A Cooperative Agent Architecture
Annual Report for ARPA/ONR Grant N00014-93-1-1394
November 1993 to November 1994

R. James Firby and Michael J. Swain

University of Chicago
January 19, 1995

For additional copies, write to:

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

A Cooperative Agent Architecture
Annual Report for ARPA/ONR Grant N00014-93-1-1394
November 1993 to November 1994

R. James Firby and Michael J. Swain

University of Chicago
January 19, 1995

UNIVERSITY OF CHICAGO

DEPARTMENT OF COMPUTER SCIENCE

Principle Investigators:	**R. James Firby and Michael J. Swain**
PI Institution:	**University of Chicago**
PI Phone Number:	**(312) 702-6209**
PI E-mail Address:	**firby@cs.uchicago.edu**
Grant Title:	**A Cooperative Agent Architecture**
Grant Number:	**N00014-93-1-1394**
Reporting Period:	**1 Nov 93 - 31 Oct 94**

A Cooperative Agent Architecture
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

Executive Summary

Our goal is to develop an understanding of how an intelligent robotic agent should represent and execute plans for everyday tasks that involve cooperative interaction with human beings. The work is being done in the larger context of the Animate Agent Project at the University of Chicago. The primary goal of this project is to implement intelligent robotic agents using a software architecture, vision processing system, and task control language, that support writing clear, concise plans for carrying out tasks in ordinary, unaltered environments.

The system uses two primary sources of knowledge to describe ways to achieve goals: a library of modular skills and a library of higher level reactive plans called RAPs. Skills are meant to capture those aspects of sensing and action tasks that are best described in terms of continuous feedback loops. RAPs are meant to capture those aspects of task execution that are best described as symbolic plans.

We believe that the key to building intelligent agents is understanding how to structure and represent behavior in a uniform manner so that the skills and plans used to carry out one goal can also be used to achieve other, similar goals. Such modularity will allow rapid reprogramming of an agent to carry out new tasks, addition of new capabilities that can be combined with the old, and plans that can be learned from a teacher or through experience.

A major accomplishment this year was developing and refining the skills and RAPs for a trash collection task. This task was motivated by the AAAI Robot Competition and we have refined the plans involved to the point where the robot has thrown away hundreds of pieces of trash. From this task we gained experience at writing reusable skills and RAPs, and learned important lessons about: the usefulness of memory in a reactive system, ways to represent of plans to ensure the tracking of indexically-functionally indicated objects across plan steps, and the limitations imposed by strict task-motivated sensing. Our entry to the robot competition received a great deal of positive attention.

Throughout the year, we also developed a variety of visual skills for finding and tracking objects and people and worked on technology to enable communication with people involved in cooperative tasks with the robot. Work was also put into improving the robot hardware and major pieces of our software infrastructure: RAPs, CRL, Datacube Server. The Datacube Server was released for public use.

Contents

Chapter 1

Introduction

The work supported by this grant requires developing an understanding of how an intelligent robotic agent should represent and execute plans for everyday tasks. In particular, we are interested in tasks that include cooperative interaction with human beings. The work is being done in the larger context of the Animate Agent Project at the University of Chicago. The primary goal of this project is to implement intelligent robotic agents using a software architecture, vision processing system, and task control language, that support writing clear, concise plans for carrying out tasks in ordinary, unaltered environments.

We believe that the key to building intelligent agents is understanding how to structure and represent behavior in a uniform manner so that the skills and plans used to carry out one goal can also be used to achieve other, similar goals. Such modularity will allow:

- *Rapid reprogramming of an agent to carry out new tasks.* In the short term, robotic systems will have to reprogrammed to carry out new goals. A behavior representation that provides an appropriate hierarchy of modular task plans will promote the reusability of existing plans as the building blocks of new plans.
- *Additive task programming.* An important goal of the Animate Agent Project is to make task programming additive so that when a robot is programmed for a new task, it will continue to be able to achieve all of its old goals as well. In other words, the robot should become more and more capable over time. Such additivity will be a natural consequence of sub-plan reuse when task plans are represented appropriately.
- *Learnable plans.* Eventually, a modular, hierarchy of task plans should promote teaching the robot new plans, through natural language and/or other instructional interaction.

Building agents that can interact effectively with human users also puts constraints on robotic behavior representations. In particular, interacting with human users means acting in understandable ways and perceiving and representing the world in ways that make communication possible. When programming a single task into a robot, representing the world can often be finessed but, when people are interacting with the robot by pointing at arbitrary objects or giving spoken instructions, the robot must perceive and represent the objects involved. Representing the world in a way that is meaningful to both people and the robot requires breaking behavior down into perceptual skills and task plans that generate and use information about the world at a shared level of abstraction.

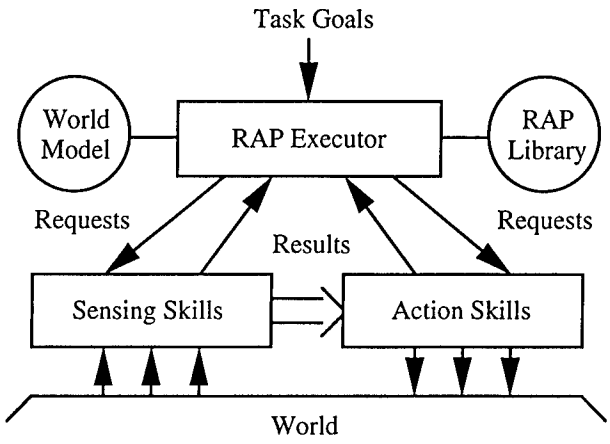

Figure 1.1: The Animate Agent Architecture

1.1 The Animate Agent Project

The first step in the Animate Agent Project has been to develop a software architecture for integrating vision and action at both the level of real-time vision guided feedback control, and at the level of symbolic task planning. The primary goal in developing this integrated architecture is to allow the skills and plans involved in intelligent behavior to be represented at the appropriate level of abstraction. Our current goals do not focus on implementing optimal control systems, but rather on exploring ways that control systems might be organized to facilitate their use in a wide variety of different settings to achieve a variety of different goals.

The organization of the Animate Agent Architecture is shown schematically in Figure 1.1. The system uses two primary sources of knowledge to describe ways to achieve goals: a library of modular skills and a library of RAPs. Skills are meant to capture those aspects of sensing and action tasks that are best described in terms of continuous feedback loops. RAPs are meant to capture those aspects of task execution that are best described as symbolic plans.

1.1.1 Skills

The skill level is a network of sensing and action processes, or skills. Each skill is defined by a separate program and can be enabled or disabled at will. When a skill is enabled it is sent a set of parameters and begins to run. All running skills can access current sensor values and values generated on global *channels* by other skills [10]. Skills can also set actuator and channel values or constrain the values actuators and channels can take on. When a skill is disabled, it does nothing.

Skills are typically enabled in groups that form coherent sets of processes that work together to carry out specific actions in the world. Skills are programmed to generate asynchronous *signals* when they detect that a task is complete or some problem has occurred. These signals are caught by the RAP system and tell it when to move on to the next step in a plan.

1.1.2 RAPs

The RAP system is essentially a reactive plan interpreter. It takes a set of task goals as input (either from a planner, or in the case of our experiments as top-level goals) and refines each goal hierarchically until primitive actions are reached. The hierarchical plans that define these expansions are the RAPs that reside in the RAP library.

RAP goals are maintained on an agenda of active tasks. The interpreter pursues the tasks on the agenda according to the following algorithm:

1. Select the next task to run from the agenda.
2. If the task is satisfied, as defined in the RAP succeed clause, go on to the next goal.
3. If the task is a primitive, execute it and go on to the next goal.
4. If the task is not satisfied and non-primitive, select a refinement method that matches the current situation, as defined in the RAP method context clauses.
5. Instantiate the method by creating a subtask for every step and placing them on the agenda with appropriate ordering constraints.
6. Place the current task back on the agenda to run after all of its subtasks have completed.
7. Loop back to select the next task.

This looping continues as long as there is a task on the agenda that is eligible to run. Tasks may become ineligible to run temporarily because they are blocked waiting for signals from the skill system or because they are constrained to follow tasks that are waiting for signals.

