NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-03-19
May 9th, 2003

Vision in Context: Using Special Purpose Routines to Achieve More General Purpose Vision

Joshua D. Flachsbart

Abstract

Recently researchers have used task driven computer vision routines to solve specific problems. These routines have been successful by relying on specific traits of the task they solve and the environment in which they operate. One disadvantage of this method is that the specific constraints on which the routines rely may not hold in a given situation, or may even cease to hold in situations where they held before due to changes in the world. We believe that reasoning about those constraints explicitly and selecting appropriate special purpose routines for specific situations allows us to create a general purpose computer vision system both robust and broad in task coverage.

This dissertation presents our work testing this hypothesis. The Intelligent Classroom is our primary testbed for utilizing vision in different contexts. The Classroom is an automated lecture facility that acts as its own audio/visual technician. The Classroom environment is a good choice for this research because it provides interesting computer vision tasks by interacting with a human user, and it also has structure which provides constraints for reasoning about that computer vision.

In addition, we present a framework for low level computer vision, and two additional systems which utilize that framework to solve similar problems in different domains.

The main contributions of the work described here are:

- The Intelligent Classroom, a system which is able to use computer vision to produce a video of a lecture, as well as reason about how to sense for different user interaction tasks.

- Our vision system, Gargoyle, which provides a framework for constructing vision routines on the fly, which is required to allow reasoning systems to manage the vision routines. It also provides a modular interface for designing and reusing parts of vision routines.

- An implementation of existing vision routines in a robotic system, CHIP, utilizing this framework, which demonstrates the capability of the framework to implement proven systems.

- The Interactive Image Mosaic, which demonstrates the ease of reuse by being built quickly using a number of the Classroom's Gargoyle routines.

Keywords: Computer Vision, Intelligent Environments, Automatic Filming

NORTHWESTERN UNIVERSITY

Vision in Context: Using Special Purpose Routines to Achieve More General Purpose Vision

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Joshua D. Flachsbart

EVANSTON, ILLINOIS

June 2003

Abstract

Vision in Context:
Using Special Purpose Routines to Achieve
More General Purpose Vision

Joshua D. Flachsbart

Recently researchers have used task driven computer vision routines to solve specific problems. These routines have been successful by relying on specific traits of the task they solve and the environment in which they operate. One disadvantage of this method is that the specific constraints on which the routines rely may not hold in a given situation, or may even cease to hold in situations where they held before due to changes in the world. We believe that reasoning about those constraints explicitly and selecting appropriate special purpose routines for specific situations allows us to create a general purpose computer vision system both robust and broad in task coverage.

This dissertation presents our work testing this hypothesis. The Intelligent Classroom is our primary testbed for utilizing vision in different contexts. The Classroom is an automated lecture facility that acts as its own audio/visual technician. The Classroom environment is a good choice for this research because it provides interesting computer vision tasks by interacting with a human user, and it also has structure which provides constraints for reasoning about that computer vision.

In addition, we present a framework for low level computer vision, and two

additional systems which utilize that framework to solve similar problems in different domains.

The main contributions of the work described here are:

- The Intelligent Classroom, a system which is able to use computer vision to produce a video of a lecture, as well as reason about how to sense for different user interaction tasks.
- Our vision system, Gargoyle, which provides a framework for constructing vision routines on the fly, which is required to allow reasoning systems to manage the vision routines. It also provides a modular interface for designing and reusing parts of vision routines.
- An implementation of existing vision routines in a robotic system, CHIP, utilizing this framework. This demonstrates the capability of the framework to implement proven systems.
- The Interactive Image Mosaic, a system which demonstrates the ease of reuse by being built quickly using a number of the Classroom's Gargoyle routines.

Acknowledgements

The road to a Ph.D. is never an easy one, and my journey has taken a number of turns. Somehow I seem to have made it. That is of course not through some miracle, but rather through the help and patience of many different people over they years.

My first thanks go to those people who saw me through to the end, to my degree at Northwestern. My advisor, Kris Hammond, probably never thought he would see the day that he had a vision student, however he took me on and allowed me to continue my work. Without that I surely would have never gotten to where I am. He went beyond that however, and helped me focus on maintaining a vision of always keeping in mind, how the research is going to be used. Especially in Computer Vision it is easy to fall into the habit of research for its own sake, and I can not thank Kris enough for keeping my eye on the utility of what I was doing. That will serve me well in the future.

I also want to thank the other members of my committee, who saw this through. Ian Horswill helped me more clearly state who I am, and who I am not, while Larry Birnbaum was available with the big picture, when I became too focused on details, sometimes on an emergency basis!

Of course, the thing that makes graduate school more bearable are the other graduate students. My time at Northwestern has been blessed with many good friends. Dave of course, muddled through it all with me on the Intelligent Classroom; through good times and bad. Robb, Jay and Shannon have also spent many years keeping me sane, and

for that I thank them. In any school, students come and go, but in the InfoLab there is a camaraderie that I have seldom seen. All of the lab members have been there for me at one time or another: Andy, Azra, Kate, Kevin, Lou, Marko, Noura, Robin, Sara, Sanjay, Vidya, Xiaobin and shouts to Ayman.

Mike Swain, Jim Firby and Alain Roy all had their parts to play starting me on my journey in Computer Science. Without them I would have never found it, and my life would have taken a very different course.

Many of my friends in Chicago have been instrumental in keeping me together through this process. Mike, Pam and Ligia have spent more years than anyone should have to listening to me complain as I pace in the living room when I get home from school. Katherine, Alain, Jose, Jen and Marty have all also done their parts. My Sensei Kevin, and my Sempai Dave have both helped to keep me in line. I want to thank my family, my mom and dad, for putting up with my constant waffling on when I was going to be done, my brother for being there when I needed him, along with my sister.

And finally, the love of my life, Sue. Sue has been there with me through the end of this process, putting up with ever changing end dates, near constant agonizing and self doubt. Without her, this work may never have been finished.

So to all the people listed here and the still more people who helped me who are not named, thank you all so very much. Your kindness and help are well remembered. Without you, I would not be here today. But here I am. Thank you.

Table of Contents

List of Figures

Chapter 1

Introduction

As the state of the art advances, more and more problems are being solved using computer vision. Vision is a very general purpose sensor. Humans use it for a wide variety of tasks, implying that computers should be able to use it for a wide variety of tasks. Unfortunately the implementation of vision on modern computers is a nontrivial and unsolved problem. However, great progress has been made using computer vision to solve specific tasks.

This dissertation examines combining these special purpose computer vision techniques to obtain a more robust and general purpose computer vision system by allowing it to choose which computer vision routine to use for different tasks. In addition, we show the validity of this framework by demonstrating it in a number of different complete systems. We are specifically interested in using computer vision to allow our systems to interact with the physical world. We found a number of commonalities between the information needed by the different systems indicating that a more general framework could be shared between them.

1.1 Computer Vision

General purpose computer vision has long been a goal of artificial intelligence researchers. From a theoretical standpoint the complexity of the vision algorithms is not a hindrance to obtaining the desired visual information from a scene. In the real world, however, we are bound by time and memory constraints which do not allow us the luxury of obtaining a complete world model. Vishvjit Nalwa defines general purpose vision as vision "without restriction to a particular task or domain" [72, page 25]. By not restricting ourselves, the general task becomes too hard to achieve. Researchers who have worked on developing general systems for solving vision tasks have found themselves either failing to solve their specific tasks, or failing to maintain the generality of their vision system.

The desire to use vision for specific tasks, and the difficulty of the general problem, led to researchers to develop task based, lightweight, and active vision routines, which we collectively call *special purpose* vision routines. Much progress has been made in the area of special purpose vision in the past decade. Many specific tasks have been solved using special purpose routines, from human computer interaction to robotic navigation. Unfortunately all of these tasks were solved with specific tasks and environments in mind, making them brittle and hard to reuse.

1.2 Using Special Purpose Routines More Generally

Looking at the research into both general purpose, and special purpose computer vision, we believe that there are problems with both approaches. On the one hand, special purpose routines allow us to build systems that work and rely on computer vision, however they break as the environment changes. On the other hand, true general purpose vision will not be attainable any time in the foreseeable future.

Active vision suggests an easier method of achieving general purpose vision: extracting only that information from a scene which is required for a given task. This alternate way of framing the problem proves to be crucial. Most systems that use computer vision have some information goal that they are trying to achieve from the vision system. If the system has a goal in mind, it does not need to extract all possible information from the scene, just the information it is looking for. By constraining it in this manner, the system can function, since it is using special purpose routines which can work reliably. However if, instead of simply relying on a specific special purpose routine, the system had a library of routines to pick from, it could pick the appropriate special purpose routine for the given situation.

This leads to our methodology for achieving general purpose computer vision: provide only that information that is required for a given task and given world state when requested, whatever that task and world state may be. Similarly to Nalwa's definition of general purpose vision, our methodology can be expanded to work for any task and world state. Unlike his definition, ours cares deeply about what those states are. Using this methodology this system actually adapts what it processes depending on the task of the system it is embedded in, and potentially on the state of the external world.

Stated another way, general purpose vision seeks to cover the entire problem space of computer vision. Special purpose routines cover only a small portion of that space. Figure 1.1 represents a special purpose routine covering only a small portion of the problem space. We seek to cover the problem space by using multiple special purpose routines which cover different parts of the space. By extracting different information for different problems we are able to cover the space.

It is the ability to change what information we are trying to extract from the scene that makes this different from special purpose routines. This dissertation examines

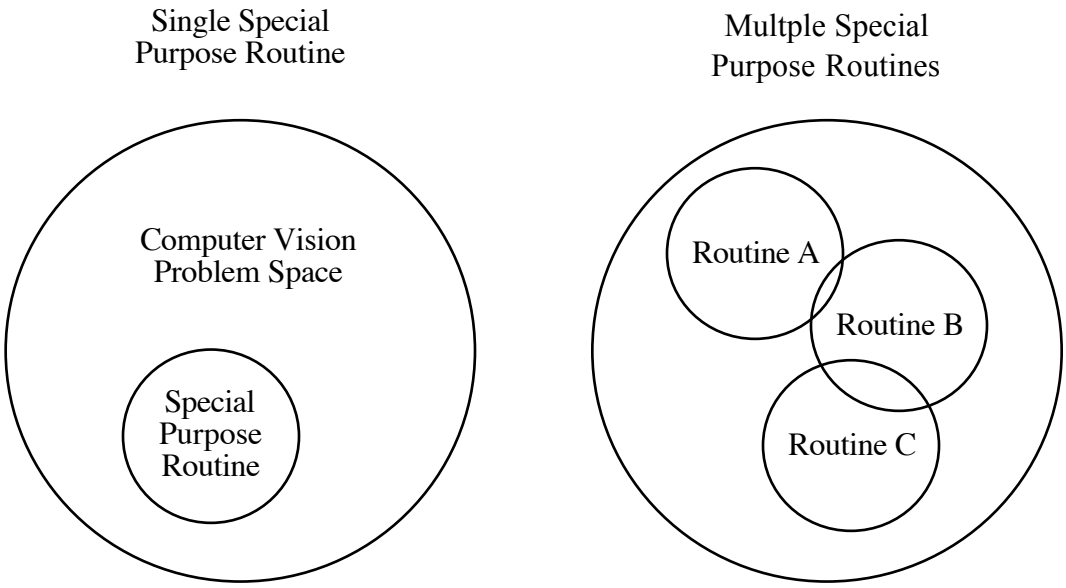

Figure 1.1: A Venn diagram representing the space covered by special purpose routines.

using different special purpose vision routines at appropriate times to achieve general purpose vision.

1.3 Thesis Statement and Contributions

1.3.1 Thesis

Reasoning about explicit constraints and using those constraints to choose between special purpose routines allows us to create a general purpose computer vision system which is both broad in coverage and robust.

1.3.2 Contributions

The main implementational contributions of the work described here are:

- Our vision system, Gargoyle, which provides a framework for constructing vision routines on the fly, which is required to allow reasoning systems to manage the vision routines. It also provides a modular interface for designing and reusing parts of vision routines.

- An implementation of existing vision routines in a robotic system, CHIP, utilizing this framework. This demonstrates the capability of the framework to implement proven systems.

- The Intelligent Classroom, a system designed from the ground up with reasoning about vision as a goal. This demonstrates the ability to reason with constraints about vision routines built in this framework.

- The Interactive Image Mosaic, a project quickly built using many existing routines, proving the ease of reuse that Gargoyle provides.

Generally, these systems embody the following contributions:

A system for filming people delivering a lecture. The initial work done for this dissertation focused on developing the ability for a computer to automatically produce a reasonable film of a lecture. This required the ability for the computer to reason about how to film the lecture, as well as the ability to see where the lecturer is. The system produces the film by deciding which part of the scene has the most relevant information and focuses the camera on that part. For example, if the lecturer is standing and speaking the facial features will be most important, so it will zoom in to his head. On the other hand, if he is writing on the board, what he is writing will be focus of attention and it trains the camera on the board instead. Special purpose vision routines were developed specifically for this task.

A system for choosing vision routines based on the current task. The Classroom has a number of other tasks for which it uses vision. In order for the vision system to be more general, we developed a system which picks the appropriate special purpose routine for the given task it is accomplishing.

A system for choosing vision routines based on the current environment. We developed a system for reasoning about which vision routines to use for which environments while filming the lecture. By expanding the capabilities of the Intelligent Classroom we were able to demonstrate a system which reasons about which vision routines to use for following a person, based on the specifics of the current environment. By providing the Classroom with different methods of tracking the lecturer it is able to produce much better films of the lecture. These contributions are described in the Intelligent Classroom, Chapter 6.

A system which utilizes previously built vision routines for novel environments and tasks. The Classroom contributions demonstrate a need to design the special purpose routines with an eye towards configurability and reuse. This allowed us to demonstrate the generality of our system by reusing it in a novel environment with similar tasks. This work is described in the Interactive Imagemosaic, Chapter 7.

Finally, we describe a methodology for reusing vision routines under different task and environment constraints. We feel that we have made some progress in the direction of utilizing special purpose routines for general purpose vision. We have taken the work done for this dissertation and distilled it to a set of "rules" for building a general vision system. These are found in Section 1.5.

1.4 Motivations

The goal of this research was very straightforward: *We wanted to build a robust framework that could provide vision for a wide variety of tasks.* We already had a robotic platform

that could accomplish a number of different tasks using vision based skills, but whenever we wanted to expand its set of tasks, we had to start over and construct new routines for the context implied by this new set of tasks. We wanted to build a system that would allow us to easily adapt the old vision routines to a new task or environment, or take parts of old vision routines and use them to build new ones for novel situations rather than starting again from scratch.

New vision systems are written for new tasks all the time; we just want to make that work available for different tasks in a reasonable manner. In order to see if the vision system was effective, in addition to building the vision system, it would need to be integrated with another system that used it.

Because of this we have two subgoals for this research: We needed to build a vision framework that would support the vision needs of our systems, and we needed to evaluate our methodology by building a real system that utilized this framework.

1.4.1 Build a framework for a general purpose vision system

Building the framework to enable this type of vision was an iterative process, but in the end it was clear that it needed to provide structure for building special purpose vision routines, and for switching between those vision routines as dictated by the changing context of the world.

Given our definition of general purpose vision, the ability to change which special purpose vision routine is running is critical. In order to do this the framework would need to be run-time reconfigurable, and controllable by the system that is utilizing the visual information. There are many ways in which the framework might need to be reconfigured. A new vision routine might be needed, or an existing vision routine might need to be reparameterized. Our system provides for both these types of reconfiguration, as well as

actually reconstructing given routines on the fly, although we have yet to actually use this ability in an automated manner.

