last year has been primarily a year of infrastructure building and consolidating research on the RAP system and visual skills so that they can be used in a single system.

Next year we plan to focus on developing a robust set of visual and behavioral skills within our architecture. Along with these skills we will be writing RAPs to use the skills in a variety of demonstrations. These plans are described below.

3.1 New Skills and RAPs

Our strategy for the near future is to devise a vocabulary of visual and behavioral skills and RAPs to solve a handful of tasks that we have devised for our robot around the lab, including fetching tools, and moving objects from one place to another (including giving the object to another person) with initial guidance from a human. Using our experience with these tasks we hope to produce an integrated system that can perform all the tasks and be readily reprogrammed to perform new tasks.

Integrated Visual Tracking

We will encapsulate the work on using multiple visual tracking skills into an integrated visual tracking skill. Tracking visual targets is a skill needed for local navigation, hand-eye coordination, and a host of other robot tasks. As mentioned previously, we have been looking at ways to make tracking more robust by switching between tracking cues as the situation changes around the robot and target. Most of the information required in making the decision to switch cues can be derived directly from the visual image and a handfull of object properties.

We intend to build an integrated visual tracking skill that takes a target's relevant properties as input and then tracks that target through different situations, switching between cues as necessary, without further input. This integrated skill will form one of the key visual skills required for everyday activity.

Stereo for Segmentation and Hand-Eye Coordination

We have obtained the NEC stereo algorithm by I. J. Cox et al and have successfully used it for segmenting objects (such as people) from their background. We plan to integrate it into skills for finding people, interpreting pointing gestures (see below) and eye-hand coordination.

Free space detection

We are implementing a visual operator for detecting "free space", or regions that the robot can freely travel without bumping into obstacles, based on the work of Ian Horswill. This operator relies on the assumption that the floor is relatively free of markings (fine texture as in carpets is ignored), and that obstacles do not protrude beyond their "footprint" on the floor. This operator runs quickly enough that it can be part of a control-loop, preventing the robot from running into obstacles as it navigates through the environment. The visual information complements sonar sensing, which can miss low or narrow obstacles, suffer from reflections, and is extremely low resolution.

Intermediate Local Spatial Representation

The simple obstacle avoidance algorithm we are using in the robot's "move-to-goal" skill works reasonably well in uncluttered environments but the robot often gets stuck in cul-de-sac's in more complex environments. The only way around this problem is to have some sort of memory of where the robot has been before and where obstacles have been seen before. The usual way to keep such a memory is as a map. Unfortunately, using a map is a difficult problem. A map of obstacles usually contains obsolete information and can easily lead the robot astray. Also, a map usually has to be used as a static resource; when the robot gets stuck it consults the map and builds a plan to go around obstacles. Following the plan then becomes the next task. When the map is inaccurate, following such a plan can lead to very strange behavior because the robot does not reconsider its plan as it encounters missing or new obstacles.

We are looking at ways to build a local object representation that can be used both to build maps and as part of feedback control loops. We will be building on the Navigation Template idea of Marc Slack [18]. As the robot moves towards its goal, it will incrementally build up representations of obstacles in its local environment and decide on the best way to get around each obstacle as it is encountered. The "move-to-goal" skill will look at this advice while directing the robot. When the robot needs to build a plan, the plan will consist of choosing new ways to get around all the recorded obstacles. If an obstacle is found to be missing in the world, its advice can be ignored.

Hand Eye Coordination

We are also beginning to look at the skills needed for picking things up off the floor. In particular, we are looking at adaptive control algorithms for moving the arm and hand, visual skills for determining the relationship between the hand and the object to be grasped, and skills for

actually grasping the object. The grasping skills will be quite simple because our robot's hand is not very complex. We hope that we can draw on the stereo and tracking visual skills already under development to give the visual feedback the arm motion and grasp skills will require.

3.2 Acquiring Knowledge about the Environment

Over the past year, one of our students, Charles Earl, has been working on an unrelated project looking at ways to learn new causal knowledge while engaged in routine activity. This work has been based on a connectionist model proposed by Drescher [4] to explain some of the early stages of human learning described by Piaget. However, while exploring this work we uncovered compelling similarities in concept between many of the structures formed in the connectionist network and the way plan fragments are represented by RAPs.