The RAP system interacts with the skill system by using primitive steps that enable and disable skills, and by waiting for signals that are generated by enabled skills. RAPs are usually structured to enable a meaningful set of skills and then wait for that set of skills to signal successful or unsuccessful completion. More detail on the RAP system can be found in [6, 7].

1.2 Progress in 1994

The last year has seen significant advances in our understanding of the representation issues involved in writing cooperative task plans within the ANimate Agent Architecture. This understanding has come from work in three areas:

1. *Technologies and experience in representing skills and plans for everyday tasks.* By actually implementing the behavior required to carry out tasks in the world, we have been developing insight into the best ways to abstract and modularize control processes and symbolic task control plans.
2. *Technologies for enabling cooperative activity with human users.* We have also been looking at the specific problems involved in coordinating behavior with human users.
3. *Robot testbed development.* We are using a robot at the University of Chicago to implement task plans, test our ideas for representing behavior, and learn from our experiences.

Experience in Representing Skills and Plans

During the year covered by this report, we implemented the skills and RAPs for a variety of simple tasks. For example, the robot was programmed to track and follow a human around the lab, to navigate down corridors and look for door signs, and to clean trash up off the floor.

The first two of these tasks were used as test cases for the basic architecture and resulted in:

- A visual skill for tracking a moving person using an integrated algorithm that switches between color histogram and motion tracking.
- A visual skill for finding freespace around the robot so that is can follow corridors without hitting obstacles.
- A visual skill for finding and reading simple office door signs.

Along with these test cases, a major thrust was to implement skills and plans for the trash collection task at the 1994 AAAI robot competition. These plans built upon the action skills for following objects with the pan/tilt head and moving while avoiding obstacles that were implemented for the tracking and navigation tests. It also required implementing new visual skills for finding and tracking objects, action skills for manipulating small objects, and RAPs for searching for objects, picking objects up, and for dropping objects inside bins.

Experience with refining the trash plans to produce robust, general purpose, reusable skills and subtasks taught us several lessons that are beginning to change the plan representations that we use. In particular, we learned:

- The value of memory and ways to use it effectively for remembering object locations.
- The need for two types of visual routines: those specialized for finding objects in the world, and those specialized for tracking them.
- The value of using indexical-functional object reference plans and skills.
- The need to leave sensing plan steps as unordered as possible so they can execute over long periods of time to detect objects opportunistically.

Enabling Cooperative Activity with Human Users

Along with implementing tasks to test our architecture and plan representations, we have been working on concurrent projects to develop the technology required to enable the construction of skills and plans that interact with human users. In particular, we have been looking at algorithms for detecting and recognizing people, techniques for watching gestures, and forging links with projects incorporating natural language understanding. This work is creating the infrastructure on which we will build cooperative task plans.

Robot Testbed Development

The robot and the Animate Agent Architecture require a substantial software and hardware infrastructure. This year we improved all the key components of this infrastructure: the robot, the vision server, the skill language environment (CRL), and the reactive execution system (RAPs).

Accomplishments

This work led to the following concrete accomplishments:

- Competed in AAAI Robot Competition. Placed 5th of 12 teams in the trash collection event. We were the only team to actually pick a piece of trash off the floor and throw it in the garbage can (the others took penalty points for unimplemented behaviors but gained points by moving faster).
- Developed and refined the trash collection task to the point that it has thrown away hundreds of pieces of trash.
- Demonstrated real-time person tracking and on-the-fly selection of algorithm appropriate to background (color or motion).
- Improved robot hardware and major pieces of software infrastructure: RAPs, CRL, Datacube Server, added re-usable visual routines for finding and tracking objects and people. The Datacube Server was released for public use.

1.3 Overview of Report

The next section of this report discusses the progress made during the last year in more detail. It begins with a detailed outline of the plan representations used in the trash cleanup task to set the context for later sections. That section is followed by a discussion of the major lessons learned about behavior representation based on the trash cleanup task. In particular, even this simple task has put pressure on our desire for modular plans and has had a significant influence on the way we represent visual skills. Next, the report discusses the projects that began this year to develop the algorithms needed to implement cooperative plans and a short section outlines changes made to the robot testbed over the year. Finally, the report concludes with a discussion of future directions and planned research.

Chapter 2

Summary of Work in 1993-94

A significant amount of the work done this year was directed at gaining experience in representing skills and plans for real world tasks. We believe that programming the robot is the only practical way to debug the Animate Agent Architecture implementation and gain experience using it in realistic settings. Experience is needed to ensure that nothing has been overlooked and our system is capable of addressing real tasks in real environments.

We began with what we thought would be a particularly simple task, picking trash up off the floor. The initial motivation for this task was to test the overall architecture, the skill and RAP representation languages, and the vision processing system. We wanted to make sure that skills and RAPs could be written to achieve real goals in the way we had envisioned. We also wanted to compare our system to others implementing the same task at the AAAI-94 Robot Competition.

The visual routine for finding and classifying items of trash on the floor are quite involved and took a few months to complete. However, once that visual skill was finished, the rest of the task went together in about two weeks. The ideas of modularity and hierarchical task control implemented in the architecture resulted in a system that was both easy to write and that worked reasonably well for picking up trash. However, we found that the representation decisions we had made in building up the skills and task control plans had significantly more effect on the generality of the robot's behavior than we had originally expected. It has required considerable effort to make the skills and RAPs robust and still apply beyond the trash domain.

Thus, we have been working on refining the skills, RAPs, and vision processing used in the trash cleanup task to the point where the task is always done reliably and effectively. We are doing this work, not because trash cleanup is a particularly interesting or difficult task in its own right, but because we feel the best way to make progress is to extract all of the lessons we can from one task before moving on to the next. Our goal is not to have a robot that can perform a variety of independent tricks in controlled surroundings. We want to be sure that the robot can cope with all of the contingencies that come up in even a simple task like collecting trash.

This section of the report describes the work that we have done in five areas over the last year:

1. *The trash cleanup task.* We describe some of the trash cleanup skills and RAPs to give a feel for our current task plan representations.

2. *Lessons learned.* We use the trash cleanup plan representation as the context for discussing several lessons that we have learned while trying to make those representations reasonably generic.

3. *Vision skills for interacting with objects.* We also discuss the vision skills we have implemented while attempting to make the trash cleanup task representations more generic. These involve different ways for finding and tracking objects in different situations.

4. *Skills for human/robot cooperation.* We have also begun work on several projects designed to build up skills and representational expertise directed at communicating with people. These skills are not required in the trash cleanup task but will form the underpinnings of cooperative tasks we will address in the future.

5. *Infrastructure improvement.* We believe the only real test of our architecture and plan representation is out in the real world on a real robot. The last year saw some significant improvements to the software infrastructure used in these tests.

2.1 Experience with the Trash Cleanup Task

One task at the 1994 AAAI Robot Competition was to pick trash up off the floor and place it in a trash can. The contest defined three types of trash (soda cans, crumpled up balls of paper, and styrofoam cups) and a single type of rectangular black trash can. The goal was to throw away as many pieces of trash as possible in a given amount of time.

While addressing the trash collection task, we are attempting to build up "generic" skills and RAPs that can be reused as building blocks in RAPs for future tasks. Basically, building such plans is an exercise in knowledge representation where we must decide on a good set of semantic primitives (in our case skills) that can be combined appropriately into plans (or RAPs) for a variety of goals.

One approach we have been using to make skills more generic is to designate objects in the world indexical-functionally as the output of tracking skills [1]. For example, we do not have a skill specific to picking of a piece of trash. Instead, the skill is constructed to pick up a small object that is being designated by any skill writing to the `target-location` channel. We can then pick up a variety of object types simply by choosing different visual skills to designate their `target-location`.

The resulting trash cleanup task plan representation breaks down into the following components:

- Visual skills for finding and identifying trash and for finding a trash can.
- Action skills for approaching a piece of trash, picking it up, approaching a trash can, and dropping off the trash.
- RAPs for tying all of these skills together into coherent flexible plans.

2.1.1 Visual Skills

Two visual skills are used in the trash picking up task: a skill for finding and identifying trash, and a skill for finding a trash can. These skills are parameterized instances of more general object-finding routines, FIND-FLOOR-OBJECT and FIND-SMALL-FLOOR-OBJECT. Both skills drive

the `target-location` channel with an (x, y) coordinate for the desired target in a global coordinate system maintained by the robot.