We also wanted to create a framework that made it easier for vision routine programmers to build new routines. Reusing old code has always been a problem, and reasoning about the constraints of those routines from scratch each time can be cumbersome at best. One thing we found very useful was the ability to break up the vision routines into atomic units. This allows the programmer to reuse parts of old vision routines for new ones. In addition, if the constraints on a given section of code are well understood, then when those parts are used in a novel routine, the programmer has a start on understanding the constraints involved in that one.

Finally, by having a single framework take on all of these tasks, it becomes very easy to reuse the same routines in completely different situations which require similar visual information. In our experience we were able to reuse not only the same routines, but in many instances also reuse parts of old routines in new routines altered for the new constraints.

One interesting thing to note with this framework is that it does not do anything without a control architecture. If there is no information for the framework to extract from the scene, then it does nothing. This makes sense because the goals of the controlling system, along with the current state of the world, determine the constraints the system can utilize, and allow processing power to be used where it will be most effective. For our purposes, general purpose vision can be achieved only through grounding the vision system with goals. Vision for vision's sake is not only an incoherent goal, but also not possible in the general case—it needs to be grounded in a real system. Therefore, the vision system is actually the combination of the low level routines and the part of a higher level system that controls the sensing. Figure 1.2 shows how this framework is combined with a reasoner to

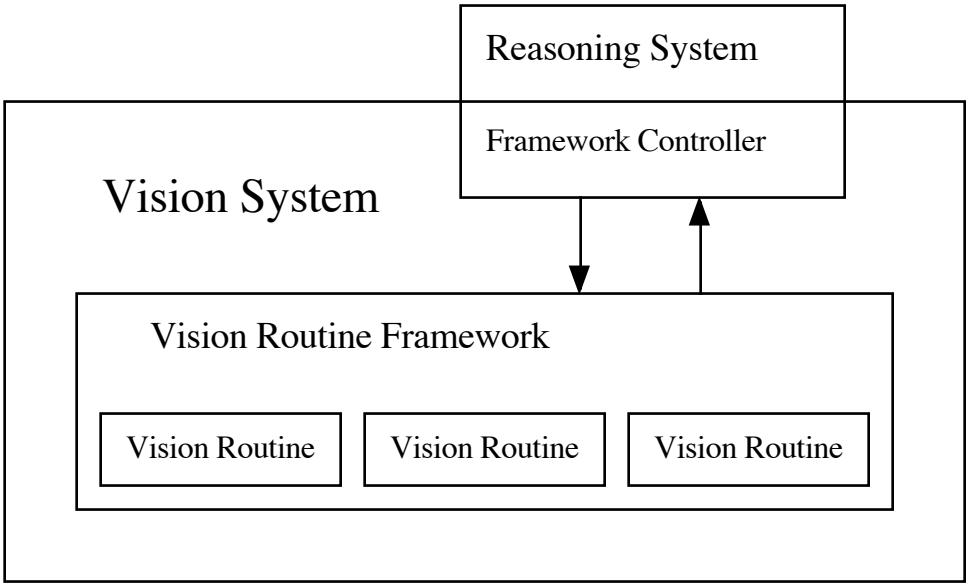

Figure 1.2: Our framework for vision routines must be combined with a reasoning system to provide a complete computer vision system.

provide a complete vision system.

This way of thinking about sensing is not new, it is similar to active vision as described by Bajcsy and Ballard [7, 10]. Active vision routines look for specific objects in the world, but only look for them in a specific manner. This framework, however, allows the system to determine not only *what to look for*, but also *how to look for it*. Since this framework relies on reasoning about the world that it lives in, we need to build a real system which can tell the vision framework what to look for and how to go about it.

Gargoyle

Gargoyle is the framework we built to meet these goals. It provides a number of mechanisms to achieve the three points discussed in this section but provides none of its own reasoning. Instead, it provides a communication architecture which a higher level reasoning system uses

to control all of the decision making that our vision system needs as shown in Figure 1.2. Section 1.4.2 talks about the reasoning behind this. This means that Gargoyle can not reconfigure itself, but is very amenable to reuse for completely different tasks, even ones with entirely different system architectures.

1.4.2 Build a real system

In order to evaluate the effectiveness of our methodology it was important to build a real system in which to run our vision framework. When we say real system, we are referring to a system that is designed around a task that we are trying to solve rather than trying to build a system around the problem that we are interested in. Given our goal of building a general purpose system, it would be very easy to fall victim to having every problem look like a nail to our hammer. In order to avoid this we were careful to define the needs of the system before applying our framework to it.

Defining the system that you are trying to build rather than simply designing the vision framework has the additional advantage of providing a structure in which to build the needed routines. Once a problem has been defined, you can determine what information is required to accomplish the tasks needed for that problem. This will then provide the task context which you can use to develop new special purpose vision routines. We have found that designing vision routines is much more straightforward from a conceptual to an implementational level when you have a real system to connect them to.

In addition to making the routines easier to construct, the system keeps track of the current context in the world. By explicitly representing the constraints the system can reason about them in the current context and tell the framework what to do. The vision framework can then assemble the special purpose vision routine that will work in the given context and return the requested information. This is critical because the system will have

Figure 1.3: Ayman using the Intellligent Classroom on the left, and the hardware for the Classroom on the right.

knowledge about what it is doing, and possibly about the state of the world, that the vision routines themselves should not need to know about.

Making a clean separation allows the special purpose routines to be written with an eye to reusability. Since the system programmer needs to understand the constraints on the special purpose routine, the routine programmer can use this information to make appropriate decisions about parameterization and how to break up the routine into atomic units. By making smart decisions here, the special purpose routines become much easier to modify for new tasks.

The Intelligent Classroom

To accomplish these ends, we implemented a system called the Intelligent Classroom. The goal of the Classroom was to provide a way for users to interact naturally with a computer system that controlled many aspects of a multi-media classroom. In order to meet this

goal, the Classroom can film the lecture, run a slide show, and do other tasks that the user might find useful. To accomplish this, the Classroom is composed of a number of interacting sensing, reasoning and execution systems. These systems allow the Classroom to listen to and watch the user.

By observing the user, the Classroom is able to reason about what the user is doing. By reasoning about this, the Classroom can decide what the user might need to do next and help him by doing it. This means that the Classroom has the goal of helping the user, and in order to do this needs to know what the user is doing.

The Intelligent Classroom was built at Northwestern University in the Intelligent Information Laboratory (the InfoLab). It is worth mentioning how the general InfoLab goals interact with those of the Intelligent Classroom project. The InfoLab mantra is that of *frictionless information access*. That is, the computer gives the user the information he wants as soon as he needs it, *without the user having to ask for it*. This is compatible with the Classroom's goal of doing what the user wants without the user having to ask for it in an unnatural manner. The behavior that the user is engaged in solicits the information or service rather than having the user break the flow of his task to ask for it. Of course this implies that the Classroom must keep track of the task that the user is engaged in.

Since the classroom has the specific long running goal of helping the user, it has a large number of sub tasks that it accomplishes to meet that end. Each of these sub tasks has specific information needs which the Classroom can fulfill by utilizing Gargoyle. This proved to be an excellent testbed for our ideas on computer vision, because the Classroom has changing goals that provide changing information needs. In addition the Classroom controls a number of environmental elements, and therefore provides a good source of contextual information about how the different vision routines should run.

Having a real system in which to test our framework is required; however to test

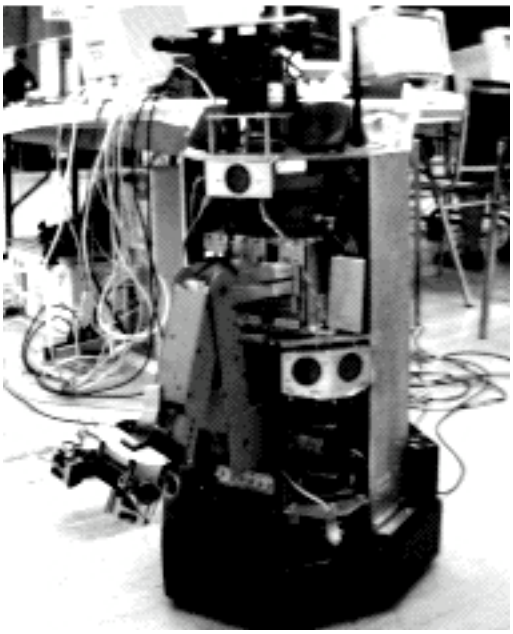

Figure 1.4: CHIP: A testbed for autonomous agent planning and vision research.

our aims of achieving general purpose vision, we really need to have more than one system in which to test it. This dissertation also looks at two other systems that used Gargoyle. They are a robot system, CHIP, and a real time Image Mosaic system,

CHIP

CHIP was a robot built at the University of Chicago. It was entered in the first AAAI robot competition and successfully located, identified and picked up a piece of garbage, and then located the appropriate trash receptacle and dropped the trash into the can, all using computer vision control techniques [37]. CHIP had a number of vision routines that it used to accomplish these different tasks. Unfortunately, each routine was specifically tuned for a given task. Worse yet, the routines ran off-board on different machines, causing a number of problems. These problems inspired the design and initial implementation of Gargoyle.

The robot competition tasks were ported to Gargoyle, and showed that this

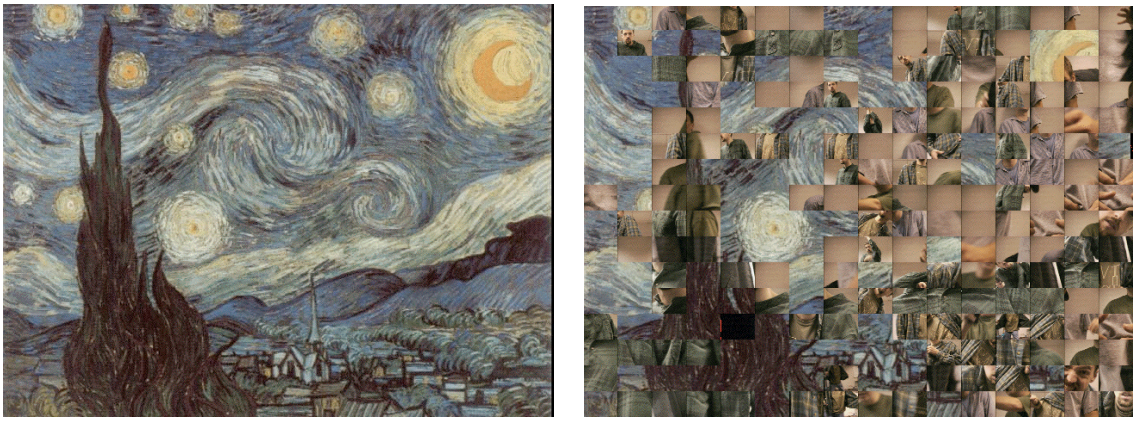

Figure 1.5: Starry Night, and a partially constructed Image Mosaic built from images of users interacting with the system.

methodology for computer vision was effective. Not only that, but these experiments showed that the same tasks could be accomplished with much less processing power, by being more selective about which routines to devote that power to. Using Gargoyle and integrating it with CHIP's planning system allowed us to take a number of different special purpose routines that used to be spread across a number of machines, and have them to coexist on the same machine. In addition, it led to the development of many of the vision routines that were later used in the Intelligent Classroom.

Image Mosaics

A further use for Gargoyle was an undergraduate student project that was planned and implemented in a single quarter using Gargoyle. The idea behind the Image Mosaic project was to have a system that would create a "copy" of another image by taking images from a camera, scaling them down, and tiling them together to make a mosaic. A fair amount of work has been done on generating mosaics from libraries of images, for example Finkelstein and Range [29]. Probably the first person to do this automatically on a computer was

Robert Silvers who used his system to create Photomosaics[TM1]

This system's information needs, while not of the same magnitude as the robot or the Classroom, are still significant. It needs to know where to place the new image and when to take the image, which it should only do when something interesting is happening.

The interesting thing that this project showed was that Gargoyle can be used to quickly implement a robust visual system to accomplish a specific task. In addition, it can suggest to the designer of the new system more possible solutions for that system. In this case, the designers did not know how they were going to determine when a person was standing in front of the Image Mosaic kiosk. Fortunately, a robust person tracking routine had already been written for a very similar situation in the Classroom. This allowed them to very easily reuse this "special purpose" routine which had never had this use in mind when it was initially written.

Finally, the mere fact of having moved the code base to the different platforms, and putting it to the different tasks, proves something of the thesis in terms of general purpose use!

1.5 Our Methodology for Computer Vision

Given the work we have done to build the systems required to meet out goals as set out in 1.4 we were able to make a number of generalization to create a methodology for building systems like ours. These ideas are drawn from our work on a number of different computer vision systems [37, 39]. This list can be thought of a high level set of instructions for constructing a general purpose vision system using our ideas on vision.

Our approach to vision allows us a bit more flexibility in designing our system.

[1]Photomosaic is a registered trademark of Runaway Technology Inc. and Robert Silvers. For more information on their methods see US Patent number 6,137,498.

In order to make use of this flexibility the system must know about the world in which it exists. By having the system understand something about the task that it is trying to accomplish and the environment that it is viewing, it can pick from a number of special purpose vision routines. By exploring this line of reasoning, and experimenting with building vision systems, we arrived at the following methodology:

- Use task based special purpose vision routines, as they have been proven many times in the past as viable, and provide a wealth of existing research which can be used to expand the system.
- Use a reasoning system to decide which special purpose routine to use when, based on the current goals of the system. That is, connect it to a "real system", one with goals which it uses visual information to achieve.
- Explicitly represent all of the constraints on the different vision routines, so that the system that is using the vision routines can reason about them.
- Break the vision routines up into atomic units which can be reasoned about and reused.

We will now briefly look at what each of these instructions means, before going on to look at the goals of the research that led us to these conclusions.

1.5.1 Use Task Based Special Purpose Routines

The first point tells us to use special purpose vision routines that are tuned to the specific tasks that will be performed. This will provide routines that work for the specific tasks that we want our system to be able to accomplish. This point is interesting, because it initially takes us away from the standard notion of general purpose vision. By using special

purpose vision, we are committing ourselves to task specific, or purposive, vision. This rule is essentially the statement that true general purpose computer vision is intractable and in some sense undesirable, so we are going to sidestep that problem, and concentrate on writing special purpose routines that we know how to write. Special purpose vision routines are discussed at length in Chapter 2.

1.5.2 Reason About Which Routine to Use When

The second point tells us that in order for the visual system to be general purpose it must be embedded in a reasoning system which decided which special purpose routine to use when. This tells us how to use our special purpose routines in a general way by making sure that the correct routine is used in the correct situation. We believe that in order to achieve general purpose vision, it must be embedded in the task and context in which the system lives. If it cannot look for everything at once, the system must decide what to look for. Therefore it must have some sorts of information goals. This implies that general purpose vision cannot happen outside of a specific task based system.

1.5.3 Explicitly Represent the Vision Routines' Constraints

The next point tells us to explicitly represent all of the constraints that make the special purpose vision routine work. Recall that special purpose vision routines rely on a number of constraints in order to be more effective. In many implementations these constraints are implicit, because the designers have just assumed certain facts to be true. Our general purpose system must know what those constraints are if it is to pick which routine to run when. If the system is going to be able to reason about how to process, all of the implicit constraints in the special purpose routines must be made explicit. Context and its use for reasoning are crucial and discussed more in Chapter 3. By understanding the context that

the different special purpose vision routines use, we can pick routines based not only on the information that they get, but how they get it as well. If we want our system to be able to operate in more than one context (and therefore be more "general purpose") we must be explicit about the context in which it operates.