We have decided to explore these similarities by moving out of the connectionist framework and applying the relevant learning ideas directly to the RAP system and hence to the overall Animate Agent Project. Basically, the learning algorithm attempts to find correlations between goals, actions, and outcomes. The robot should be able to learn that when asked to come to office 159 it will usually have to pick up the trash so it should bring along a trash bag. Eventually, the robot should also be able to learn some of the selection criteria for visual tracking routines. For example, it might learn that motion tracking is unreliable out in the hallway even though it works well in the office.

3.3 Integrated Demonstrations

We will be entering the robot in the AAAI robot competition in July '94. The competition consists of two tasks: an "office delivery" task and a "trash collection" task. The object of the office delivery task is to navigate to a known location through a known office-like environment in which obstacles will be placed and doors will open and close. In the trash collection task the robot will collect soda cans, styrofoam cups and crumpled pieces of paper distributed in unknown locations around another office-like environment. We will compete against a number of teams from other universities for points assigned by judges, based on criteria agreed on before the competition.

To provide an integrated demonstration of the capabilities of our robot and the software systems we are developing, we are aiming for the following demonstration, for fall 1994. In the demo, the robot will follow someone (its "master") to a location chosen by the master, pick up the object he points to, and follow him back to the starting point. The robot would then return to the chosen location as many times as necessary, pick up any other objects grouped next to the original object, and return them to the starting location.

Bibliography

[1] Ronald C. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *International Conference on Robotics and Automation*, Raleigh, NC, March 1987. IEEE.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[3] R. Brunelli and T. Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1042–1052, 1993.

[4] Gary Drescher. *Made Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, Cambridge, MA, 1991.

[5] Sean P. Engelson and Drew V. McDermott. Image signatures for place recognition and map construction. In *SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion IV*, 1991.

[6] R. James Firby. Building symbolic primitives with continuous control routines. In *First International Conference on AI Planning Systems*, College Park, MD, June 1992.

[7] R. James Firby. Interfacing the rap system to real-time control. Animate Agent Project Working Note AAP-1 Version 1, University of Chicago, July 1993.

[8] R. James Firby. The crl manual. Animate Agent Project Working Note AAP-3 Version 1, University of Chicago, April 1994.

[9] R. James Firby. Task networks for controlling continuous processes. In *Second International Conference on AI Planning Systems*, Chicago, IL, June 1994.

[10] R. James Firby, David Christianson, and Tom McDougal. Fast local mapping to support navigation and object localization. In *Sensor Fusion V*, Boston, MA, November 1992. SPIE.

[11] Daniel Fu, Kristian J. Hammond, and Michael J. Swain. Vision in man-made environments: Looking for syrup in all the right places. In *Proceedings of the IAPR/IEEE Workshop on Visual Behaviors*, To appear, 1994.

[12] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Tenth National Conference on Artificial Intelligence*, San Jose, CA, July 1992. AAAI.

[13] Roger E. Kahn, R. James Firby, and Michael J. Swain. The message hub. Animate Agent Project Working Note AAP-4 Version 1, University of Chicago, April 1994.

[14] Roger E. Kahn, Michael J. Swain, and R. James Firby. The datacube server. Animate Agent Project Working Note AAP-2 Version 1, University of Chicago, November 1993.

[15] Randal C. Nelson. Visual homing using an associative memory. In *Proceedings of the DARPA Image Understanding Workshop*, pages 245–262, 1989.

[16] D.W. Payton. An architecture for reflexive autonomous vehicle control. In *International Conference on Robotics and Automation*, San Francisco, CA, 1986. IEEE.

[17] Peter Prokopowicz, Michael Swain, and Roger Kahn. Task and environment-sensitive tracking. In *Proceedings of the IAPR/IEEE Workshop on Visual Behaviors*, To appear, 1994.

[18] Marc G. Slack. Situationally driven local navigation for mobile robots. Technical Report JPL Publication 90-17, Jet Propulsion Laboratory, April 1990.

[19] M.G. Slack. Sequencing formally defined reactions for robotic activity: Integrating raps and gapps. In *Vol. 1828 Sensor Fusion V: Simple Sensing for Complex Action*, Boston, MA, November 1992. SPIE.