A simple lesson that we learned is that each of these skills must come in two versions: a version that finds the desired object in the visual field and signals where it is, and a version that tracks an instance of the object type as it moves around in the visual field. The find-object version is required to gather information about the world while the track-object version can use information about the objects last detected location to speed tracking from frame to frame and keep the system focussed on a single instance of the object type. This issue is discussed in more detail in section 2.3.

Finding a Piece of Trash

The visual routine FIND-SMALL-FLOOR-OBJECT takes a high resolution edge image, closes the edges into connected segments, and computes texture, aspect ratio, size, and brightness statistics on each segment. Thresholds on these statistics classify each region as a cup, can, piece of paper, or something else. The skill sends a signal, :OBJECT-AT, identifying and locating the piece of trash closest to the robot, or :NO-OBJECT. The related routine TRACK-SMALL-FLOOR-OBJECT locates and identifies the nearest piece of trash, and then tracks it through the image, using a simple model of smooth motion with possible abrupt stops. With a cycle time under 0.5 seconds, we can predict an object's image location well enough to restrict processing to a small window, without explicitly accounting for body and head in motion (these movements are usually compensatory - see below). TRACK-SMALL-FLOOR-OBJECT signals :NO-OBJECT if it fails to find anything on several consecutive tries.

Finding a Trash Can

A visual routine called FIND-FLOOR-OBJECT is parameterized with a color histogram of the target and a set of edge template models. The histogram is back-projected [20] into the image to find regions of interest. A Hausdorff-distance template-matcher [14] searches the regions for the object across a range of scales and skews. The tracking version of the skill can restrict the range of scales and edge template models, based on the prior match. These skills send the same signals as above.

Pointing the Camera

An important part of tracking an object in the world is moving the pan/tilt head to keep the object in view. This is accomplished on CHIP using a skill called HEAD-TRACK-TARGET which constantly adjusts the head pan angle to center the current `target-location` in the camera's view. The system can track any object by enabling a visual tracking skill and the HEAD-TRACK-TARGET concurrently. This skill generates no signals unless the pan/tilt head malfunctions.

There are also two simple skills for pointing the camera is a desired direction. The HEAD-PAN-TO and HEAD-TILT-TO skills will move the head to point in the pan and tilt directions, relative to the robot, passed as parameters when they are enabled. They generate `:pan-at` and `:tilt-at` signals respectively when the head has reached its destination angle.

2.1.2 Navigation Skills

In addition to finding and tracking objects, CHIP needs to move around while picking up trash. The skills used are:

- Move to the target (x, y) coordinate while avoiding obstacles.
- Orient to the target (x, y) coordinate while avoiding bumping the arm.
- Move directly forward or backward a given amount.

Notice that these skills incorporate obstacle avoidance directly – it is not a separate process. Other authors suggest building navigation skills out of more primitive pieces [2, 4, 18] but that approach seems to lead inevitably to the problem of arbitrating between contradictory instructions from various subprocesses. This arbitration requires merging contradictory instructions in different ways depending on the navigation goal. Since doing arbitration based on navigation goal effectively means building a separate skill for each type of goal anyway, we have chosen to avoid the arbitration problem by defining those skills explicitly.

Moving To a Given Location

The MOVE-TO-TARGET skill moves the robot to the global position defined by the `target-location` channel while avoiding obstacles. The obstacle avoidance algorithm is currently very simple. It looks for traversable gaps in the local sonar data and picks the gap that will move the robot in the direction most directly towards the desired target location.

When the skill is enabled, it is passed a parameter that gives an acceptable dead-reckoning error radius around the goal location. The skill generates a `:at-target` signal when it moves within the error radius of the goal. In addition, the obstacle avoidance algorithm is not perfect and occasionally the robot will get stuck in local minima. The navigation skill detects these minima by noting that a given amount of time has passed without getting any closer to the goal or there is no reasonable gap in the current sonar readings. In these situations the skill signals `:stuck`.

Orienting to a Given Location

The ORIENT-TO-TARGET skill adjusts the robot's heading to point toward the `target-location` channel and then moves backwards or forwards to get a given distance from the target. This skill attempts to back away from sonar readings that suggest the arm would bump something while the robot is turning. This skill is used to align the robot's arm from manipulating objects and to turn the robot to face a particular direction.

This skill generates a `:oriented-to-target` signal when the robot is properly aligned. If the robot cannot back away from an obstruction and thus can't turn to face the target, or move the correct distance away, it generates a `:stuck` signal.

Other Navigation Skills

The ROLL-FORWARD and ROLL-BACKWARD skills move the robot directly forward or backward a given distance. They are used to move the robot so its arm is over the trash can and then move

it back again so its arm can be lowered. These skills are useful only over short distances because they just dead-reckon the distance traveled and do not refer to the current target or avoid obstacles. Each skill signals `:at-goal` when it has traveled the required amount or `:stuck` if it cannot move forward or backward because of a detected obstruction.

The world is not completely predictable and sensor and actuator uncertainty will always exist so there are many situations in which the navigation skills might make mistakes. For example, while moving around an obstruction, a person might suddenly step out in front of the robot violating all of the active skill's assumptions about where it is safe to move. Thus, the robot might bump into objects with its body or its arm.

To deal with these situation quickly, three skills are used: FRONT-SAFETY, BACK-SAFETY, and TURN-SAFETY. Whenever skills that move the robot are enabled, these skills are enabled concurrently. Each is given a radius of expected free space and will constrain the robot to stop moving forward, backward or the direction it is turning if an object is detected within that corresponding radius either with sonars or touch sensors. If one of these skills detects an obstruction it also generates a signal named after itself so the RAP system will know what is happening.

2.1.3 Arm and Gripper Skills

The trash cleanup task also uses several skills for moving the robot's arm and gripper. These skills are fairly simple because CHIP has a simple four degree of freedom arm that extends outward in front of its body. For a more complex robot, more complex skills would be needed to implement more complex grasping strategies.

Moving the Arm and Wrist

CHIP's arm has no redundant degrees of freedom so the skill for moving the arm simply takes the desired gripper height and distance in front of the robot, does the inverse kinematics, and moves the arm shoulder and elbow joints appropriately. This skill generates an `:arm-at` signal when the arm has reached the right place.

There is a similar skill for moving the wrist to a given tilt and twist. It generates a `:wrist-at` signal when the wrist has moved through the appropriate angles. These two skills do not detect problems because the arm does not have enough sensors on it to detect a collision. In the future we hope to fix this.

Moving the Gripper

There are a variety of gripper sensor conditions that can be used to signal the end of gripper action so a family of simple skills have been implemented:

- Move to a given angle of opening.
- Close until a given gripper pressure is reached.
- Open until the IR beam breaks.

The skill POSITION-GRIPPER moves the gripper open or closed until a given angle of opening is reached. It is useful for opening the gripper to get ready to grasp something and for closing the gripper when tucking the arm away. The skill CLOSE-GRIPPER-TO-PRESSURE takes a desired finger tip pressure as a parameter and closes the gripper until that pressure is achieved. It is useful for grasping things. The skill OPEN-UNTIL-IR opens the gripper until the IR beam between the fingers shows that the hand is empty. It is used to let things go. All three skills generate the signal `:gripper-at` when the appropriate condition has been achieved.

Another signal generated by these skills is `:stuck` which occurs when the gripper stops opening or closing before the desired endpoint is reached. This message is generated when the motor moving the gripper continues to turn but the fingers stop moving.[1]

2.1.4 Primitive RAPs

The interface between the RAP system and the skill system is in the form of RAP methods that enable and disable skills. For example, to move the pan/tilt head the following RAP is used:

```
(define-rap (primitive-pan-to ?angle)
  (succeed (and (head-pan-angle ?b)
                (within (- ?angle ?b) 0.02)))
  (method
   (primitive
    (enable (pan-to ?angle))
    (wait-for (:pan-at ?a) :succeed (head-pan-angle ?a))
    (disable :above))))
```

This RAP describes a task for placing the head at a given pan angle. If the current `head-pan-angle` is within 0.02 radians (about 1 degree) the task is complete and the RAP succeeds. If not, the single method gives a primitive plan to enable the PAN-TO skill and wait for the `:pan-at` signal from that skill. When the signal arrives, the method finishes by disabling the skills it enabled.