1.5.4 Break Up the Vision Routines into Atomic Units

Finally, we should break up the vision routine into atomic units. This provides two major benefits. First, it is much easier to reuse the vision code for alternate special purpose routines. This proves critical because in order to create a general purpose system the system needs a large number of special purpose routines; at least one for each information task that it must accomplish. Second, it provides a starting point for representing the constraints on the entire routine. If the constraints on the atomic units are known, then the researcher has a starting point for representing the constraints for the entire system.

Additionally, this provides the groundwork for future systems to generate special purpose vision routines on the fly specifically tailored to the current situation. A system that could reason about hoe to sense in this manner, and generate new special purpose routines as needed would be truly general purpose.

1.6 Road map to the dissertation

The rest of this dissertation is divided up into three main parts. The first part gets into the details of how one uses the methodology given in Section 1.5 which themselves are simply an expansion of the thesis statement. Chapter 2 provides the details on designing special purpose vision routines so that they are amenable to being utilized in a general purpose vision system. Chapter 3 talks about how to reason about the different vision routines. Chapter 2 provides the background on the constraints that are used for reasoning

and should be read before Chapter 3. This part gives the theory of the dissertation. If you are interested in how to implement our brand of general purpose vision, but not our actual implementations, then read this part. This part is also helpful for understanding our implementations.

The next part of the dissertation deals with actual implementation, and follows the time-line that the research followed. They can be read in any order that the reader desires, depending on the areas of interest. Chapter 4 talks about CHIP, which was our initial implementation of a vision system. It did not use Gargoyle, but was the basis for many of the ideas in this dissertation. Chapter 5 gives the details of the Gargoyle system. It is probably helpful to read this chapter if you are interested in any of the implementation details of our systems. This chapter can also be read alone to learn about the design of Gargoyle. Chapter 6 gives the details of the vision routines implemented for the Classroom as well as the control structure used to control Gargoyle. Finally, Chapter 7 describes the Image Mosaic student project.

The last part finishes up with a look at related work in Chapter 8, and some closing thoughts in Chapter 9.

Chapter 2

Special Purpose Vision Routines

Section 1.3 introduced our methodology for building a general purpose computer vision system. To recap: *use special purpose vision routines; decide which routine to use based on the current constraints; explicitly represent the constraints of the special purpose routines so they can be reasoned about; and break the routine up into atomic units.* The first rule, use special purpose vision routines, dictates that these heavily constrained routines are the bulk of our vision system. The goal is to combine these special purpose vision routines to obtain general purpose vision. Given this, it is crucial to understand how these special purpose routines work.

There are a number of specific advantages to special purpose routines which this research leverages in order to achieve its results. For example, understanding why a specific vision routine works in a given situation allows the system to decide when it should be used. If the current routine needs to have a stable background, then the system must pick a different routine when the camera is moving. In addition there are a number of disadvantages to special purpose vision routines that need to be understood in order to prevent the complete system from becoming brittle. The example here is the opposite of the

previous example—besides knowing when a vision routine works, the system must also know when the vision routine breaks. This allows it to know when *not* to use a particular routine. Chapter 1 gave a brief overview of why special purpose vision routines are effective, but in order to understand how to integrate them into a cohesive system, a deeper understanding of both why they work, and why they fail, is necessary.

This chapter examines implementations of special purpose vision and examines the reasons why those techniques are effective. The key is to understand what it is about the targeted nature of the special purpose routines that make them effective; understanding the specifics of the particular algorithms is not necessary. By identifying ways of thinking about special purpose vision routines in general, it becomes easier to design new ones, and to understand when and how to use existing ones. In order to build a system that reasons about how it will sense, you first need to understand why the effective sensing techniques work.

To understand why special purpose routines are necessary, it is helpful to reexamine why general purpose vision is hard. In order to achieve general purpose vision, the system would need to be able to generate any information that might be needed in any possible circumstance. To pick a concrete example, the Intelligent Classroom needs to be able to know where the lecturer is at any given moment, with different possible lighting conditions. There are also times when it needs to be able to read what the lecturer has written on the board. To expect a single algorithm to be able to accomplish all of these tasks, and any others that we may decide that we need, is extreme. Even in the case of simply tracking the lecturer, writing a single tracking algorithm that can handle a large number of different lighting conditions, or other environmental changes, is quite difficult, and in fact can lead to fragmentation in the routine code as more and more complexity is added.

In order to get around these difficulties computer vision researchers have built

vision routines which only work in the specific conditions for which they were built. The next section looks at the advantages of these techniques.

2.1 Advantages of Special Purpose Vision Routines

One could imagine building a vision system whose sole purpose is to extract as much information out of a scene as possible [67]. This shotgun approach would ignore the fact that vision systems generally serve a specific purpose. That is to say, the information that is being extracted from a scene is going to be used for something, and only a small portion of that information may be used, or worse, an entirely new piece of information may be needed. Along similar lines, one could imagine a vision system whose sole purpose is to improve the extraction of a particular object [93]. This suffers from the same problem of not realizing that the information that is being extracted is going to be used for a purpose. Most current vision systems that provide information for use, either for reactive systems in tight control loops or for human consumption, are very much special purpose vision systems. Because this describes almost all vision systems currently being built, it is clear the these systems are effective to some extent. The question becomes why are they effective. The answer is that they constrain themselves to the problem that they are trying to solve, and in some cases constrain the problem to make it easier to solve.

Rather than trying to make more and more general systems, researchers have found that by constraining what the vision routine is able to do, they are able to get much more accurate results much faster. By looking at the problem, and identifying ways in which they can define the task more precisely, designing systems to solve that task becomes much easier. More to the point, there are many problems that cannot be solved in a general manner, so the researchers are forced to examine the nature of their problem to find just what they are looking for. Take for example a robot which picks up trash. When

looking for garbage to pick up, rather than try to look for garbage anywhere, the robot can look for small objects on the floor, which are much easier to define, and depending on the environment, are probably garbage. We call constraints like this, that is constraints on information the routine can extract from the scene, *task constraints*. Since the task that the system is trying to accomplish requires specific information, it can use routines that specifically extract that information. Task constraints allow the researcher to limit the capabilities of the routine, and therefore greatly simplify the problem. The small object finder mentioned above will not be useful for finding garbage everywhere in the scene, but if our robot can only pick up garbage on the floor, then the routine is completely adequate to the task. Additionally, by limiting the scope of the routine to just finding small objects on the floor we are able to write a routine which is much more effective and utilizes fewer computational resources. In general, task constraints allow the vision routine to limit the amount of information it needs to extract from the scene.

In addition to constraining the vision routine to only extract information needed by the system for the current task, the vision routine can be simplified further by making assumptions about the environment. In the trash collecting example above, the assumption is made that all small objects on the floor are trash. This *environmental constraint* must hold for the information returned by that vision routine to be accurate. If there are small objects on the floor that are not trash the routine will fail because it will assume they are all trash. The advantage that is gained by relying on the environmental constraints is that the vision routines are much simpler, both to write and in computational complexity. It is critical however to make sure that the vision routines are only used when their constraints hold.

In addition to simply taking advantage of existing environmental constraints, researchers often place constraints on the world in which they are working. One possible trash collecting example would be to color the trash a specific color so that simple color

tracking routines could be used to locate the trash. To give an example from the Intelligent Classroom, when watching a person give a lecture in the front of a classroom, the vision routine designer could decide to make sure that the background is unchanging. This requires that there is only one person in the scene, allowing a much simpler background subtraction based tracking routine to be used. In general environmental constraints allow the vision routine to be simpler by making assumptions about what it expects to see in the world.

By constraining the systems in these ways they become more effective and useful. The fact that they are being used to accomplish a task is critical to the design of the routines. Knowing what information is needed from the routines immediately gives the task constraints that can be used. Understanding the environment in which the routine is going to be used provides a starting point for deciding how to simplify the problem.

Effective systems in the past have succeeded by using strong constraints placed on the world by the tasks and the environments in which they operated. These constraints are either forced to be true by the nature of the task, or assumed to be true about the environment.

All special purpose vision routines use constraints that generally fall into one of these two categories: task constraints and environmental constraints. Task constraints are determined by what information the system needs to get from the world. Environmental constraints are determined by the current state of the world.

2.1.1 Task Constraints

Task constraints are inherent to the nature of the task that the system is trying to accomplish. In order to accomplish certain tasks the system will need certain things to be true about the information that it is receiving from its sensors. This can vary from getting a specific piece of information grounded to internal symbols, to needing directional information for tight

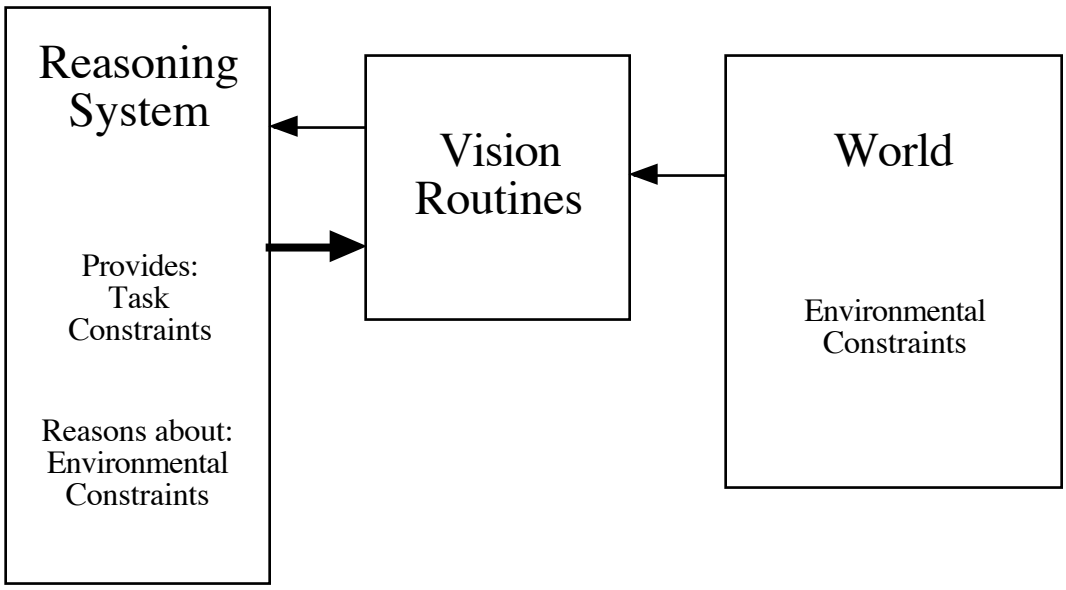

Figure 2.1: The constraints that a vision routine relies on come from the world and the task the system is accomplishing.

control loops, to saving specific images for later use by a human operator. By constraining the vision system to only get the necessary information when it is needed, the vision routine designers are able to make very strong assumptions when building their systems.

There are a number of ways that task constraints allow special purpose vision routines to be effective. First and foremost, they allow the routines to be simple. By keeping the routine simple, the researcher is more likely to be able to understand the assumptions that he is making about the world. In addition he is also more likely to be able to understand the consequences of any changes he may make to the routine. As computer vision routines become more complex, there tends to be loss of modularity. People have fought against this for years [87] but even if your system makes only the most generic assumptions there will be tasks for which those assumptions do not hold. Keeping the routines small and generating very specific information needed by the system, allows different tasks to be kept completely separate from each other and it becomes easier for the researcher to determine

when a specific routine is failing, and to give a starting point for investigating why it did not work.

Examining these simple routines it becomes clear that simpler routines, while providing less information, will be more likely to provide that information accurately, and in a timely manner. These ideas came initially from robotics research [52, 35, 63] where there is a strong real time constraint. For example, the robot needs to know what is in front of it, so that it will not run into anything. This is a strong task constraint which actually demands simplification of the vision routine. If the routine becomes too complicated, it will not be able to finish its calculations in time and the robot will run into something and damage valuable lab equipment. If the routine wastes time calculating extraneous, that will delay the response time.

The solution to the problem of calculating extraneous information is the final, and almost definitional advantage of task constraints. In order to accomplish a task, the system only needs specific information. We already know that you cannot get all the information all the time, but drowning the reasoning system with excess information is not only a waste of computational resources, but it is placing addition reasoning demands on the system trying to accomplish the task. This has been used to simplify the process of figuring out which part of the scene is represented by which part of the system's model of the world, the symbol grounding problem, by allowing the reasoning system to deal with one object at a time [21, 22, 26]. Rather than having to sort through all of the information coming from the vision system to determine which data corresponds to which symbol, the system is able to reason about only the information that it is currently interacting with. Agree and Chapman first used this activity directed representation in their work on Pengi [2]. Rather than represent everything in the scene, they only represent the objects with which the system is interacting. Task constraints allow the vision system to look for only the information that it needs. In the past this has been called active, task based, or purposive

vision.

Thus, task constraints tell the vision routine what the system needs to know and when it need to know it. This is only half of the problem though. Environmental constraints tell the vision routine what assumptions it can make to extract the required information from the world.

2.1.2 Environmental Constraints

Once the task constraints have been defined, the job of the vision routine is made much easier. However, the vision routine can still be simplified further, by looking at the environment in which it will be operating. For example, when the Classroom is tracking a lecturer, and it is known that the background is unchanging, a simple background subtraction tracking routine can be used. This constraint on the environment, that the background not be moving, allows a simple algorithm to provide fast and accurate information.

Environmental constraints come into being in two different, but related ways. The obvious way is that the constraint is fundamental to the environment in a given state, and can therefore be assumed to hold. In the Classroom example, the wall behind the lecturer will not change, and since the camera is stable, it is known that the background is stable, allowing us to use background subtraction. Another way of obtaining environmental constraints, is to impose the constraint on the environment. For example, if a robot has the task of picking up trash off the floor, the researcher can constrain all of the trash to be a specific color, allowing simple color routines to be used instead of more complex shape based methods [50].

These two methods of obtaining environmental constraints, inherent in the environment and imposing them on the environment, are related in a crucial way. Both of them are *assumed* to be true. Even if a constraint is inherent in the environment at a given

moment, it could change and cause that constraint fail at another time. It is critical to understand what assumptions are being made, to know when to use a specific routine. We talk about this more in Section 2.2.

In some ways, computer vision can be thought of as a method of figuring out the current state of the world. Therefore if we are able to make assumptions about part of the environment, extracting the other parts can be made easier. By utilizing more of these constraints the vision routines become faster and more accurate, and often much simpler to create. So once we are given this idea, how does one map from a set of constraints to a new vision routine?

There are many possible ways in which the environment can affect the vision processing. In the background subtraction example given above, the assumption of a stable background actually determines how the vision routine will be designed. In other cases the environment dictates how the vision routines are parameterized. For example, the robotic trash collector example requires clean results from an edge detector. If the results of the edge detector are too noisy, then its parameters will need to be changed, or the trash finder will fail.

This means that in addition to informing the design of vision routines, environmental constraints also affect the continued operation of those routines. If we are able to monitor these constraints, in addition to picking vision routines to use, we can modify them on the fly. We examine this more in Chapter 3. Next we are going to look at how these constraints fit into existing systems that use special purpose vision routines.

2.1.3 Successful Systems

Now that we have an idea of what makes special purpose vision routines work, it is helpful to examine some successful systems in terms of how they utilize task and environmental

constraints to make the vision task easier.

Polly: Small Task Needs Use Simple Routines and Constraints

One early successful use for special purpose vision was obstacle avoidance on mobile robot platforms. For this type of task real time performance for the vision system is the defining constraint. If the location of an obstacle is not reported in a timely manner the robot will not be able to avoid it. The first use of computer vision for this task was Ian Horswill's Polly system [52]. Polly's obstacle avoidance vision routine was used in a tight control loop directly with the robot's actuators. This meant that the vision system would only return exactly the information that was needed to determine which direction to go. In order to accomplish this, the vision routine would look up from the "bottom" edge of the image for the first edge that it intersects. It then assumed that this edge was an obstacle. By dividing the space in front of the robot into a few sections, this vision routine provided information about which direction was blocked and which was free. This information, along with the goal location, allowed the robot to decide where to go to get where it was going without hitting anything.