Appendix A

Summary of Progress Measures

A.1 Productivity Measures

Refereed papers submitted but not yet published: 2
Refereed papers published: 1
Unrefereed reports and articles: 6
Books or parts thereof submitted but not yet published: 0
Books or parts thereof published: 0
Patents filed but not yet granted: 0
Patents granted: 0
Invited presentations: 3
Contributed presentations: 1
Honors received: 3

- Dr. Swain was Guest Editor for the International Journal of Computer Vision special issues on Active Vision.

- Dr. Swain was a member of the Program Committee for the 1993 IEEE Conference on Computer Vision and Pattern Recognition

- Dr. Swain was the organizer of the CVPR Color Vision Symposium at CVPR '93

- Dr. Firby was a member of the Program Committee for the 1994 Conference of the American Association of Artificial Intelligence

- Dr. Swain was a member of the Program Committee for the IEEE Workshop

Prizes or awards received: 0
Promotions obtained: 0
Graduate students supported (more than 25% of full time): 2
Post-docs supported (more than 25% of full time): 1
Minorities supported: 0

A.2 List of Publications

Promising Directions in Active Vision, Michael Swain and Marcus Stricker, eds, *International Journal of Computer Vision*, IJCV 11, 1993, 109-126.

Task Networks for Controlling Continuous Processes, R. James Firby, To appear in *Proceedings of the Second International Conference on AI Planning Systems*, Chicago IL, June 1994.

The Capacity of Color Histogram Indexing, Markus A. Stricker and Michael J. Swain, To appear in it Proceedings of the IEEE Conference of Computer Vision and Pattern Recognition, Summer 1994.

An Architecture for a Synthetic Vacuum Cleaner, R. James Firby, In *Proceedings of 1993 AAAI Fall Symposium Series Workshop on Instantiating Real-World Agents*, Raleigh NC, October 1993.

Task and Environment Sensitive Tracking, Peter Prokopowicz, Michael Swain, and Roger Kahn, To appear in *Proceedings of the IEEE Workshop on Visual Behavior*, Summner 1994. Earlier version in *Proceedings of the Sixth International Symposium on Robotics Research*, ISRR-93, Philadelpia PA, October 1993.

Vision and Navigation in Man-Made Environments: Looking for Syrup in all the Right Places, Daniel D. Fu, Kristian J. Hammond, and Michael J. Swain, To appear in *Proceedings of the IEEE Workshop on Visual Behavior*, Summner 1994.

Interfacing the RAP System to Real-Time Control, R. James Firby *Animate Agent Project Working Note AAP-1 Version 1*, University of Chicago, July 1993.

The Vision Server, Roger E. Kahn, Michael J. Swain, and R. James Firby, *Animate Agent Project Working Note AAP-2 Version 1*, University of Chicago, November 1993.

A C++ Interface to the Khoros Visualization File Format, Roger E. Kahn and Michael J. Swain, *Animate Agent Project Working Note AAP-5 Version 1*, University of Chicago, April 1994.

A.3 Transitions and DoD Interactions

There has been a limited amount of technology transfer involved with the Animate Agent Project. In particular, the RAP system has been ported to Common Lisp and is being used at Northwestern University as a basis for research on opportunism. Also, the integrated architecture proposed is being implemented at MITRE Corporation for use on their mobile robot testbed using a different skill manager. Johnson Space Center is also currently using the RAP system to control two real robots and a robot simulator. The RAP system is available for wider distribution via anonymous FTP (contact Dr. Firby, firby@cs.uchicago.edu, (312) 702-6209).

The Vision Server software has also been made available via anonymous ftp. Researchers at Hughes Research Labs (Malibu, CA) and the University of Rochester have obtained the code and added to it for their own uses. (contact Dr. Swain, swain@cs.uchicago.edu, (312) 702-3495).

A.4 Software and Hardware Prototypes

There are two software prototypes that we plan to make available to other researchers as a result of this work. The first is the RAP system that will be made available over the Internet this fall. The release will include the original RAP system along the the Truckworld robot simulator testbed. The new RAP system for controlling continuous processes should become available next year.

The second software system we hope to make available over the Internet is the vision server. This piece of software is currently undergoing extensive testing and documentation. It should be available for other researchers to use within a year.

There are currently no plans to commercialize any of the software being developed for the Animate Agent Project.