The third argument to the wait-for clause declares the result of executing this method if the specified signal is received. The RAP system allows rules to be defined that change the RAP memory based on the results of completed tasks and methods [9] but for this paper the results can be treated as if they are simply asserted directly into memory.

Visual skills are used exactly the same way, such as in the RAP for generating a `target-location` using a color histogram and a list of Hausdorff edge models as cues for identifying an object to find:

```
(define-rap (primitive-find-floor-object ?hist ?models => ?x ?y)
  (method
    (primitive
       (enable (find-floor-object ?hist ?models))
       (wait-for (:object-at ?type ?x ?y) :succeed (object-at ?x ?y))
       (wait-for (:no-object) :fail)
       (disable :above))))
```

The knowledge represented for the trash cleanup task includes primitive RAPs for:

[1] This condition is detectable because the fingers themselves are connected to the motor via springs. Unfortunately, the skills used at the competition did not detect this state and broke the gripper during the finals.

- Looking for trash and objects with edge features or histograms and Hausdorff models.
- Tracking objects.
- Moving the pan/tilt head.
- Moving to a target location.
- Orienting to a target location.
- Moving the arm and wrist to given orientations.
- Opening and closing the gripper.
- Moving forward and backward.

Each of these primitive RAPs corresponds to a description for a discrete task that accomplishes a simple action in the world.

2.1.5 Composite RAPs

A hierarchy of more abstract RAPs is built on top of the primitives. The hierarchy is currently defined by the subgoals that we believe will be generally useful for other tasks in the future. Only at the very top of the hierarchy does information specific to trash cleanup come into play. This section of the paper describes part of the RAP reactive plan hierarchy.

Finding an Object

The RAP for finding an object in the current field of view is shown below:

```
(define-rap (look-for-object ?type)
  (method
    (context (small-object-model ?type ?thresholds))
    (task-net
      (t1 (primitive-look-for-small-floor-object ?thresholds) => ?x ?y)
          (mem-add (location ?type ?x ?y)))))
  (method
    (context (object-model ?type ?hist ?models))
    (task-net
      (t1 (primitive-look-for-object ?hist ?models => ?x ?y)
          (mem-add (location ?type ?x ?y))))))
```

This RAP chooses a visual skill for detecting the desired object type based on the models needed for those routines stored in the RAP memory. When the RAP that invokes that skill completes successfully, an annotation of the found object's location is made in memory.

This SCAN-FOR-OBJECT RAP builds on top of this RAP by using it in a simple visual search to look for an object of a specific type. The visual search consists of looking in three different directions 0.5 radians apart in front of the robot until an object of the appropriate type is seen. Each step in the search first moves the pan/tilt head to look in the appropriate direction and then generates a LOOK-FOR-OBJECT sub-task.

The end result is knowing where an object of the appropriate type is located. Similar RAPs exist for starting up the corresponding tracking skills.

Approaching an Object Type

The RAP for approaching a known object is shown below:

```
(define-rap (go-to-object ?type ?radius)
  (succeed (at-object ?type))
  (method
    (context (and (location ?type ?x1 ?y1)
                  (my-location ?x0 ?y0 ?heading)))
    (task-net
      (sequence
        (t1 (primitive-pan-to (angle-between ?x0 ?y0 ?heading ?x1 ?y1)))
        (parallel
          (t2 (track-object ?type))
          (t3 (go-to-target))
          (t4 (primitive-pan-to-target)))))))
```

This RAP first points the camera in the direction of an object of the type to be approached. Then, three sub-tasks are generated to run concurrently: TRACK-OBJECT selects and starts up a tracking skill to generate a `target-location` to the object being looked at, PRIMITIVE-PAN-TO-TARGET starts up the corresponding skill to keep the pan/tilt head pointed in the direction of the target location, and GO-TO-TARGET moves the robot towards the target location while avoiding obstacles.

Picking Up Trash and Dropping It Off

There is also a RAP for orienting to an object of a given type that is very similar to that for approaching the object. It enables to the ORIENT-TO-TARGET skill along with skills for tracking an object of the desired type and for moving the camera to keep the object in view.

It is particularly important that the reactive plan for orienting to an object also track the object as the robot is moving. Continually tracking a target and generating new target coordinates helps to eliminate the inevitable errors that arise in the robot's position and movement. Also, averaging the target coordinates over time helps to minimize tracking errors in the visual skill itself. Closing the visual feedback loop this way is an essential error reduction step.

Once the robot has lined itself up accurately with an object, there is a collection of simple RAPs to enable the skills for positioning the arm down near the object, for moving forward until the object is in the gripper, closing the gripper, and then lifting the object up. These RAPs specialize the grasp strategy somewhat based on the type of the object to be grasped. For example, paper will not always break the IR beam so the corresponding RAP does not fail if the beam is not broken when grasping paper.

Similarly, there are RAPS for moving the arm up higher than the trash can, for moving forward, for dropping off the trash, and then for moving back away from the can. For example:

```
(define-rap (drop-held-object ?container-type)
  (succeed (not (arm-holding ?anything)))
  (method
    (context (my-position ?x0 ?y0 ?heading))
    (sequence
```

```
    (t1 (look-for-object ?container-type)
        ((location ?container-type ?x ?y) for t2))
    (t2 (raise-arm ?container-type))
    (t3 (move-forward (distance-to ?x0 ?y0 ?x ?y)))
    (t4 (release-held-object))
    (t5 (move-backward (distance-to ?x0 ?y0 ?x ?y)))
    (t6 (fold-arm)))))
```

2.1.6 Putting it Together

At the top of the RAP hierarchy is the cleanup-specific RAP CLEAN-UP-TRASH:

```
(define-rap (clean-up-trash)
  (succeed nil)
  (method
    (task-net
      (sequence
        (t1 (scan-for-object trash))
        (t2 (go-to-object trash 80))
        (t3 (orient-to-object trash 40))
        (t4 (pickup-object trash))
        (t5 (scan-for-object trash-can))
        (t6 (go-to-object trash-can 120))
        (t7 (orient-to-object trash-can 70))
        (t8 (drop-held-object trash-can))))))
```

2.2 Lessons Learned

In implementing the trash cleanup task, we learned some interesting lessons which, in hindsight, should be obvious from the way we represented the plans. This section of the paper discusses four of those lessons:

- The usefulness of memory.
- The need for visual skills to find and track objects.
- Problems with indexical-functional designation and plan continuity.
- Shifting sensing steps forward in plans.

2.2.1 The Usefulness of Memory

An issue of current debate in the AI reactive planning community is the role that memory should play in an intelligent agent. Some researchers believe that recording facts in memory is pointless because the world is constantly changing and recorded facts will always be out-of-date. They argue that little or no memory should be used to keep an agent from making errors based on such incorrect facts [3].

We have considerable sympathy for this point of view but both our intuition and our experience has demonstrated clear benefits in using memory for remembering where things are. In the trash cleaning task, the robot must continually return to the trash can to drop off pieces of trash. Remembering the trash can location reduces the amount of search required to find it significantly because the robot knows where to start looking. Seeding the trash can search with a remembered location cuts the time to drop off a piece of trash by around 30% in our simple cleanup scenarios.

The problem with memory is not that it *might* be wrong, but figuring out how to use it wisely given that it *will* be wrong. The role of memory is to help an agent make intelligent choices when direct sensory data about the objects in question is not available. Agent's should base as much of their behavior as possible on sensing rather than memory, which is to say, they should prefer feedback control over open-loop control whenever possible. However, when feedback is not available, decisions can be made based on memory instead. When the trash can is not currently in view, which direction should CHIP turn to bring it into view? If CHIP has know idea where the trash can is, it must scan around until it comes into view, but when CHIP knows where the trash can was last, it can turn in that direction and start to search from there.

The RAPs used in the trash cleanup task treat the trash can location as an initial place to look because memory of that location will sometimes be wrong, either due to accumulating errors in the robot's perceived location or because the trash can has moved. Basically, when the robot goes to the trash can at a remembered location, it moves to bring the trash can into view and then executes the SCAN-FOR-OBJECT plan step. This RAP is written to check forward first, so if the trash can is in the remembered location and the robot brings it into view, the RAP will succeed right away and the robot orients itself to the can. On the other hand, if the trash can did not come into view, SCAN-FOR-OBJECT will continue searching for the trash can.