The task constraints are what allow this routine to be so simple. By specifying the information need to be only where an object is, the problem of constructing this vision routine becomes much simpler. In Polly's implementation, the task constraints even provide the metric that the vision routine must return in order to drive the steering mechanism. This is an ideal example of figuring out exactly what the problem is that needs to be solved to take as much work away from the vision system as possible.

Given a simple vision routine like this, understanding the possible environmental constraints that it relies on becomes much easier. For Polly there are two things which must be true about the environment for the vision routine to return the correct results. First,

the areas where the robot is free to drive must not have any visually sharp edges. This means that the floor must not be textured or have lines drawn on it. Second, all of the obstacles in the scene must provide edges that the edge detector can pick up. If this is not true, the robot will obviously not be able to see the obstacles to avoid them. These are nice constraints for the researcher, because it is easy to check whether or not they are true. They also show that it can be easy to think about when simple routines are appropriate to use. Of course sometimes the information that is needed by the system will be more complex than this, and not amenable to such drastic simplification.

Pfinder: Larger Information Needs Give More Complex Routine and Constraints

Some problems require more information than can be given by quite so simple a vision system. The Pfinder system [90, 91] developed at MIT's Media Lab was just such a system. Pfinder was the vision system that provided the information for a number of interactive projects at the Media Lab, including ALIVE. The ALIVE system provided a full body interactive experience. These capabilities required precise positioning information on the user of the system. In order to achieve these results the Pfinder system used some complex, yet highly targeted, vision routines. The two main techniques it used were modeling the person, and modeling the scene. Modeling the scene allowed Pfinder to know which parts of the image did not belong to a person. It remembered what the image looked like without the user in the image, thus allowing it to discard those parts of the image when looking for the person. Modeling the person is interesting because, given the task of finding where the person is in the scene, they were able to make a number of assumptions. They defined a person as a number of "blobs" with different color and spatial characteristics.

Pfinder made heavy use of task constraints. Because they were looking for a person, they knew a lot about the morphology of what they were looking for. They were

able to divide the person up into a number of blobs—two leg blobs, two arm blobs, a torso blob, and a head blob. These blobs had color attributes which were initialized depending on the user. In addition since the blobs were known to be specific parts of the human shaped user, they were constrained to be within a range of specific shapes and locations. When combined with the color information the system could give very accurate results for people in the scene. It is clear that with this more complex vision routine there are many more constraints, and figuring out what the environmental constraints are is going to be much trickier.

The environmental constraints imposed for Pfinder are considerably more numerous than for the Polly routine, because of the increased complexity of the vision routine. One constraint on the Pfinder system is that if it is using the model of the scene, the scene cannot change rapidly. Since it is relying on a relatively stable scene to find the person, its ability to successfully apply the person model to the scene will degrade if the scene changes rapidly where the person is not standing. This obviously leads to a large number of other environmental constraints; for example, the camera cannot move. If the camera moves, then the scene will change, and a new model will have to be made. There are many other constraints that need to hold to allow this complex routine to work so well. For example, once the color histograms for the blobs are learned, they cannot change—that is, the user cannot take off his coat. The key is that as special purpose routines become more complex, they rely on more assumptions about the world. When those assumptions hold, they produce the required information.

2.1.4 Special Purpose Routines Work

There are many more examples of successful systems that utilize special purpose computer vision routines, some of which are discussed in Chapter 8. It becomes clear that strong

constraints about the environment are crucial to effective vision routines. The most effective systems strongly tie themselves to their environment. Most special purpose vision routines utilize many environmental constraints. Unfortunately for our purposes, these environmental constraints are often used implicitly. The more reliance on constraints assumed about the environment a system has, the more careful the researcher need so be in knowing when to apply that routine. Heavy reliance on constraints makes these systems robust for exactly the situations that they were designed for, but they can become brittle, or useless, as the environment or task changes.

2.2 Problems with Special Purpose Routines

While heavy reliance on constraints makes special purpose vision, and therefore any computer vision, possible today, it comes with a cost. Oftentimes in the effort to obtain specific information out of a vision routine, it will become over-constrained to the specific problem. One cause for this is that the assumptions that the vision routine relies on about the constraints are implicit. When this happens those vision routines lose their robustness. This is because if a routine designer has missed a constraint that the routine relies on, then he does not know to stop using the routine when that constraint fails.

It is important to define what it means to be robust when we are operating in a world where all the vision routines that are effective rely on strong constraints. Robust can mean that it operates in a number of different conditions, or fails gracefully. A vision routine is often described as robust to specific changes in the environment. This is fine, but what if one of the constraints that is assumed to be true fails? In this case it would be nice to have the vision routine fail gracefully, that is to acknowledge that it is breaking down rather than simply respond with incorrect information.

Unfortunately none of these things can happen if the constraints that the vision

routines rely upon are implicit. A constraint becomes implicit when the designer has over fit a solution, a vision routine, to a given problem. This generally happens when a routine is tested on many positive examples, but few negative examples which cause the routine to fail. This is a common problem in machine learning. Making sure that this does not happen is essentially the work of debugging the vision routine by trying it out in as many of the potential situations it may encounter as possible. By iteratively debugging the routine like this, it becomes possible to miss a critical constraint that your routine assumes is true, and then when tried in a new environment the system will fail.

A robust routine is then one where the constraints are well understood, and is only used when the constraints hold. Thus, a vision routine can still be robust even if the possible situations that it operates in are limited, as long as its limits are well understood. So in addition to giving the right information—that is fulfilling the needs of the task constraint—the vision routine must also be able to operate in the specific situation in which it is being used.

This may sound redundant, but special purpose vision routines rely on many different constraints, and oftentimes these constraints are poorly understood. There are three major issues that come to the fore because of heavy reliance on constraints: the possibility of constraint failure, concerns about speed, and concerns about reusability of the vision routines. These problems cause researchers grief when trying to use them in a general purpose system.

2.2.1 Constraint Failure

Constraint failure is the most obvious problem with special purpose vision routines. If one of the constraints that is assumed to hold no longer holds, the vision routine will fail, sometimes dramatically. For example, in the background subtraction tracking vision

routine, the main constraint is that the background must remain fairly stable. If the camera moves for some reason, the background is no longer stable, and the tracking routine will fail miserably.

Understanding the constraints that the vision routine is relying upon is crucial for our general purpose vision system. Few special purpose vision routines have clean failure modes; if the constraints on which they rely no longer hold true, the routines simply return inaccurate results. Depending on the magnitude of the constraint failure, it might be difficult to tell that the vision routine is failing.

The main point here is that constraint failure affects all special purpose vision routines. Knowing when constraint failure occurs is critical if our vision routines are to be robust. Beyond not having clean failure modes, some vision routines do not even know when they fail. This does not mean that these algorithms are useless, they just need to be used in the correct circumstances. How that decision is made is left for the next chapter—what is important here is that if the constraints are not understood, no decision can be made about when they no longer hold.

Everyone expects vision routines to be robust. The fact is that vision routines are designed with specific constraints in mind. The trick it to keep these constraints in mind, keep them explicit, and make sure that the routine is not used when those constraints do not hold.

One constraint that deserves a little extra space is the need to have information at a specific time.

2.2.2 Speed Concerns

If a system needs specific information in a certain amount of time, then the vision routine has a constraint on it of how long it can take. Given computational constraints, this means

that decisions will have to be made about which special purpose vision routines are going to be run. Basically, a system cannot have all the information at the same time.

We were talking about making decisions about which special purpose vision routines to run, and only running them when they hold, but we also need to be careful to not run too many special purpose vision routines at any given moment. How many can run at once will depend on the complexity of the vision routine needed depending on the current environmental conditions, and on what type of constraints we can rely.

For example, in the Classroom, if the camera tracking routine is active, it needs real time information about where the lecturer is, so it can point the camera. If it decides that it needs to read something off the board, a processor intensive task, a decision needs to be made, since it cannot accomplish both of those tasks at the same time.

Another example is that of a robot picking up trash. If it needs to look for a trash can, it had better put itself into a safe state (stopped) before transferring vision computing power from obstacle avoidance and to finding the trash can.

These thoughts will get you a robust system for one task, or set of tasks, but a truly general system has one more consideration.

2.2.3 Reuse

With vision routines that have been heavily constrained for specific tasks and do not need to run in different situations, thoughts of reuse can easily fall to the wayside. We have already shown many instances where effective vision comes from constraining the routine in different ways. Re-engineering these routines to be effective under a different set of constraints can be very difficult. This is not a problem when solving specific tasks, but when designing a general purpose system the ability to reuse vision routines with relative ease becomes critical.

A general purpose vision system should be relatively easy to extend to solve new problems, and operate in new situations. We have already decided that our general purpose system will not be able to operate in any condition, only the ones that it knows how to operate in, and know when its constraints do not hold, but it should be able to be extended to new tasks and environments in a straightforward manner. We really want to be able to take the code that we have for one problem and apply it to another problem, but heavily constrained vision routines make this task much more difficult.

2.3 Moving On

So it is clear that in order to accomplish vision tasks, it is necessary to use special purpose vision routines. It is also clear that special purpose vision routines do not lend themselves to the general case by their very nature. How can we use this information to gain the benefits of cheap vision techniques, while avoiding the pitfalls of over-constraining the problem? Chapter 3 examines our solution to this question.

Chapter 3

Reasoning About Computer Vision

Chapter 2 outlined a number of advantages and disadvantages of using special purpose vision. It was this examination of special purpose vision routines that led to our current thoughts on how to build a more general purpose vision system. This chapter presents our ideas on what is required to build a general purpose vision system. We have seen many advantages of special purpose vision routines, however they have a number of problems as well. This chapter looks at our ideas on how to overcome these problems by combining multiple special purpose vision routines into a more general purpose system.

3.1 Our Methodology for Building a Vision System

We begin by examining in a bit more detail the methodology that we gave in the Section 1.3: use special purpose vision routines, pick the right one for the right conditions, explicitly represent all of the constraints, and break the vision routines up into atomic units.

3.1.1 Use Special Purpose Vision Routines

The first and most important conclusion we came to was that in order to have effective computer vision, we were going to need to use special purpose vision routines. As we have said, it is impossible to write one big vision routine that will obtain all of the potentially needed information, so the information gathering task must be broken down into manageable chunks. Vision research has shown that if computer vision is to work it must be special purpose vision.

Special purpose routines are easier to write, and understand. In addition they allow the researcher to understand when it will work. In order to solve its tasks, the system must be able to pick a special purpose routine appropriate for that task. In order to facilitate writing multiple vision routines, we built a vision subsystem, Gargoyle, which is described fully in Chapter 5.

By using special purpose vision routines we get the advantages of using task and environmental constraints, which allows the task to be possible, but we must be able to pick which routine to run when to overcome the problems of special purpose routines mentioned in Section 2.2.

3.1.2 Reason About Vision

In order to be more general purpose we use a library of special purpose routines rather than a single one. In our systems, Gargoyle provides this library. Different routines are used for specific tasks, and shut down when not needed. In order to know which routines to run when and when to stop running them, it is critical to have a reasoning system to make these decisions about the vision routines. Figure 3.1 shows a representation of a reasoning system selecting a special purpose routine from a library of routines. Given the disadvantages of special purpose vision routines, it is clear that if we want to create a general purpose

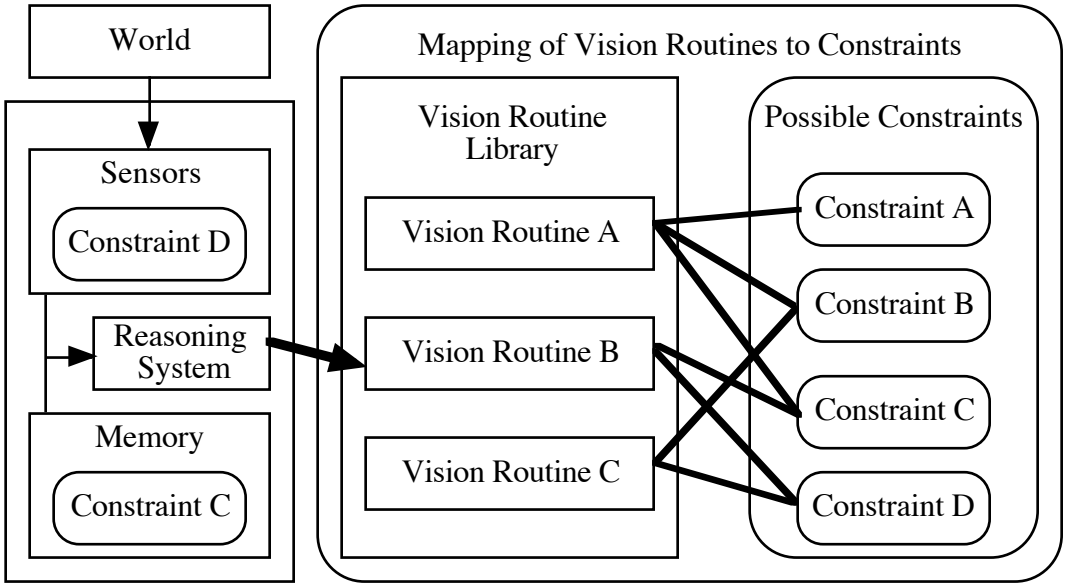

Figure 3.1: A diagram of a complete vision system built using our methodology.

system from special purpose routines, it is necessary to reason to decide which routines to run when.

This comes from the observation that the only effective vision routines are special purpose routines, and in order to have general purpose vision in a world where only special purpose routines work, the system needs to reason about how you sense. The system needs to know what to look for and how to look for it. This helps deal with the problem of constraint failure, both task and environmental. The system keeps track of the current constraint that hold either from memory, or directly from the world via sensors, as represented in Figure 3.1. Additionally, by designing routines that fit the specific tasks, speed concerns are taken into account.

Vision can become expensive if we try to do everything at once, but we do not need everything at once, we only need what our task calls for. Therefore instead of using everything in our arsenal of vision routines, the controlling system decides to only get the

information that it needs for its task by picking the appropriate special purpose vision routine. Note that the act of sensing itself has become a task with an information goal that can be reasoned about.

What form this reasoning system takes depends on the tasks that the designer is trying to solve with his system. For example, if the system is very complex with a wide range of information needs and internal goals one could imagine a top down system. This could take input from the vision subsystem and then reason about what to do next using traditional planning methodology. On the other hand, consider a robot which needs to get around the room, and has a small set of vision routines which control its movement directly. Deciding which routine to use in this case needs to be much more reactive. You cannot sit and think too hard about what you are going to do next when heading towards a wall. We use the Classroom's planning and execution system to reason about which routines Gargoyle should run. The Classroom implementation is described in Chapter 6, however that is not the only possible implementation of a control system for Gargoyle. Gargoyle was designed for a robotic environment and used as part of a class project, described in Chapters 4 and 7 respectively.

In order to support all this reasoning, the vision routine designer needs to be explicit about the constraints on which the routine relies.

3.1.3 Represent vision Routine Constraints Explicitly

From Section 2.2, the primary problem with special purpose vision routines is that of constraint failure. When utilizing special purpose routines we must be careful to make sure that the routine is only being used in the correct situations. Since the general purpose vision system that the we are designing is going to serve a higher level system, we have a big advantage in making sure that it is not being used inappropriately. That is, our reasoning

system knows something about the world and is in a position to make the decisions about which routine to run when. In order to be able to do this, the controlling system needs to know about the constraints that the special purpose routines rely on. Those constraints must be fed to the system explicitly.