One of the general representational principles we have learned is that memory is a powerful tool but RAPs that use memory must be written to exploit memory when it is correct but cope effectively when it is wrong. Furthermore, whether memory is used or not, errors always creep in and uncertainty-reducing steps like tracking and orienting will always be necessary. This is one reason the robot's behavior is always guided by indexical-functional reference.

Using memory for object locations also allows additional sensory information to be put to good use. For example, when the trash finder generates several candidates, they can all be placed in memory so that they can be found more easily later. During the competition, our system detected only one piece of trash at a time and after putting that piece in the trash can, the robot had to search all over again for more trash. It would have been substantially more efficient to record the position of all pieces of trash detected.

2.2.2 The Need for Object Finders and Trackers

Another lesson we have learned while writing plans is that they are most intuitive when they refer explicitly to objects in the world. Thus, executing plans that refer to objects requires vision skills for two distinct purposes: for finding objects in the world, and for tracking known objects to close control loops.

Finding an object is an important first step in any plan that uses an object. CHIP must find a piece of trash before it can pick it up and it must find the trash can before it can throw the piece of trash away. Finding an object typically requires searching the entire visual scene looking for clues

that the desired object might be present and then applying specific analysis algorithms to confirm its identity.

Tracking an object is required to close skill-based control loops. For example, CHIP tracks the piece of trash it is approaching so that movement errors do not take it to the wrong place and we have programmed CHIP to track moving people and follow them around. Tracking an object typically must be done very quickly to keep up with changes in the world in real time. To attain this speed, tracking skills can often take advantage of continuity to reduce the area of the visual scene that must be examined and the detail with which it must be analyzed. Once a person has been identified in the visual field, simple motion or color cues that can be computed in real-time are often sufficient to track the person around the room.

The dual use of visual skills has caused us to think of virtually *every* visual task as two separate tasks: an identification task and a tracking task. Plans that make refer to an object cannot be executed robustly unless the robot can find and identify the object and then track it in real time.

2.2.3 Using Indexical-Functional Designation

To help cope with position and sensing errors, the RAPs and skills used in the trash cleanup task make indexical-functional reference to objects in the world. That is, the RAPs for orienting and grasping are essentially orient-to-the-thing-of-this-type-in-front-of-me and grasp-the-thing-of-this-type-in-front-of-me.

A problem with using indexical-functional reference in plans consisting of modular steps, like the trash RAPs, is maintaining a consistent target designation from plan step to plan step. The piece of trash seen originally might not be the same one approached, which might not be the one oriented to for grasping because at step boundaries, the object tracker is stopped and restarted allowing it to "lock on" to a different object of the same type. In a purely reactive world like the trash collection task, this is not necessarily a problem since one piece of trash is as good as the next. However, this behavior does not appear very intelligent and can cause unexpected problems.

For example, if the trash tracker always tracks the closest piece of trash, the robot might be turning towards a target when a closer piece comes into view; the robot backs up and turns toward it, and another, closer piece comes into view, and so on. There is a need for coherent indexical functional reference. A single piece of trash needs to be chosen and tracked even as distracting closer pieces come into view. Furthermore, this "lock" must be maintained across RAP plan steps. This can be achieved in two different ways:

- Change the tracking visual routines so they start with the visual location and characteristics of the desired object. The tracker can use this information to maintain better continuity of the tracked object across the brief periods that it is stopped between steps.
- Add another layer of abstraction in the RAP hierarchy and create a find-and-pickup-object RAP as shown below. This RAP defines a finding, approaching, and orienting step sequence, which is executed in parallel with a single tracking RAP. The tracker is never stopped and the "lock" is not lost over the sequence of find, approach, pick up steps.

```
(define-rap (find-and-pick-up-object ?type)
  (succeed (and (arm-holding ?x)
                (isa ?x ?type)))
```

```
(method
  (task-net
    (sequence
      (t1 (scan-for-object ?type))
      (parallel
        (t2 (track-object ?type))
        (t3 (primitive-pan-to-target))
        (sequence
          (t4 (go-to-target 100))
          (t5 (orient-to-target 40))
          (t4 (pickup-target ?type)))))))
```

In the long run, both of these strategies will be necessary. When targets, or the robot, are constantly moving, say, when trying to pick up a hamster, tracking a single target over multiple plan steps will be vital or else visual coordinates will get rapidly out of date.

However, it is also important to be able to separate RAP steps neatly to maximize composability. When continuity across steps is necessary but less time-critical, the second approach is preferable for this reason. We have found that in real plans with indexical-functional reference, there is a constant tension between composability of steps and continuity across them.

2.2.4 Shifting Sensing Steps Forward

A problem with using object specific active vision skills to find and track objects is that the agent only sees what it is looking for at the moment. For example, given the trash cleanup plans above, CHIP looks for a piece of trash, picks it up, and then looks for the trash can. If the trash can appears in any of the visual scenes processed to find the trash, it simply goes unnoticed because the skill to identify the trash can is not active.

We would like our robot to notice useful data as it first becomes detectable, but as a matter of principle, all sensing (except for danger signals) is done only as needed. The trash finder looks for only trash because vision processing *must* be organized around special purpose active vision skills rather than a general purpose "object finder." To maintain object specific vision skills and still notice an object that will be needed later, the sensing steps for detecting the object must be moved earlier in the plan. For CHIP to find the trash can while searching for trash, it needs to be running the trash can finding skill in parallel with the trash finding skill as shown below:

```
(define-rap (clean-up-trash)
  (succeed nil)
  (method
    (task-net
      (sequence
        (parallel
          (t1 (find-and-pickup-object trash))
          (t2 (scan-for-object trash-can)))
        (t3 (go-to-object trash-can 120))
        (t4 (orient-to-object trash-can 70))
        (t5 (drop-held-object trash-can))))))
```

It is certainly possible to leave sensing steps less ordered in RAP plans so they will be running in parallel, but such lack of order has two disadvantages: it requires abstracting plan steps into new intermediate steps that the incorporate parallelism (like CLEAN-UP-TRASH), and it requires careful coding of visual skills so they do not interfere with each other when operating in parallel. The first of these is a knowledge representation issue that again hints at the complexity required in realistic plans. The second places many new demands on the visual skills, since they must run across a broader range of contexts (*e.g.*, head position, body speed, etc.).

2.3 Parsing the World into Objects

In representing real plans for tasks, we have found that the role of vision is almost always to answer specific questions regarding concrete objects that may be in view. These objects may be central to the task (*e.g.*, a piece of litter to be collected), or instrumental to it, (*e.g.*, a room sign that can establish the robot's location while driving in a hall.) The routines we use to gather this information are embodied in separate visual skills that are applicable under different circumstances.

Here we describe the different visual skills we have developed, the circumstances for which they are appropriate, and the software which they use. We have been able to use a good deal of publicly available research software, but we have also had to develop some new algorithms and code as well.

2.3.1 Segmenting Objects

When a behavior depends on knowing where some object is, the RAP for that task must enable a visual skill capable of finding the object. Part of the representation for such tasks includes selecting the visual skill appropriate to characteristics of the object and background. The most important distinction between visual targets in this regard is whether or not the object is well-known and will appear in a typical pose, that is, whether the robot knows in advance how the object will appear.

This section describes the different visual skills for locating objects that we use, depending on how well we can anticipate what the object will look like.

Finding a Specific Object

Often the robot needs to know where a particular very familiar object is. If this object always appears in the same pose (for example, a usable trash can will always be upright), then the robot knows what the shape and color will be.

To find a well-known object, we compare edge-models of the object, taken from different views, against an edge image, using the Hausdorff image-distance metric. The Hausdorff metric allows deformable template matching, with partial occlusion, across a controllable range of scales and skews. For well-known objects, we also store a color picture of the target, and use the histogram back-projection technique [20], computed on the DataCube Server [16], to find regions of interest over which the Hausdorff matcher searches. For finding trash cans (Figure 2.1), the region of interest restriction typically improves performance by a factor of fifty. The Hausdorff matcher uses software available publicly from Cornell [14].