This means that when the researcher is designing a new special purpose vision routine for the system to use, he must first understand all of the assumptions that the routine is making. He must then encode those constraints in a way that the system that will be using these routines can understand. Figure 3.1 shows a number of vision routines mapped to the constraints on which they rely. Understanding how to use constraints is examined more in Section 3.2.1.

Once the system understands about the constraints, it must be able to monitor them. Our special purpose routines need to either know when they fail or have a monitoring system that knows when they fail so that the controlling system can somehow learn that one of the constraints has failed. Figure 3.1 shows a reasoning system which finds out which constraints hold from its own internal memory, and its sensory data from the outside world. The system then switches to a more appropriate routine. The remaining sections of this chapter look at the problems of representing the constraints and reasoning about them.

The three "rules" we just looked at deal with some of the problems of special purpose routines, constraint failure and speed issues, and provide a way to get the constraint information that the controlling system needs; however, the problem of reuse still remains.

3.1.4 Break Up the Routine

Part of the reuse problem is solved by being explicit about what the constraints on the vision routine are. This allows the researcher to know when a routine can be safely reused. It would be nice however, to be able to reuse code at a lower level than that. We find

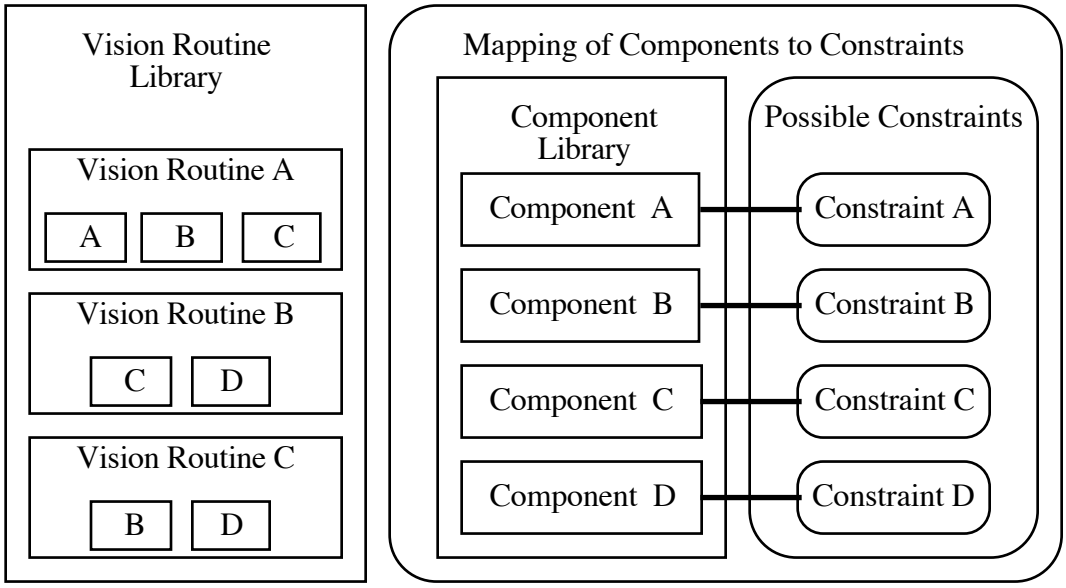

Figure 3.2: A diagram showing a library of vision routine components being used to make a number of routines.

that by breaking the vision routine up into "atomic units" we are able to very easily reuse components of vision routines, as shown in Figure 3.2.

Breaking up the special purpose routines into atomic units allows us to reuse pieces of vision code. In addition, breaking up the routine into atomic units can be aid the vision researcher in being explicit about the constraints. If constraints on individual units are known then it is much easier to recognize when a different unit can be helpful in a new situation. Figure 3.2 follows this notion through and lines each routine component up with the constraints that that particular part relies on. The complete vision routine then relies on the constraints that the parts rely on. Sometimes the interaction of the parts of the routines cause new constraints to be required.

At first glance, breaking up the vision routines appears to be more about good coding practice than building general purpose vision systems, but it it is critical for two reasons. First, it makes it possible to more easily extend the library of available vision

routines, and for a vision system based on special purpose routines to be general, it must be extensible. This fact was the inspiration for the modular design of Gargoyle described in Chapter 5. The second reason, however, is more fundamental. If the pieces of the routines are designed well enough, the reasoning system which picks which routine to run could also decide how to build the routine. If the constraints on each unit are well understood, it would be possible to allow the system to come up with its own special purpose routines tailored to a new task. None of our current implementations do this, but it is a research direction that we would like to follow in the future. Gargoyle actually supports this behavior through its module system described in Section 5.2.4.

The methodology we just looked at makes it clear that the step of explicitly representing the constraints relied on by the vision routines is crucial. The following section looks at that issue and talk about how to use constraints to allow the required reasoning to happen.

3.2 Understanding Constraints

The previous section describes how we came to our decision about how to implement our vision routines: use special purpose vision routines, pick the most appropriate one for the given situation, explicitly represent the constraints on the routines, and break up the routines. This section examines one crucial feature of our method of solving vision problems, that of explicitly representing and then using constraints. This is critical to the success of our systems, and is seldom considered when programming vision routines, and thus deserves a more detailed examination.

In order for a vision system to be more general purpose it must be able to sense any of the required information under the required conditions. Deciding which routines to run when comes down to figuring out which constraints currently hold and picking the

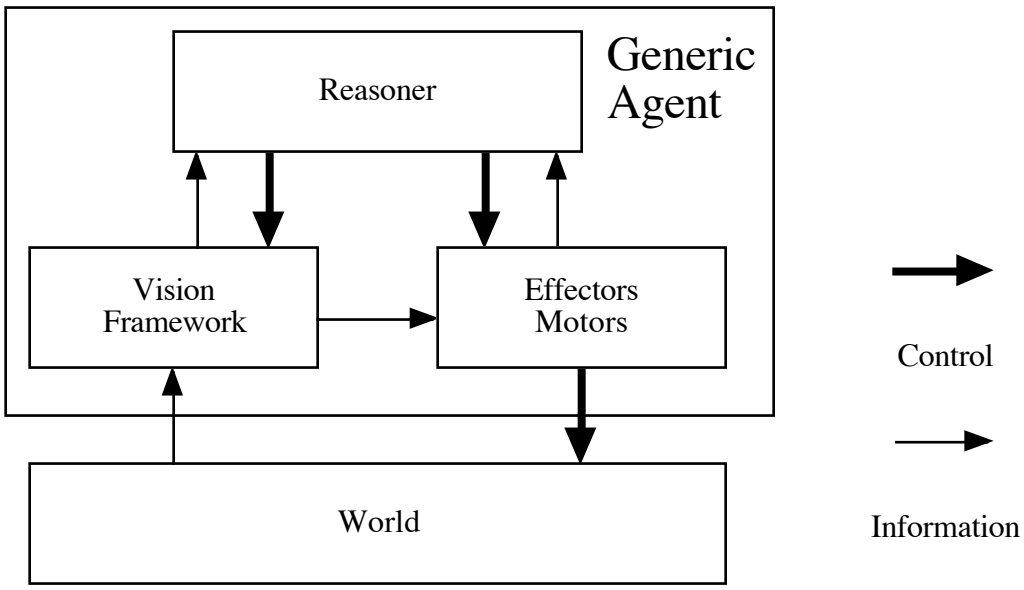

Figure 3.3: The component breakdown for a generic system which interacts with the world using a vision framework for sensing data.

routine that satisfies those constraints. This is because the constraints encode the relevant information about the routines. This means that it is crucial that those constraints are encoded explicitly for the controlling system.

3.2.1 Acquiring Constraints

Figure 3.1 shows how a reasoning system can be used to decide which vision routine to use when from a library of routines. Figure 3.3 shows how an agent designed with that in mind. The two parts that concern our use of vision are the vision framework which can be modified at run time, and the reasoning system which chooses which routine to run. The framework needs to be able to change which special purpose routine is running, while it is still operating in order to allow the reasoning system to change the routines when it deems it necessary. In order for the reasoning system to be able to control the vision system, it will have to somehow understand what the vision system provides for it to manipulate. At

a basic level this just means letting the reasoning system know which vision routines are available to it.

It immediately becomes clear, however, that simply knowing what is available is not enough. The reasoning system has to have some way of deciding when it is appropriate to use the different routines. This means that the reasoning system will need to know the constraints of the different vision routines, that is the conditions under which the different routines should be used. In order for the reasoning system to use these constraints, they will have to be explicitly represented somehow in the reasoning system.

The exact way that the constraints are represented depends on the reasoning system being used, however the need to have them represented remains. This being the case, the vision routine designer has a new responsibility, which is to clearly understand and explain the constraints that the vision routine uses. Others have noted the need to understand the constraints on the vision routines and only use them when appropriate [51, 89]. It is the additional requirement of articulating those constraints that is new.

Once the constraints are known, they can be integrated into the reasoning system. The specifics of how they are implemented are entirely dependent on the reasoning system that is being used. If a traditional planner is being used, they might be propositions that the planner can reason about. If, on the other hand, some sort of reactive behavioral system is being used, the encoding will need to be different. The behaviors will include the vision routines that are appropriate to that behavior. Therefore, the duty of making sure that the appropriate routine is run will fall to the behavior arbitration scheme. This is not unnatural as the behaviors themselves have their own constraints about when they should be run, so adding the vision routine's constraints is reasonable. It does however put limitations on the types of constraints that can be used. In a system that interacts with the world, the reasoning must be able to run very quickly so as to not interfere with the

interaction. This means that the reasoning about the constraint, that is checking whether or not they hold cannot be something that takes a long time. One consequence of this is that not all of the constraints of the vision routines will be implemented in the complete system. If computing the validity of a given constraint is so computationally expensive or resource use intensive that it limits the ability of the system to accomplished its task, then that constraint will need to be assumed.

Explicitly understanding the constraints is still necessary as they need to hold for the system to run successfully. The constraints which are not implemented in the reasoning system are simply not going to be used for decided which routine is used, they are still being used, just not for reasoning. The key is to remember why we are building this system. This system will be solving a specific problem. If reasoning about the constraints gets in the way of solving the problem then we have failed. This implies that only the constraints specifically needed to provide appropriate vision for the system that is using it should be encoded into the reasoning system. Before the constraints can be encoded, however, they must be known. This is what we mean when we say not to let the constraints on the special purpose routine be implicit. If all of the constraints are known explicitly, then the appropriate ones can be encoded into the system that will use that routine. If some constraints are implicit, they will not be encoded, and may affect the performance of the system. This can be summarized as three rules:

- List all constraints.
- Pick ones needed for picking routines for *your problem*.
- Implement only required constraints.

It is important to note that the constraints that need to be encoded into the reasoning system may change over time. As the vision routines are used in more and

different environments, implicit constraints that were missed when the routine was written for a specific environment may become apparent. The reason that a routine does not perform as expected when used in new environments could be that there was an error in the vision routine, but it is also possible that the designer simply missed another constraint. This "new" constraint on the system, can be incorporated with the known constraints, and the reliability of the whole system will improve. Additionally, this is another constraint that is being made explicit for the special purpose routine, providing more information about the routine for the next time it is integrated with a new system.

They key is to keep the constraints explicit so the system designer can decide which ones need to be encoded for the system to be able to decide about which special purpose routine to use when. We will now look at what types of constraints there are, to give an idea of what to look for when describing the constraints on the special purpose vision routine. We also examine how the different types of constraints are used.

3.3 Types of Constraints

In order to know how to represent and use the constraints it helps to categorize the types of constraints that you might need to work with. We have identified two main classes of constraints: task constraints and environmental constraints. These classes of constraints were introduced in Chapter 2. This section examines how to implement these constraints and how to utilize them once they are in place.

3.3.1 Task Constraints

Task constraints are fairly easy to understand. If you are picking a tool to accomplish a job, you need to pick the right tool for the job. Task constraints tell the reasoning system which tool to pick. They tell what information the special purpose routine is designed to

provide. That is to say, the routine is constrained to only give the information specified by the task for which it was designed.

Encoding this information for use by the reasoning system is straightforward. Given the system's current task it will need to get some information out of the world. This is the system's information goal. The system can compare the information that it needs with the information that the different routines provide. It then picks the routine that provides the required information. This means that representing the task constraints for the special purpose routines is a simple matter of providing the reasoning system with a correspondence between the special purpose vision routines and its information goals. The reasoning system has its information goals based on what it needs to know in order to accomplish what its current task. This goal task may come from a high level planning system, or user input.

To take an example from the Classroom, one of the tasks that it does is read an icon that the speaker wrote on the board. If the speaker draws an icon on the board, the Classroom needs to know what that icon is in order to use it. In order to read the icon it needs to know where the icon is. So the Classroom now has two ordered information goals. First, it uses a routine that tracks the speaker's hands. This will tell the Classroom where the icon is, as well as letting the Classroom know when the task of drawing the icon on the board is finished. Once it knows where the icon was drawn and the speaker has moved away from the board, it can zoom in on the icon and run a different routine for identifying icons.

Beyond just saying what information is needed, the task that the system is performing might constrain how long it can take to get that information. The time constraint is also a task constraint. This means that in addition to stating which routine matches with which information goals, the representation must also include a notion of how quickly the

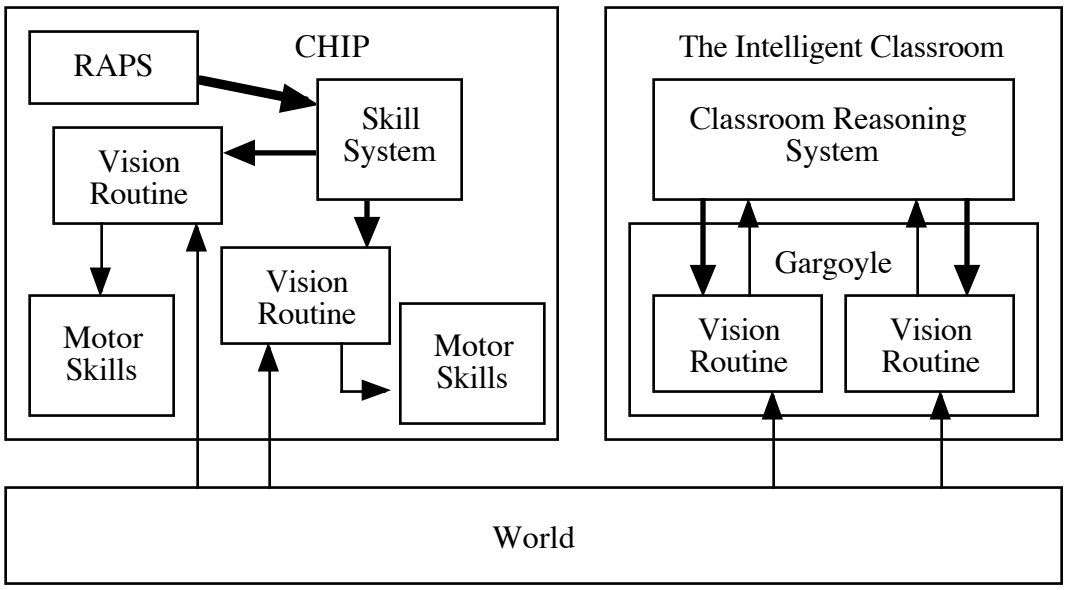

Figure 3.4: CHIP merely selected routines to use, whereas the Classroom updates how those routines run.

information will be returned, if that is required by the task. This way when the reasoning system is deciding which routine to run, it will only pick the one that gives the correct information in the correct amount of time.

Implementation Examples

The icon recognition example from above contains a great deal of deliberation about when to run the routine. As currently implemented in our systems, this decision making process actually happens at a much lower level. In our robotic experiments, each vision routine was tied to a specific behavior, or set of skills. In this case the deliberation, which is done by the RAP system, is merely picking which behavior to run. This means that the task constraints are encoded by having the RAP system activate a particular routine for a specific behavior. Figure 3.4 shows that each vision routine is tied with motor skills in a specific task. Chapter 4 gives a complete description of this system.