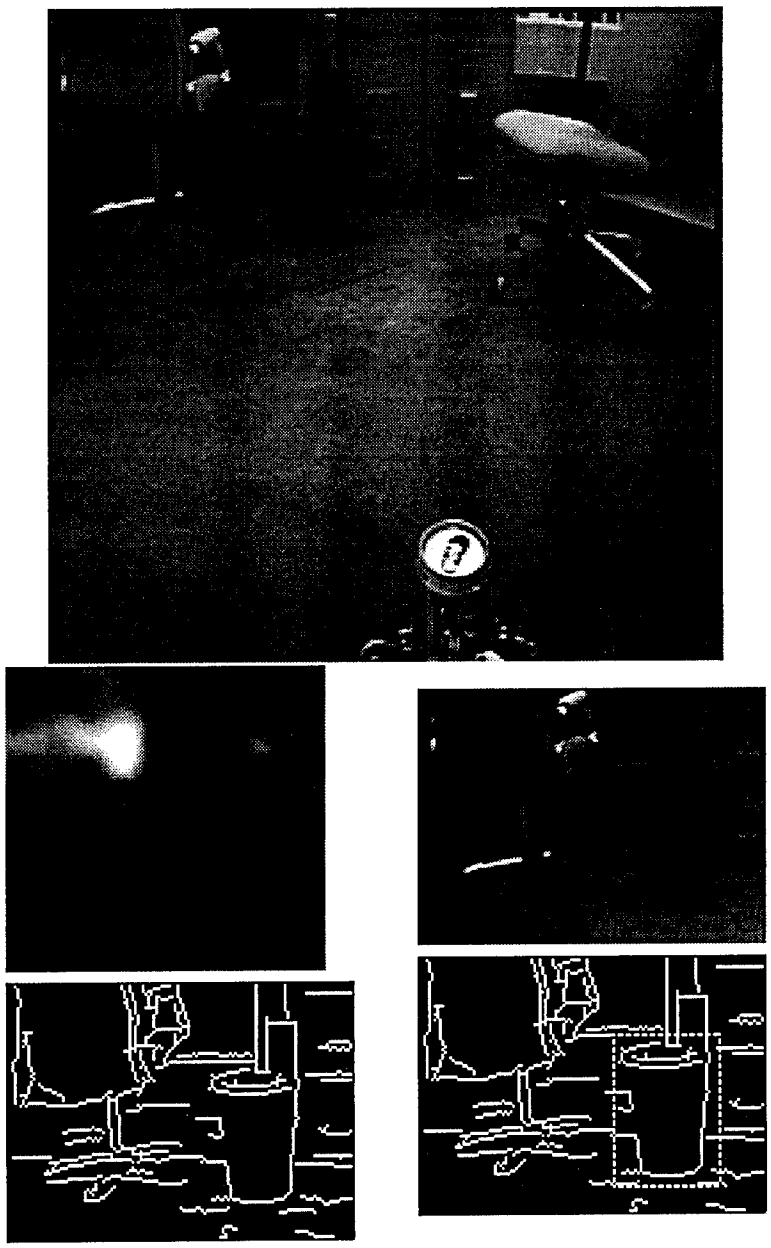

Figure 2.1: Top: Scene with target (trash can) in view. Middle, left: low resolution color match. Middle, right: Region of interest. Bottom, left: edge image in region of interest. Bottom right: best match to model found.

Finding Generic Small Objects

The Hausdorff matcher is less effective for finding instances of a broader class of objects, such as all pop cans, because the common characteristics of these objects can't be captured as an edge model. For example, the common outline of all cans is too vague a model, and will result in many false matches to generally rectangular objects. The writing and logos on cans differentiate them from most of these objects, but there is no common model for all these logos.

We therefore developed another visual skill for segmenting unknown small objects, against a plain background, without knowing exactly what they will look like. Our algorithm segments a scene into closed boundaries, and selects those regions that could arise from a small object (one that would fit in the robot's hand) lying on the floor [8]. The DataCube Server grabs and smoothes an image and detects edges, and the Khoros image processing library [11] computes morphological and intensity filtering operations to remove noise. The result is a list of possible image regions corresponding to small objects on the floor. This skill is appropriate for avoiding trash in the robot's path, for example.

Finding Classes of Small Objects

This skill for finding any small object on the floor was refined to identify particular classes of small objects, including pop cans, paper wads, and styrofoam cups. To make these distinctions, we compute texture, brightness, aspect ratio, and size parameters and classify these against statistics manually extracted from a database of sample images. To recognize new classes, we would need to determine new parameter sets; we are beginning to automate the process of acquiring these parameters from test images.

Using triangulation and cached calibrations, the small object locator gives us an accurate three-dimensional location of an object when it is constrained to lie on the floor, sufficient to orient the robot to it and pick it up (a tolerance of about 2 cm.) The classification procedures identifies more than 90 per cent of real targets during normal performance, with almost no false positives. In the trash collection task, the cost of failing to see a piece of trash is very low, but the cost of picking up and throwing away something that is not trash is high.

Finding and Reading Signs

Finally, we have developed a specialized visual skill that is useful for reading room number signs while navigating. This routine has proven useful at distances that make edge detection for the small sign letters problematic, which prevents us from using the more general Hausdorff-based object finder. Like the Hausdorff object finder, this algorithm first finds regions of interest using the known color of the signs. Then, it scales each region to the same size and shape as the models in the model base (a set of 30 rectangular room signs.) Each image region is correlated against each model, and the best matching model, above a recognition threshold, is found. The position, scale, and pose of the sign determine the robot's location with respect to it.

2.3.2 Tracking Objects

The object segmentation routines above answer visual needs such as, "find a piece of trash, and tell me where and what it is." However, they can not currently answer a question like "which of these

cans is the one I was going to pick up?" All the classification routines, except the sign reader, only establish the location of some class of object, not a particular instance.

Combining Tracking and Object Finding

To maintain coherent reference to a particular instance of some class, the robot visually tracks the object continuously. The software for visual tracking has two components: an object-specific recognition routine, and a generic image-tracking routine. The recognition routines are usually the same as the segmentation routines described above, but modified to run on a region of interest established by the image-tracker. The tracker combines object sightings into an image trajectory and predicts a window in which the object's image will be found on the next cycle. The smaller the window, the faster the object can be found, and with fewer distractions. This shortens the viewing cycle, and shorter cycles in turn leads to tighter predictions, i.e., smaller windows. The tracking version of the trash classification routine typically runs more then ten times faster than the non-tracking (whole scene) version.

Specialized Tracking

We have also successfully tracked more than one person using a combination of color and motion cues. The DataCube Server computes color histogram backprojection and normal optic flow [13] across the scene. These images are rapidly segmented into coherent moving blobs or target-colored blobs using a new, pseudo two-dimensional segmentation algorithm [19]. The same tracking routine knits successive sightings into independent trajectories, and allows the robot to predict where an object will appear. The integrated color-motion tracker automatically switches from color or motion tracking according to the number of distracting like-colored or moving objects in the scene.

2.4 Technologies for Cooperative Interaction

We also have a number of projects underway aimed at enabling natural ways for people to interact with robot agents. These projects build on the infrastructure and basic capabilities that we have been incorporating into CHIP over the last year.

There are two ways that a robot can communicate with people in a cooperative way: by using natural language, and by watching for physical gestures and signals. Natural language is especially good for communicating new plans and giving advice on improving known plans. On the other hand, gestures are often better for communicating environmentally based details of tasks in progress, such as which object to pick up. We have been developing the underlying technology for both types of communication by forming connections to projects that include:

- *Natural language understanding.* New plans and advice can be communicated to the to the robot interactively most easily through natural language.
- *Visual routines for watching people.* Finding people and their body parts is a prerequisite to any sort of visually based communication such as:
  - *Recognizing gestures* such as pointing, for directing the focus of attention of the robot's visual system and disambiguating task descriptions.

- *Face recognition*, for confirming that a given object is actually a person, and for disambiguating individuals.

- *Physical interaction.* Many simple cooperative tasks require physical interaction between the robot and a person, like handing objects back and forth.

2.4.1 The Marathon Advice Taking Project

The Animate Agent Architecture is being tested within simulated domains as well as on the robot CHIP. The simulated domains are two video games in which the agent interacts with a realistic three-dimensional rendered scene: Pathways into Darkness and Marathon, popular games for Macintosh computers known for their rich, realistic graphics. They permit other processes to obtain the image of the 3D scene and return a keymap denoting the agent's action. The game takes place in a dungeon filled with useful objects and dangerous monsters. We have built visual routines for the critical tasks of finding the floor, and finding objects lying on the floor and identifying them.

The cooperation in this project is with an experienced player of the game who views the same three-dimensional scene as the agent and gives the agent advice using natural language, *i.e.* English. Using a simulated domain permits us to slow down the world by a chosen amount to avoid expensive needs for advanced computer hardware and time-consuming optimization of our perceptual routines, yet at the same time it still retains some notion of "real-time", so the agent still has resource bounds.

Our goal in this work is to concentrate on higher level problems, such as development of intermediate visual primitives that can be referred to in conversation with a human advice-giver.

2.4.2 Face recognition

We have conducted extensive experimental tests with Matthew Turk's eigenface matching algorithm [21], and have determined that accurate alignment of the model with the face is vital for accurate recognition. We have also made some progress in representing the face classes that describe different people, so that we can learn the class representations over multiple exposures.

2.5 Testbed Development

Over the last year, our experiments in implementing tracking, navigation, and trash collecting tasks have been done primarily on the Animate Agent Project testbed robot CHIP. As we try to move to more sophisticated behavior, we have been improving both the hardware and software components in this testbed.

2.5.1 Improving RAPs and CRL

Our implementation of the Animate Agent Architecture consists of two major software systems: the RAP system that implements symbolic reactive plans, and the CRL system that implements skills.

The RAP system has been in use for several years and has been released for public use via the Internet. It has become a very robust piece of software, but as our experiments with more complex,

real-time tasks progress, it continues to evolve. In particular, the RAP language for representing task plans was changed in two major ways in the last year. The WAIT-FOR clause needed to coordinate symbolic and continuous processes was refined and made much clearer over the course of experiments at MITRE Corporation (which uses RAPs) and over the implementation of the trash cleanup task. These changes are described in [7]. Also, the notion of memory rules for updating RAP memory in response to external events and subtask completion was further refined. This work is described in [9].

The CRL system (Chicago Robot Language) is a simple LISP-like language used for writing our continuous skills. It interfaces to the RAP system, the robot, and the vision system to create the run-time environment used on CHIP. The major features of CRL are:

- Its runs distributed across the computer onboard CHIP and the Macintosh computer offboard that runs the RAP system.
- Skills can be changed interactively without having to recompile while debugging.
- When skills have been debugged interactively, they can be compiled into 'C' code for faster execution.
- It is easy to port to other hardware platforms (as long as they run LISP).

Over the last year, CRL has gone through one revision to fix a variety of bugs in the initial implementation. We have also been talking with MITRE to see if it might be possible to build a joint skill based on CRL and a 'C'-based skill compiler they have developed independently.

As our hardware infrastructure evolves, and we get more experience with implementing tasks and skills, we expect to make more use of commercial real-time operating system technology within CRL. However, we will continue to need a language for writing skills and an overall system designed to allow individual skills to be enabled and disabled independently.

2.5.2 Improving the Vision Infrastructure

CHIP's vision processing is distributed over a Sun Sparc-20 workstation and a DataCube MV-200 image processing computer with a DigiColor digitizer. The DataCube is a pipelined processor that can perform many operations including convolutions, histogramming, and feature extraction in real-time. The DataCube Server [15] allows multiple processes to access the DataCube simultaneously from any machine on the network. The Server permits us to implement our visual routines as separate processes potentially distributed across multiple computers while still allowing each visual routine to use the DataCube's processing power.

There is a set of low-level visual operations that are widely used among the visual routines and/or are time-critical and therefore have been implemented on the DataCube. These operations include the creation of Gaussian and Laplacian pyramids, extracting subregions from a level of one of the pyramids, convolution, and histogramming, color histogram backprojection and motion detection. Additions to the Server in '94 were the Laplacian pyramid (useful for stereo vision), edge detection via the Sobel operator, and the ability to do fast graphics operations.

One difficulty we have had with the Server is that the DataCube's memory is global among all the processes that are connected to it. Therefore each process must defensively re-load images

and look-up tables stored in the DataCube memory between subsequent calls to the Server. We intend to make the Server easier to use and faster by keeping track of each clients state within the Server. With this modification the Server will automatically re-load each clients state when they are scheduled, thus simplifying client programming. The Server will only re-load the parts of the clients state that have been corrupted by other clients and that are necessary for the current clients next operation. This will resulting in significant reductions in execution time.

The Server was released publicly in November '93 and numerous institutions have downloaded the system. Hughes Research Laboratories in Malibu, CA. has taken particular interest in the Server and we have modified it extensively to interface with their DAP computer (a SIMD machine).

2.5.3 Improving CHIP

The robot we have constructed as a testbed for our ideas uses a three-wheeled omni-directional mobile base built by Real World Interface (RWI). Stereo color cameras are mounted on top of the robot on a two degree of freedom computer-controlled pan-tilt platform. Video signals are can be transmitted off-board to the Datacube digitizing hardware by a video transmission or by cable link. We have built a signal multiplexer transmits both camera signals in alternating frames, allowing video transmission of the signal from both cameras. The robot sensors also include sonar sensors and infrared proximity sensors. The sonars can be turned to face the surface nearest the robot to minimize problems due to specular reflection. The infrared sensors are well equipped for detecting obstacles near the robot (their signal drops off quickly after about ten centimeters), and have failure modes that are complementary to the sonar sensors. For manipulation, CHIP is equipped with a Heathkit Hero arm and manipulator. Force and contact sensors on the gripper provide tactile feedback.

Microprocessors mounted on the robot run the sensors, do initial processing of the data they create, and skills that do not require visual processing, which is done off-board. The microprocessors share a small amount of memory to allow communication and synchronization. The on-board microprocessors talk via radio modem to a Macintosh computer, which runs some control routines and the reactive planner.

Improvements in '94 included creating the video multiplexer board, adding a faster 68000 processor on-board, so that skills could be run locally without transmission to off-board computers. We are also in the process of adding a bumper to the robot, making it touch sensitive so that it can detect collisions with other objects.

Chapter 3

Plans for Next Year

Our plans for the next year focus on demonstrating cooperative behavior between our robot people and human users. In particular, we will concentrate on sensing people so that the robot can find, identify, and physically interact with them. In addition, we have plans to introduce some ability to communicate using natural language. These abilities will be illustrated using three integrated demonstrations: expanded and improved trash cleanup, a robot "gofer", and a software natural language advice taking agent in the Marathon 3D-graphics simulated game world.

3.1 New Visual Skills

These demonstrations will require a number of new visual skills built upon those we have already.

Finding Unknown Objects

The robot sometimes needs to segment objects *without knowing what they will look like, and without being able to assume that they will not overlap with edges in the background.* We will use stereo disparity to segment a scene containing objects that can visually overlap. This lets the robot locate large obstacles, such as furniture, walls, and door frames, for visual navigation. We do not need to identify the objects, only to establish that they are present. This algorithm complements the small object finder, which finds obstacles clearly visible on the floor, but too small to segment from the floor by disparity. Together, they will allow the robot to navigate successfully in cluttered rooms. We are developing this behavior now.

Sensing People

Visual routines for perceiving salient features of the people that interact with the robot are some of the most important perceptual capabilities for building an effective cooperative agent. Finding people, locating the face and hands, identifying the person, and interpreting basic gestures such as pointing are some of the necessary visual capabilities.

Another task that will exploit stereo object segmentation is finding the part of a scene pointed to by a person. A visual skill for segmenting people in a scene uses motion to find moving parts of

people, and stereo disparity to separate the entire person from the background, beginning with the points of motion. With the person segmented, a line can be drawn from the head through the tip of an extended arm to generate a region of interest cone. This could be used to point out a particular area of the floor to be cleaned, or a place to deposit a tool that was fetched. The software we use to compute disparity is publicly available [5].

One of our students, Roger Kahn, is working on situated visual attention and he will be showing how constraints such as those introduced by a cooperating person can be used to focus the resources of a robot's visual system, *e.g.* selecting a region of the scene to addend to. It is within this context that Kahn will develop techniques for locating and tracking salient aspects of the person the robot is interacting with. In particular, he is using motion and stereo cues to locate the head and hands, since knowing the location of each is necessary for understanding where the person is pointing.