The Classroom's architecture, on the other hand, provides a richer method of deliberating about the vision routines. The Classroom is able to affect the special purpose vision routines with more precision than simply associating a single routine with a single behavior, or to use the Classroom's terminology, process. Figure 3.4 shows the classroom controlling vision routines in Gargoyle directly, using information passed to it by the routines. It still has the ability to link motor skills to vision routines, but it has more leeway in matching them. This implies that the task constraints need to be represented in a more complex manner. The Classroom does this by treating the vision routines as another skill. Sets of skills make up a single process, and are chosen depending on the task. This means that a different set of skills could make up the same, or more precisely, similar process. The skills have a context which determine which skill set to run for a given process. This allows the Classroom to then pick the appropriate set of skills based on the propositions that are in its memory. Chapter 6 describes the reasoning system that the Classroom employs as well as the skill system in more detail.

Complications With Task Constraints

The examples we just looked at are fairly straightforward, however there are a number of complications with task constraints that must be mentioned. The first one was hinted at in the Classroom example—it is the fact that there are instances where more than one vision routine matches with the given task constraints. If this is true, the tie can be broken best by examining the environmental constraints that the routine relies on. We will examine environmental constraints in Section 3.3.2. If these do not break the tie, then alternate methods of deciding which routine to use are needed. In general, when designing a special purpose vision routine the researcher will have a specific task in mind. This means that specific tasks will most likely have preferred vision routines, specifically, the ones which were built for that task. In the Classroom we represent this with a preferred ordering in

the skills that are used to accomplish specific tasks.

Having multiple routines for a single task can be a problem, but sometimes having one routine for a number of different tasks can also cause problems. Occasionally a routine may provide a number of pieces of information that are required for a given task. For example a person tracking routine might return the location of the head, hands and center of the person. When a task requires more than one piece of information from a given vision routine, reusing that routine for a different task might mean that useless information is being handed to the system for the new task. In the person tracking example, the hands and head locations will be extraneous for a task that only requires the location of the center of the person. This was one of the problems with general purpose vision that we solve by using special purpose vision. We use two separate solutions for this problem. First, since the new task requires less information, the vision routine has a new task constraint that it can take advantage of. This could be a good opportunity for a new routine or part of a routine, which is more carefully tailored to the given task. In Chapter 2 we saw that the more constrained the routine was, the more effective it often turned out to be for that task, thus the new constraints provided by the new tasks may allow a more specialized routine to be used. On the other hand, it may sometimes not be possible or desirable to create a new routine. In this case we simply structure the routine so that it only gives the appropriate information for the given task. If only some of the information is needed, the routine should be able to only return the requested information. This allows the routine to be reused in tasks with slightly different task constraints and information requirements.

One final difficulty is that a given task might require information from multiple vision routines. In the behavior case this problem is easily solved since the behavior should encode all of the vision routines that need to run. In a more deliberative system this can cause problems because it is simply harder to decide which ones to run. The solution that is implemented in the Classroom is to generate vision routines that combine all of the needed

sub routines into one larger routine that returns all of the needed information.

In this section we looked at how task constraints allow us to use task based vision. With the task constraints represented, the system is able to decide which vision routine to use to get the information it needs to accomplish a given task. Task constraints tell us what to look for. Environmental constraints, on the other hand, tell us how to look for it.

3.3.2 Environmental Constraints

Environmental constraints are constraints on the environment that the vision routine relies on to operate correctly. In Chapter 2 we saw how special purpose vision routines gain their utility from restricting the domain in which they operate. The environmental constraints are the result of that restriction.

With properly represented environmental constraints a system is able to choose the correct vision routine for the current situation, by knowing whether or not the constraints that a given routine relies on holds. In addition, the system can tune the parameters of a vision routine based on the current world state if it has the appropriate information about how the parameters are constrained by the environment. The question then becomes how does one represent the constraints that the routine needs.

The key to using environmental constraints to decide how to sense depends on a number of different factors. First, the implementation of the system making the decision will be a major factor, just as it is for task constraints. In more deliberative system, the constraints can be preconditions on the specific vision routine for it to be chosen. In a more reactive system they will need to be included in the arbitration scheme. Again, this should not overload the arbitration scheme because the behavior should only be used when the appropriate constraints hold anyway. The other factor is what the constraint will be used

for. We now look at some of the possible uses for constraints and how to implement them.

3.4 Using Constraints

We have seen that task constraints can be used by a system to decide which routine to run. Environmental constraints can also be used in this way. They can also be used to change how a routine runs, and both types can be used to know when a routine should stop running. This section examines the details of implementing a system that exhibits these behaviors.

3.4.1 Pick a Vision Routine to Run

Similar to how we use task constraints, environmental constraints are used to pick which vision routine to run. In addition to merely helping pick which routine to run, the environmental constraints help decide when to run the routine. Unlike with task constraints, where we just compared the information needs of the system to what the vision routine provides, the actual state of the world needs to be queried in order to determine which environmental constraints hold. This is a potentially serious problem. In order to know how to sense, the system will need to do some sensing. With the experiments we have done in the Classroom we deal with this problem in a number of ways. We now look at a number of ways of determining whether a constraint holds, giving examples from the Classroom where appropriate.

The easiest environmental constraints to deal with are those that the system already knows about. There are a number of reasons that a system could have knowledge of the world already, but if it has this, then checking the validity of a constraint is trivial. The Classroom has a model of the world which is updated regularly. This model can be quickly checked for specific facts. If the Classroom's memory does not have the required information, one of the following techniques is used to update the memory so the new

routine can be run.

In order to make it more likely that the routine we want will be able to run, we make as many of the constraints as possible things that the Classroom can enforce on its own. For example, when using background subtraction to follow the user, the background must not move. The Classroom enforces this constraint by holding still the camera it is using to watch the speaker. Since the Classroom controls the environment in which it exists there are a number of constraints that it can control in this way. Once the effectors have run to make the specific change, the Classroom's memory will be updated as appropriate.

Another solution is to have the constraints checked by vision routines which are currently running. For example, to be able to read an icon off the board, the area in front of the board must be clear, and the area where the icon was drawn must be known. Fortunately these can both be known by tracking the user while he draws the icon on the board. Since this routine needs to already be running to activate the goal of reading the icon the Classroom simply waits until the appropriate environmental conditions arrive. Once the correct constraints hold, the Classroom starts the new vision routine.

A third solution is to run a more general version of the routine, which discovers, as part of its processing, if the required constraint holds. For example, in order to get good locality on tracking a person, the Classroom uses edge information. This means that the background must be textureless. The tracking algorithm can be run, however, even without this information. While running though it can examine its own output to determine whether or not the edge detector is producing useful information. The Classroom tends not to use this technique when deciding which new routine to run, but uses similar reasoning for changing how a currently running routine is operating as discussed in Sections 3.4.2 and 3.4.3.

Finally, the system could run another vision routine to determine whether a

given environmental constraint holds. This is an extremely deliberative action and the Classroom does not exhibit this behavior. The reason we mention this at all is that there are instances when the system is starting for the first time when one could imagine that it simply has too little information about the world to know which routines to use. If this is the case, then it can actively query the world to discover the piece of information that it is lacking. An example of this would be a robot that used a free space finding routine. This routine requires that the floor be relatively textureless which is easily determined by running the free space finding routine. But another constraint is that the textureless area must not be an object. Once the routine is running this should continue to hold, but when the system is starting, the robot might be in front of an object. If this constraint was represented, the robot could use a disparity method to check that it is not right in front of something, or perhaps it could simply ping its sonar to get the required information.

The last example brings up an important point: the information about the world does not necessarily need to come from the vision routines. The system that is utilizing the information from the vision framework may have a number of other methods of finding things out about the world. The Classroom has information about its internal state, things like whether or not cameras are moving or which camera is feeding the vision framework, basic proprioception. If the current state of that Classroom element is not in memory the Classroom can sense it quickly. Similarly, one could imagine a number of additional sensors that the Classroom could integrate to provide more information for the vision routine. It has the ability to listen to what the user is saying, which is particularly useful for initialization. The robot also had additional sensing modalities. In addition to computer vision and some proprioception, the robot had sonars and a laser ranging device. This information can be combined at the skill level to improve the performance of the vision routine, or it could be accessed when a fact about the world that it provides needs to be known.

It is useful to be able to query the world to figure out what the required

information is, but it can be very expensive and in general should be avoided. The cost of switching vision routines could be large, especially if the routines have some startup cost. It is much better if the information can be extracted from routines that are already running. We have already seen that environmental constraints can be used to pick which routine is to be used, but they can also be used to determine how to parameterize a routine.

3.4.2 Change How a Vision Routine Runs

In order for special purpose routines to achieve their good results they often need to be parameterized for the specific problem to which they are being put. Instead of just reasoning about which routine to use, it is also useful to reason about how to parameterize the routine. For example, the threshold on an edge detector might need to be changed depending on the lighting conditions. The vision routine may be able to do this adjustment internally, but we found a number of examples where the constraint on the environment must be represented at a higher level. The methods of checking whether these constraints are true are the same as for picking a new routine, with an emphasis on using information to which the system already has access.

In order to represent these constraints and use them to affect the routine while it is running, the system must be able to monitor the running of its processes. It can then be informed of specific events to look for. These events can come from actions that the system takes. For example if the Classroom starts playing a video on the screen, it know that the part of the wall that the screen is on will no longer be static. Since the background subtraction routine requires that the background be static, that routine will no longer function properly. Since the Classroom knows where its projection screen is, it can actually change the parameters of the routine so that it only tracks the speaker in the portion of the room where the background is still stable.

Besides having the information come from external sources, a well designed vision routine will also be somewhat introspective. When the person tracker used by the Classroom is utilizing edges to more precisely determine the location of the person, it must have a relatively noiseless edge image. The person tracker does a number of morphological operations which it uses to know if the image is becoming too noisy. On the other hand if the edge detector is not sensitive enough, then the edges that represent the different parts of the person will not come out. If the routine is able to modify itself, it can change the threshold of the edge detector as needed. In order to maintain a clean separation between atomic units of the vision routines, the Classroom architecture requires that the person tracker simply informs the Classroom of the error. This allows the Classroom to take whatever action it deems appropriate, whether it be modifying the current routine or selecting a new one to run.

Finally, the system can simply watch the output from the routine and know when it has stopped making sense. One of the symptoms of a noisy edge image in the person tracker is having the different body parts that the system is tracking move into impossible configurations. Some of this problem is alleviated in the routines used by the Classroom by tracking and filtering the data, but sometimes there is simply too much noise. If this is the case, the Classroom notices that it is no longer receiving the expected information, and has the routine stop using the edge data. Beyond picking new routines, and altering how current routines are running, environmental constraints also allow the system to know when to stop using a routine.

3.4.3 Stopping a vision Routine

In addition to allowing the special purpose routine to start, the environmental (and task) constraints must continue to hold as long as the routine is operating. When one of the

constraints fails, the routine will cease to operate correctly, and should be modified or stopped. If the constraints are easy to check, the system can monitor those constraints to know when the routine is no longer valid. If some of the constraints cease holding, and the system cannot modify the routine as in the above example, the system must stop that vision routine. When a vision routine is stopped there are two options available to the vision system: it can start an alternate vision routine or decide that it cannot obtain the required information.

When a constraint that a routine relies on fails, there is a good chance that the routine will quickly fail as well. In order to prevent this from happening, the system must select a new routine. In the background subtraction tracking example, the user may walk into a portion of the scene that does not have a stable or known background. This will prevent the Classroom from tracking the user using this vision routine. In order to continue tracking the user, the Classroom must switch to a different tracking routine (in our case color based).

By not only using vision routines built for extracting the specific information required by the system tailored for the given world state, but also monitoring to make sure that that world state does not change while running, the system provides more robust processing. Monitoring the environmental constraints is advantageous to less deliberative systems as well.

A reactive behavior based system that knows when one of its vision routines should be swapped out is at an distinct advantage. Since a given behavior will not be able to operate when the constraints on the vision routine that controls it have failed, it is crucial that the arbitration scheme understand what those constraints are. These constraints are then another input to the arbitration scheme, and can cause a behavior change, just like any other input.

One important note here is that in our actual implementations, the realization of constraint failure happens on the level of internal changes. That is, it is based on things that it anticipates rather than things that have actually failed. The system does not detect when the routine is really failing, but rather when the state of the world dictates that the constraints that the routine relies on no longer hold, and therefore the routine will probably fail in the near future. This is related to monitoring for failure, but our work does not currently extend into the area of vision routine monitoring, another major area of research. However, the fact that our systems do have a set of constraints that they monitor means that they will fail gracefully. Assuming that the constraint that is failing has been implemented in the system, it will let the user know that the constraints have failed, if it has no other routine available to it. This provides a warning that things may begin to break, which is better than just waiting for it to happen.

3.5 Final Thoughts on Constraints

Given how many possible ways there are of checking whether or not a constraint holds, it is clear why only the constraints that are actually going to be useful to the system's reasoning should be implemented. It cannot be overstated though, that the constraints which are not implemented for the purposes of reasoning will still need to hold. This is why they must still be explicit even if they are not implemented in the system's internal representation.

All special purpose vision systems have constraints, and the vision routines should only be used in the correct situations. This is also true for a system built using the methodology provided here. What we are providing is not a general purpose vision system, but a way of getting to general purpose vision, and having a system that you can use on the way there. It is not general purpose vision, but provides a way to gracefully expand to fill the problem space. At any given stage, in order to use it, you must know where it will

and will not work. Then if it is needed in a situation where it currently will not work, it is easily extended to utilize the constraints in that new situation.

The next four chapters describe our implementations of these ideas. Chapter 5 describes a robotic platform which only partially utilized the ideas presented here, but revealed the need for a new platform for constructing vision routines.

Chapter 4

CHIP

Many of the ideas described in the first three chapters of this dissertation came out of our experience with a robotic platform, Chip. Chip was a robot built at the University of Chicago Artificial Intelligence Laboratory. One of the goals of this project was to produce a robot whose primary sensor modality was computer vision. Because of this, we needed a very robust set of vision routines to allow the robot to accomplish its tasks. To achieve this, we used vision routines which were highly constrained to the robot's tasks. These special purpose routines were effective at their tasks, but had many of the disadvantages described in Chapter 2. This led to the idea of reasoning about the constraints on which the special purpose routines rely.

Chip reasoned about vision in a very simple manner. Specific vision routines were tied to specific tasks. This meant that it only used the implicit task constraint to choose the routine to use. It did however point us in the direction of reasoning about explicit constraints, and the problems with implicit ones.

This chapter examines the work done on Chip as it relates to the ideas of reasoning about special purpose vision routines. As with all mobile robotics projects

there were many researchers involved in many different aspects of building the robot, and programming it to accomplish different tasks. There are a number of papers detailing the many aspects of building Chip and making him work—those detail can be found elsewhere [31, 32, 33, 34]. This chapter focuses on the design of the vision subsystems, and those details of the rest of the system that directly impacted the vision system.

4.1 Background

Chip was a mobile robotics platform, hand-built by researchers at the University of Chicago in the early 1990's. There were two main research groups working on Chip at the time: a planning and robotics group, led by R. James Firby and a vision group, led by Michael Swain. Both these groups came together to research how reactive planning, execution, and computer vision could work together on a mobile robot platform. The advantage of a robotic platform from the perspective of a vision researcher is that there are a large number of visual tasks that the robot might need to do, so there are many lines of research that can be followed. From the perspective of the research presented in this paper, obtaining more general purpose vision, this is an ideal scenario.