For identifying people, we have been exploring face recognition. We have found that face recognition using correlation approaches such as eigenface-matching [21] is greatly improved if accurate registration is done first using features that can be well localized, such as eyes, nose and mouth. Thus, our belief is that face recognition is best broken down into the routine: find head, find facial features, correlate. These improvements, and others such as those introduced by [17] will be combined with improvements that rely on continuous viewing of the face: repeated application of the recognition algorithm as the face and features are tracked, and determining when the person is facing the camera (which provides the most effective recognition). Knowing where they are facing when not facing the camera can also be of use in human-machine interaction.

Human-Robot Physical Interaction

A useful skill for a robot helper is to be able to hand objects to a human and to receive them in return. Tasks such as these test our abilities to develop skills that are equipped with adequate sensing to sense the important features of a changing environment, and to develop skills that can represent a task that is dependent on the actions of the human.

Our first step will be to work on the eye-hand skills necessary to grab an object held out to the robot, based on visual servoing techniques such as [12]. Such techniques allow the robot to compensate for movements of the hand of the person holding the object. Our aim is to be able to effectively hand simple-shaped objects back and forth between the robot and a person who is somewhat practiced at the handoff. Effective hand-offs require sensing the orientation of the object and of the hand, sensing the state of the human grasp (held, not held, clear), and avoiding collision with the human.

3.2 The DMAP/RAP Integrated Language System

One of the most useful ways to interact with another agent is through natural language. However, natural language interfaces are extremely difficult to build because the meaning of utterances often relies very heavily on "non-linguistic" context, such as the task the speaker or hearer is engaged in. Thus, language understanding must be highly integrated with an agent's current plans, perceptions, and actions. The Animate Agent Project seems like a natural testbed for language understanding systems because it supplies the context required.

We have been working with Professor Charles Martin at the University of Chicago to find ways of integrating his Dynamic Predictive Memory Parser (DMAP) with the RAP system we use for executing reactive plans. As we build up more complex task plans we hope to have the robot interact more meaningfully with human users by integrating natural language through the DMAP/RAP project.

3.3 New Integrated Tasks

We plan to work on three new complete tasks to test the architecture and illustrate the techniques we have developed.

Extending the Trash Cleanup Task

We believe that we can continue to learn from the trash-finding trash both by increasing performance and by extending the task to include new challenges. Increasing performance will come from implementing the lessons referred to in Section 2.2, such as developing a spatial representation of trash located but not picked up, tracking individual objects from one plan step to the next, and developing more flexible sensing strategies to allow some opportunistic sensing.

We also intend to fill in gaps in the robot's error recovery, so that it can take corrective action and complete the task after all but the most catastrophic of errors. Recoverable errors include mistaking trash or the trash can, dropping trash, or not finding remembered objects.

Extensions of the task will include retrieving trash that is not directly on the floor, and sorting the trash into receptacles by type. Retrieving trash not on the floor will challenge us to make our eye-hand coordination strategies more general. Sorting the trash will also test the generality of our eye-hand skills, and give us more experience with finding known objects.

We will enter and compete strongly in the 1995 AAAI Robot Competition, in which one of the tasks is to pick up and sort trash by type into recycling bins.

Robot Gofer

As a demonstration of the gesture recognition and person finding and recognition skills we are working on implementing the plans for two robot "gofer" tasks. The first task is to move a number of objects from one location to another and the second is to go and find a particular object and take it to a designated person. In these tasks we will be relying on gesture recognition to indicate the objects to be moved, the place to move them to, and the person to be given the designated object.

Marathon Project

Within the Marathon Project, we intend to construct a visual system that is complete enough to enable the artificial agent to play the game (at a rate reduced by a constant factor), and generate intermediate representations that can be referred to by an advice-giver speaking in natural language, and to hook up a complete vision/CRL/RAPs/DMAP system on which the advice-taking agent can be constructed.

Bibliography

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Sixth National Conference on Artificial Intelligence*, Seattle, WA, July 1987. AAAI.

[2] 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.

[3] Rodney Brooks. Intelligence without representation. In *Workshop on the Foundations of Artificial Intelligence*, MIT, June 1987.

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

[5] I. J. Cox, S. Hingorani, B. M. Maggs, and S. B. Rao. Stereo without disparity gradient smoothing: A bayesian sensor fusion solution. In *Proceedings of the British Machine Vision Conference*, pages 337–346, 1992.

[6] R. James Firby. Adaptive execution in complex dynamic worlds. Technical Report YALEU/CSD/RR #672, Computer Science Department, Yale University, January 1989.

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

[8] R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain. An architecture for vision and action. Submitted to IJCAI-95.

[9] R. James Firby and Marc G. Slack. Task execution: Interfacing to reactive skill networks. Submitted to IJCAI-95.

[10] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* PhD thesis, Computer Science and Applications, Virginia Polytechnic Institute, 1991.

[11] Khoros Group. Khoros 2.0. Dept. of EECE, Univ. of NM, Albuquerque, 1992.

[12] David J. Heeger and Eero P. Simoncelli. Sequential motion analysis. pages 24–28.

[13] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[14] Daniel P. Huttenlocher, William J. Rucklidge, and Gregory A. Klanderman. Comparing images using the hausdorff distance under translation. In *Proceeedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 654–656, 1992.

[15] Ari Kahn. Using the datacube client class. The University of Chicago, Computer Science Department, 1993.

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

[17] Baback Moghaddam and Alex Pentland. Face recognition using view-based and modular eigenspaces. In *Automatic Systems for the Identification and Inspection of Humans, SPIE Vol. 2277*, 1994.

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

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

[20] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[21] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3:71–86, 1991.

Appendix A

1994 Accomplishments

Year from November 1993 – November 1994 included the following accomplishments:

- Competed in AAAI Robot Competition. Placed 5th of 12 teams in the trash collection event. We were the only team to actually pick a piece of trash off the floor and throw it in the garbage can (the others took penalty points for unimplemented behaviors but moved faster).
- Developed trash collection task to the point that it has thrown away hundreds of pieces of trash.
- Demonstrated real-time person tracking and on-the-fly selection of algorithm appropriate to background (color or motion).
- Improved robot hardware and major pieces of software infrastructure: RAPs, CRL, Datacube Server, added re-usable visual routines for finding and tracking objects and people. The Datacube Server was released for public use.
- First demonstration of DMAP/RAPs, which integrates natural language understanding into the Animate Agent Architecture.

A.1 Related Publications

Task Networks for Controlling Continuous Processes. R. James Firby. *Proceedings of the Second International Conference on AI Planning Systems*, Chicago IL, June 1994, pp 49-54.

Task and Environment-Sensitive Tracking. Peter N. Prokopowicz, Michael J. Swain, and Roger E. Kahn. *Proceedings of the Workshop on Visual Behaviors*, CVPR-94, Seattle WA, June 1994.

The Capacity of Color Histogram Indexing. Marcus Stricker and Michael J. Swain. *Proceedings of the IEEE Conference of Computer Vision and Pattern Recognition*, CVPR-94, Seattle WA, June 1994.

Architecture, Representation and Integration: An Example from Robot Navigation. R. James Firby. *Proceedings of the AAAI Fall Symposium Series on Control of the Physical World by Intelligent Agents*, New Orleans LA, October 1994.

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.2 Edited Journal Issue

Michael Swain. Special Issue on Active Vision II, *International Journal of Computer Vision*, Vol. 12, Nos. 2-3, April 1994.

A.3 Invited Talks

James Firby: Integrating Reactive Planning and Active Vision. Naval Research Laboratory, Washington DC, November 1994.

Michael Swain: Active Vision and Control. *AAAI Fall Symposium Series: Control of the Physical World by Intelligent Agents*, New Orleans LA, October 1994.

Michael Swain: An Architecture for Perception and Action. NASA Johnson Space Center, Houston TX, November 1993.

A.4 Panels

Michael Swain: Environments for Situated Perception and Action. *Second International Conference on AI Planning Systems*, Chicago, IL, June 1994.

A.5 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.6 Software and Hardware Prototypes

The RAP system is now available over the Internet. The release includes the Truckworld robot simulator testbed and the RAP system including all of the recent extensions for interfacing with continuous processes. This can be found at URL `http://cs-www.uchicago.edu/ firby/raps`.