Because of the desire to do many different experiments with Chip, it was put to a number of different tasks, which required different visual data. One of the goals of the Chip team was to use computer vision as much as possible for sensory input and only rely on sonar for last second collision avoidance. This means that almost every task that Chip was put to required computer vision.

The primary task for all mobile robotics is that of navigation. The ability to move around the room and not hit something is not a given in the world of mobile robotics, and we used vision as our primary obstacle avoidance sensor. In addition to simply moving around we wanted the robot to be able to interact with people, so it needed to be able

to recognize when a person was in front of it. Finally, it had a manipulator which Chip could use to find, recognize, and throw away trash. This involved a large number of visual techniques. We even ended up combining some of these skills, doing things like having the robot pick up trash that a person pointed to.

Given the breadth of the tasks that needed visual input for Chip, it is clear that a more general purpose system, able to handle many visual information needs, was required. This is what Gargoyle was developed for, although it was not fully implemented until the end of the robot project. Chip used a number of disparate vision routines, one for each task that it could be put to. As we have seen in the previous chapters, in order to control these routines Chip needed a reasoning system that determined which special purpose routine to use when, which inspired the design of the Intelligent Classroom. We now will briefly look at Chip's hardware setup followed by the reasoning system and how it controls the vision subsystems before looking at the special purpose routines themselves.

4.2 Hardware

Chip was a complex, hand-built mobile robot on an RWI base. This section briefly looks at the subsystems that relate to computer vision. All of the vision processing was off-board, which was accomplished either through a tether or radio signals. This signal went though an MV-200 DataCube computer, which could compute standard vision operations on the full resolution output of Chip's cameras in real-time [58]. This filtered image was then distributed to a network of SparcStations which would do additional processing to complete the vision routines. Each of these Sparcs were used for a specific vision routine as shown in Figure 4.1. They would in turn hand data off to the robot's central controlling Macintosh which would assure that the appropriate skill received the required information [56]. Many skills resided completely on the robot, but because the vision portion of some

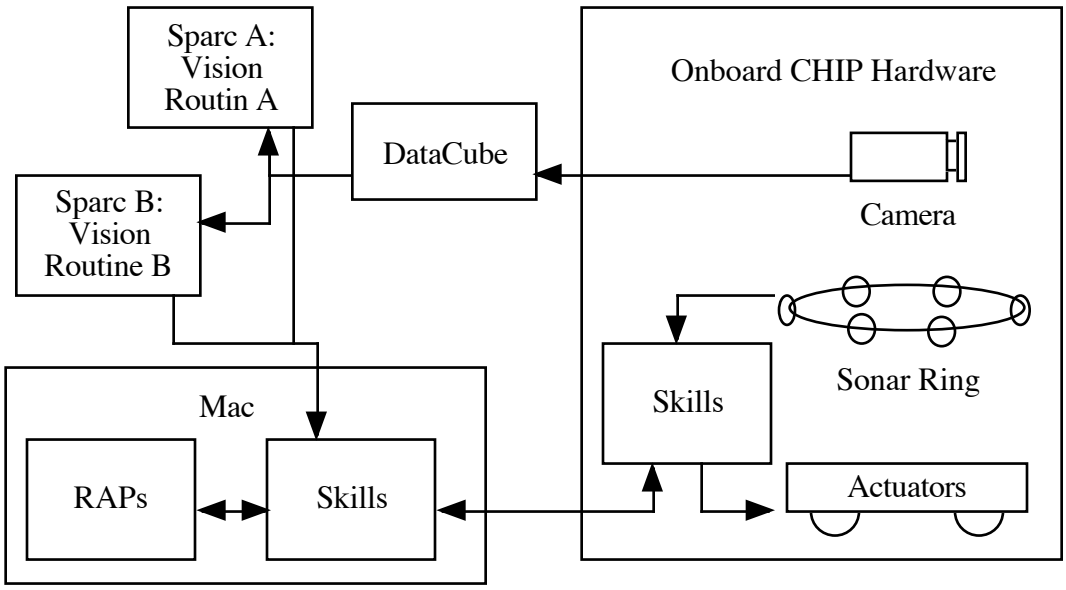

Figure 4.1: A representation of CHIP's processing spread out over a number of off-board computers.

skills were all off-board, their information needed to be routed through a central computer. This Macintosh was also tasked with the main planning duties, as discussed in the next section.

4.3 Animate Agent Architecture

While the planning and vision research aspects of the robot were under separate groups, there was a shared common vision about how sensing and acting needed to be tightly coupled. This coupling of sensing and action was codified into a system called the Animate Agent Architecture [34, 36]. All of the design goals stated in Chapter 1 were being formulated at the time we were working on the Architecture, although many of them had not yet come to fruition. The main point which was used was the most important: use highly constrained visual routines to provide real-time information for the execution system. In

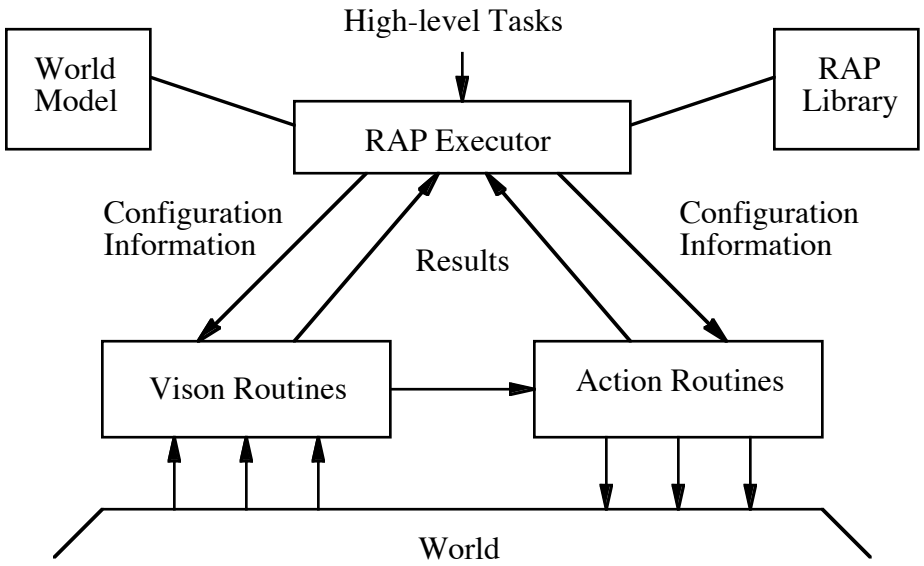

Figure 4.2: The high level breakdown of the Animate Agent Architecture as implemented on CHIP.

actuality the Architecture was designed for more general use than just computer vision, and any perceptual input source could be used.

On top of this all was a reactive planning system, the RAP system. In order to maintain a clean separation between the continuous inputs of the world, and the reasoner, the Animate Agent Architecture strived to maintain a leveled hierarchy, as shown in Figure 4.2. This was very similar to the generic architecture seen in Figure 3.3 and followed in the tradition of three layer architectures described by Gat, Bonasso, Kortenkamp and others [13, 44].

In the Animate Agent Architecture each perceptual routine ("vision routine" for this dissertation) was paired in a tight servo loop with action routines to allow CHIP to complete the task at hand. This type of control loop was called a *reactive skill*. These skills would then be enabled and disabled by the higher level RAP system. This meant that the RAP system would then be responsible for deciding which skill to use when, and therefore

which special purpose visual routine to use when.

4.3.1 The RAP Execution System

The Rap system, designed by Jim Firby, was the execution system which controled the Animate Agent Architecture [30, 32, 34]. In the Animate Agent Architecture, the Rap system made the decision about what to do when. It turned the reactive skills on and off as needed. The way it decided which skills were needed was through task descriptions called Reactive Action Plans (Raps).

A Rap can be thought of as a description of a number of possible ways of accomplishing a given task. Rather than being a single plan, it can be thought of as a collection of possible plans which are instantiated as needed by the current situation. When the Rap is invoked, the steps in the plan are chosen from a preexisting library as needed.

By choosing the steps of the plan when they were needed, rather than planning the entire chain of events beforehand, the plan was tailored to the current situation, rather than built on the predicted state of the world. While Chip only used a single vision routine for each task, it was this ability that allowed our later systems to choose which routine to use for a given task. By specifying which skill was used at run time rather than preordaining it at planning time, the skill that is most appropriate to the current context will be run.

Even though the plan was not instantiated until run time, there was still error from sensor noise and uncertainty, and this could cause inappropriate skills to be chosen, or even appropriate plans to fail. The Rap system dealt with this by making sure that each method had achieved its goal. If the goal was not achieved it could choose another method and try again.

The Rap system was ideal for reasoning about vision as described in this

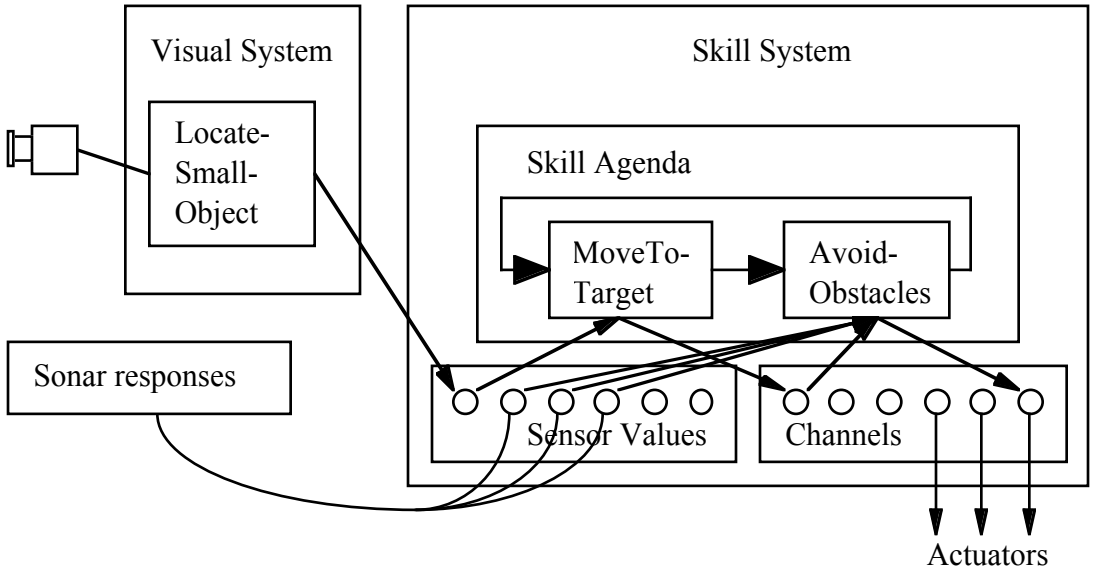

Figure 4.3: The skill system as implemented on CHIP.

dissertation, because it instantiated the vague plans based on the given situation—that is the current context. In this way the RAP system decided which special purpose vision routine, in a specific skill, to use for the next step for the current task.

4.3.2 Reactive Skills

The discrete steps that were chosen by the RAP system were called *reactive skills* [31]. These skills were not simply atomic actions that the system could take; the reactive skills tied sensing to action directly. A large body of work has shown that effective behaviors can be constructed in this manner [15, 17, 70, 51]. This tight coupling of sensing and acting was a central point of the Animate Agent Architecture, and is used in the Classroom work as well. On the robot, the skills continued to loop in a sense/act loop until the specific action that that skill was meant to do had been completed. It then signaled the RAP system that this step of the task was done.

The skills themselves were in two parts, an action routine and a vision routine. When the RAP system activated a skill needed for the current step in a given task, the vision routine started up, providing the necessary information to the action routine to move the robot. Figure 4.3 shows how this was accomplished through the use of channels. The sensory skills set global channels which the effectors then read. This provided a tight control loop, so the robot could react to any changes in the environment in which it was acting. Since the skill specifically tied the vision and action routines together, the vision routine was assured of providing the needed information to the action routine.

By constraining the skills to perform specific tasks, the vision routine which made up the sensing part of the skill could be easily designed as a task specific special purpose visual routine. Some of these routines were tied in a tight loop with action routines in the reactive skills on CHIP the real time constraint were critical, and the vision routines needed to be special purpose and lightweight in order to give the robot both correct and timely information. Other routines, like recognizing a person pointing, were decoupled from the system, and the skill had to wait for results to run. When the RAP system decided which skill to use in a given situation it took all of the given constraints into consideration to choose the appropriate skill. Thus, task context was included in deciding which vision routine to use implicitly, since the vision routine was tied directly to the skill being used to solve the task.

4.3.3 Vision Routines

The vision routines for the reactive skills on CHIP were designed to solve specific tasks, in order to incorporate them into reactive skills. These routines were constructed using many of the design considerations outlined in Chapter 2. Specifically, they were designed to make use of the constraints of the environment in which they were to be used. Unfortunately

these constraints were not reasoned about by the RAP system when choosing skills.

The visual routines on the robot were also inspired by Ullman's visual routines [87], utilizing early maps and visual markers to track objects in the scene. In order to provide the early maps in real time, the visual processing was spread over a number of off-board machines. A DataCube MV-200 image processing computer was used to prove real time, full-scale early maps. These maps were then used as needed on a series of SparcStations which would compute the needed information—for example the updated location of a visual marker—to pass on to the action routines.

These visual routines were designed to be as reusable as possible; however, in use a number of problems became apparent.

4.3.4 Problems

While the vision routines were designed to be reusable, the fact of the matter was that for every task a specific new routine tailored to the specific task was needed. As more skills were developed for CHIP to run, more SparcStations were added, one for each vision routine. The vision routine was implemented as a program which would constantly run on its own processor, waiting for the command to run from the RAP system, and all were constantly receiving image information from the DataCube in the form of early maps. Even though there was a common communication language for the vision routines and a common interface to the early maps, there was no common architecture for actually building the vision routines. This led to the conclusion that vision routines should be broken up into atomic units which could be more easily integrated and reused. In addition this led to us building Gargoyle, a general architecture for building the vision routines, which is described fully in Chapter 5.

In addition, since the special purpose routine and the action routine were tied

into a skill, it was difficult to modify the operation of that routine. Chapter 3 showed that for more general purpose vision, the reasoning system would need a good deal of control over the operation of the vision routines. Unfortunately, the RAP system lacked a good deal of knowledge about how the vision routines operated. It also lacked a coherent method for altering how the routine was running. The RAP system picked a skill to use based on the task it was currently completing, therefore only task context was used to make that decision. This meant that picking which vision routine to use was trivial, however it meant that the vision routine would only be useful for that specific problem.

In spite of these problems, CHIP successfully used computer vision to accomplish a number of difficult tasks. It also gave us the knowledge to go ahead with the design and implementation of Gargoyle, as well as providing the basis for the methodology to reason about computer vision used in the Intelligent Classroom.

4.4 Vision on CHIP

One of the goals of the Animate Agent Architecture was to provide a framework in which it would be easy to quickly expand the capabilities of the robot to new tasks. This section investigates the vision routines used in those tasks. Some examples of tasks that the robot was put to are described in Section 4.3. Since a goal of the Animate Agent group was to move toward using vision for all sensing tasks, and rely on sonar only for redundant safety, the first tasks that were needed were navigational and tracking tasks.

4.4.1 Obstacle Avoidance and Tracking

We used visual navigation to move around the room while not bumping into objects—obstacle avoidance—and we used tracking to follow or approach something. The first set of skills that were produced on the robot allowed it to move around a room without bumping

into things, or to follow a specific color, say, a person's shirt. Obstacle avoidance was provided by a variant of the Polly algorithm developed by Ian Horswill [51]. A number of tracking tasks were provided by Peter Prokopowicz, Michael Swain, and Roger Kahn: *watch, approach,* and *pursue*. Since only one tracking skill could run at a time, features appropriate to the target's characteristics needed to be tracked. Simple tracking features included motion, color, and work was begun on using disparity. A complete description of the tracking system can be found in [75]. Choosing the correct tracking routine for the given task was crucial. However, the RAP system's understanding of what the vision routines were doing was extremely limited. This meant that the specific routine would need to be chosen and parameterized ahead of time by the user of the system. With a limited set of tasks this was fine, but the set of tasks that the robot was being put to quickly extended beyond following a color around a room.

4.4.2 Object Identification and Tracking

The next major task that the robot was put to was the 1995 AAAI robot competition trash cleanup task. This task required a larger set of skills, and therefore more vision routines. We used vision to locate and identify the garbage in the room and then to locate trash cans in which to drop the garbage. Once the objects were identified, the robot would need to align with the object it wanted to pick up. This was a tight loop, and the vision routine used environmental constraints to target the object and track it quickly enough for the skill to move the robot. It is important to note that the environmental constraints for that routine were encoded in the vision routine, not at a higher level. This meant that the routine needed to parameterize itself as the constraint changed, and there were no alternatives for when the constraints failed. Our solution to this task is described in [33, 37].

This task was what really highlighted the need for a new vision system for us.

The trash collecting task took a number of steps: look for trash, pick up the trash, look for a trashcan, and thrown the trash away. Each of these steps required a unique vision routine, even though the actual work that they did was very similar. The trash identification step was fairly expensive, but was a combination of many standard vision operators, some of which were used in the next step. Tracking the selected piece of garbage to servo toward it required a very fast sensing loop, so only a small portion of the scene could be sensed. This meant that a new routine was needed, even though much of the vision work had been done in the previous routine. This was because we were unable to extract specific pieces easily from other routines, and image location context was not saved externally. The trashcan finding routine suffered from this same fate. Looking for the trash can also used the location constraint to figure out where to do the expensive Hausdorff matching, but again that was encoded into a complex two step visual routine.

4.4.3 Person Tracking

At the same time the robot was picking up garbage, other researchers were working on making it be able to see people. Since we wanted our robot to interact with people, it was critical that it had vision routines to track people and locate what they were pointing at. This culminated in the Perseus system designed by Roger Kahn [57]. This was a very complex multi-tiered system that tracked a number of different features and actually contained its own memory for tracking and determining when certain states in the image had been reached. This allowed it to recognize when the person made certain gestures [55]. For example, one of the tasks it could accomplish was to wait for a person to point to a can. It would then extend a cone out from where the person was pointing, to determine if a can was present where the person was pointing. It would then attach a visual marker to that can, and report its location to the Rap system. A complete description of the Perseus system can be found in [59].

Perseus was extremely effective at what it did, but it was not amenable to reuse. The next section looks at a task where we used Perseus as a sub-step in another task. Unfortunately it really lived in its own world. Once it started running, the rest of the robotic systems needed to wait for it to report back before doing anything. It took control of all of the vision routines it contained, but then didn't provide a strong connection to the system that it was being used for. This was another lesson learned for Gargoyle: leave as much reasoning as possible to the higher lever reasoning system.

4.4.4 Robotic Waiter

Both Perseus and the robot trash cleanup tasks were completed at about the same time. This led to a class project which combined these into a novel task: getting soda for people. In order to do this, the class simply combined the vision routines which had already been built for the other tasks. It would track the user until he had pointed at a can. Once the user selected a soda by pointing at it, the robot would move to that can and pick it up using the same routine from the trash cleanup task. Finally, it would approach the known location of the user to offer the can to him. A complete description of this project can be found in [43].

This new combination of vision routines appears at first glance as if it should be simple. Unfortunately due to the nature of the vision routines used for the other tasks, this took the entire class a number of months and a good deal of new vision code to glue all of the parts together. This work, done by Shannon Bradshaw, David Franklin, Mazin As-Sanie, Mark Langston, Jiayu Li, Alain Roy, and myself, produce an interesting new behavior, although the work involved for such a relatively simple adaptation of preexisting code was large. Chapter 7 shows another student project which utilized existing vision code in a much more straightforward manner thanks to improvement made from our experience

with the Waiter project.

4.5 Obituary

The robot provided a vibrant platform for vision research, and started much of the work found in this dissertation. Unfortunately, once the Intelligent Classroom project was under way, we moved to Northwestern and Chip did not follow us. The first task to which Gargoyle was put was a reimplementation of the Perseus vision routines. This proved to be quite successful, and led to further research on Gargoyle. Due to Gargoyle's portable nature and more efficient use of resources it could also be run on a laptop, allowing the computer vision to be moved on-board the robot. This solved a large number of problems for Chip, and proved the efficacy of Gargoyle. In fact, many of the Gargoyle routines built for the robot are still used in the Classroom. So, while Chip does not live on as a research platform, the research started there lives on in the Intelligent Classroom, through both the Animate Agent Architecture, and for providing the initial development on Gargoyle. The next chapter explores our current implementation of Gargoyle.

Chapter 5

Gargoyle

The Animate Agent Architecture dictated a clean separation between the low level sensing and motor details from the high level reasoning system. As the vision routines became more complex, this separation became extreme. The robot would request that a vision routine run, for example to watch for a person pointing, and the vision routine gave back what the person was pointing at. This structure allowed the routines to be effective; however, they ended up not being generally useful.

The routines were special purpose and the reasoning system did decide which routine to run when, but they lacked a number of other properties which would allow us to implement our methodology for general purpose vision. First, it was extremely difficult to reuse the vision routines. They were heavily paramaterized to the tasks for which they were built. Building new routines from the old ones required pulling out parts of complete programs, and figuring out how the parameters interacted with the new routines. Second, the static routines which the robot used were actually separated *too much* from the reasoning system. In order for the routines to serve in a more general purpose manner, the reasoning system must be able to alter how the vision routine is running at a high level. Unfortunately

on the robot, the only access that the reasoner had to the operation of the routines was to turn them on or off.

These concerns led to the design of *Gargoyle*. Gargoyle was designed to provide a unified structure in which to perform active vision. The Gargoyle framework was developed as a system that provides a framework for achieving the general purpose vision as described in Chapters 2 and 3.

In terms of our methodology, Gargoyle provides the following capabilities. It allows researchers to easily build and test special purpose vision routines by providing a set of reusable vision components and a framework in which to build them. It also provides an environment in which the vision routines can be dynamically reconfigured for the changing contexts. This allows a reasoning system to connect to Gargoyle and change which vision routines are running and how they are running as needed. Since Gargoyle does not provide any reasoning capability of its own it also does not provide a method for representing the constraints on the routines—this is left to the reasoning system. This chapter examines Gargoyle's design and its operation. Implementation details of specific routines are given in Chapter 6 for the Intelligent Classroom and in Chapter 7 for the Interactive Image Mosaic.

5.1 Design

Section 4.3.4 discussed a number of problems with the vision system utilized by the robot. Gargoyle was created to overcome these problems, and led to the following design requirements:

- Enable the dynamic creation and reconfiguration of vision routines specialized to the current context.
- Enable code reuse and reduce debugging time.
- Foster sharing between researchers.

By providing the capability to dynamically create and reconfigure vision routines, Gargoyle enables reasoning about vision routines. It does this by providing a runtime environment in which the vision routines operate. In order to allow it to be used by different reasoning systems it exposes a number of control mechanisms which can be used by those systems. These mechanisms allow the reasoning system to tie its internal constraints with specific vision routines. This way, each vision routine can have its constraints explicitly represented.

Since the vision routines all operate in the same environment, they have many similar properties. Gargoyle takes advantage of this and also provides a development framework which allows the code from different vision routines to be easily reused in new routines. This simplifies the sharing of vision routines among researchers using the same framework, since the the researcher is able to use the parts of the routine that are relevant to his research. This is accomplished by following our rule: break up the vision routine.

So Gargoyle is a framework that allows researchers to implement vision routines which can then be reasoned about. Before getting into the details of how Gargoyle works it is helpful to look at two specific features Gargoyle must provide: the ability to break the vision task up, and control mechanisms to connect it to a reasoning system.

5.1.1 Breaking Up the Vision Routine

Our previous experience with Chip showed that large monolithic programs are generally difficult to test and debug. When writing large vision routines, elementary processes are often embedded deeply in the code making them difficult to use and reuse. In addition, there may be interactions between the different parts which become difficult to understand. Thus, pieces of code are repeatedly rebuilt for use in other routines when similar vision techniques already exist. In order to help ameliorate this situation, vision routines in Gargoyle are

broken up into logical elements, called *modules*, which can be exchanged at runtime. These modules are combined into a *pipeline* to form a vision routine. Breaking up the routine in this manner provides a number of advantages.

First, by allowing the pieces of the vision routine to be dynamically changed, Gargoyle provides a mechanism which allows the controlling system to change which routine is running when

Second, breaking up the vision routine into basic modules, not only encourages code reuse, but also provides a simple mechanism for the sharing of algorithms between researchers. The modules are designed to be incorporated into new vision routines, so they are easily incorporated into the work of other projects, as well as for new routines in the same project. Another advantage this provides is that it allows the researcher to take only the parts of the routine that he wishes to use, rather than taking the entire routine.

Also, breaking up the vision routine makes it easier to explicitly represent the constraints for the vision routine. This is because each of the modules will have constraints that it relies on in order to operate. Representing the constraints of the modules rather than the entire routine is a much easier task and provides a stepping stone for representing constraints for the entire routine. Since a vision routine is just a pipeline of vision modules, it will rely on the constraints of all the modules. There will be additional constraints due to interactions between the modules, but this provides an excellent starting point.

Many of the constraints on the modules are exposed through parameters. This allows a reasoning system to change how the routine is operating, as discussed in Section 3.4.2. In addition to these simple changes, the use of module pipelines in Gargoyle allows a vision routine to be dynamically modified in non-trivial ways. The operation of the modules of a pipeline may be modified while they run, or the actual pipelines themselves can be changed by swapping out one module and swapping in another. This means that if one of the

modules begins to fail, the inadequate module can be swapped out and replaced by another one. Gargoyle also allows the parameters of the different modules to be changed during operation. This allows the higher level system to fine tune the functioning of the modules to specific environments.

5.1.2 Providing Control Mechanisms

While Gargoyle provides many of the elements required for reasoning about computer vision, it does not provide the reasoning system itself. This was done intentionally in order to allow Gargoyle to provide more general purpose use. Reasoning systems are often tailored to the specific tasks that they accomplish. If Gargoyle implemented its own reasoning system, it would necessarily limit the range of systems it could be used in. This is because many systems which require vision use their own method of reasoning. Forcing Gargoyle's reasoning system on every system that wanted to use it would greatly reduce its usability. The lack of a system internal to Gargoyle to reason about vision in Gargoyle is critical also because utilizing the constraints to reason about vision requires a good deal of knowledge about the world. An internal reasoning system would necessarily be unable to access all the other information sources available to the complete system which would lead to much more complex interactions between the high level reasoning system and Gargoyle.

Because of these issues, Gargoyle provides mechanisms which allow it to be controlled by an external system as shown in Figure 5.1. Combined with a reasoning system Gargoyle provides a complete vision system.

5.2 Implementation

Gargoyle is an active vision system that is runtime configurable, designed to meet all of the goals laid out in Section 5.1. This section describes how Gargoyle was implemented and

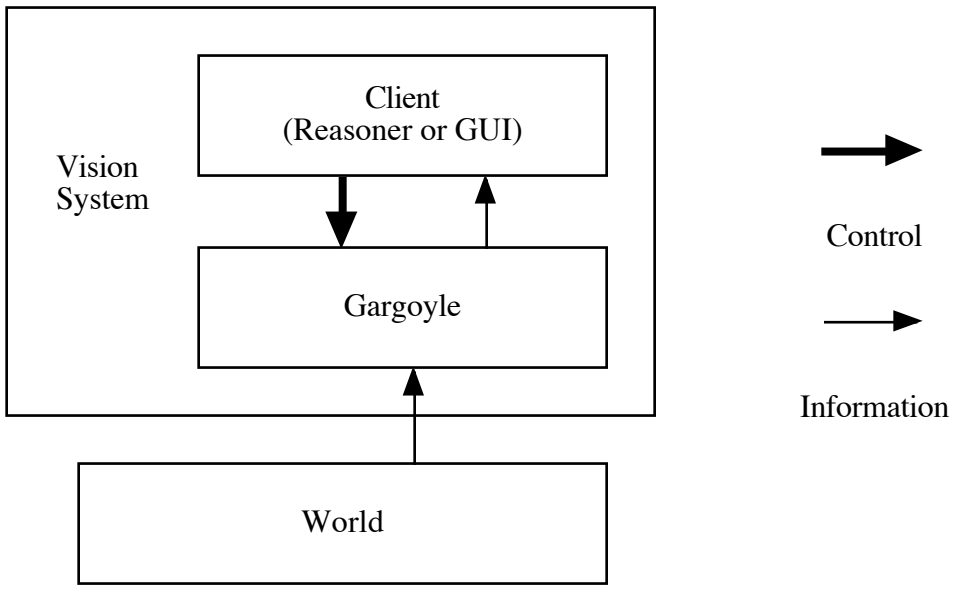

Figure 5.1: Gargoyle when used as part of a complete vision system.

how this implementation meets those goals.

5.2.1 Overview

Gargoyle accepts commands from a remote client, which could be the reasoning element of an agent or a manual client run by the user, as seen in Figure 5.1. The client constructs pipelines and manipulates them by sending messages to Gargoyle. The reasoner controls the sensors (like the cameras), manipulates the pipelines, and interacts with the world. For example, as a speaker moves across the room while being filmed, the client uses the information that Gargoyle passes to it to have the camera follow him. Gargoyle returns requested information continuously to facilitate tight control loops.

The Gargoyle system meets its design goals by delivering an integrated framework for designing and using modules. An example of a module might be an edge detector or a color histogram back-projector, as in Figure 5.2. The client requests specific vision routines

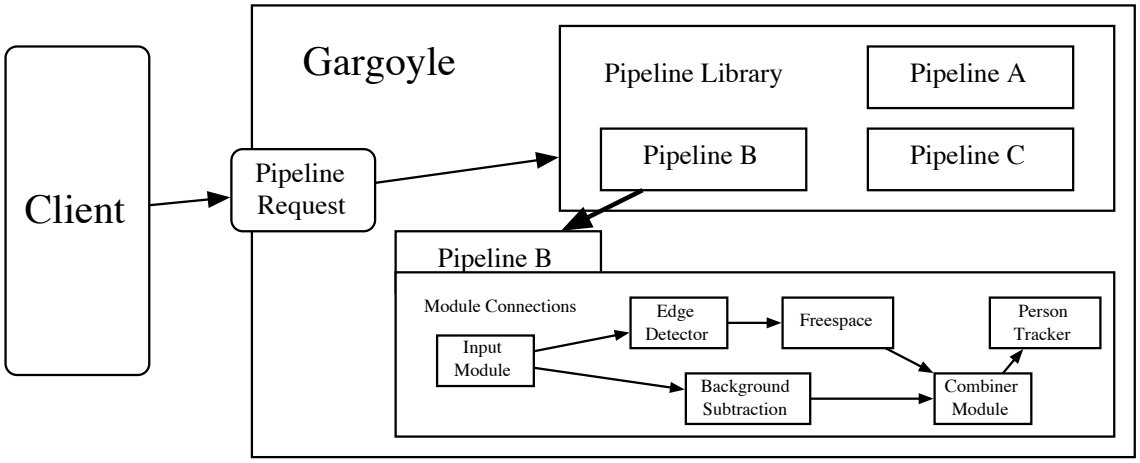

Figure 5.2: A client requesting a specific pipeline from Gargoyle's vision routine library. The pipeline is made up of connected modules.

in the form of a pipeline from the Gargoyle server. Separating these elements into Gargoyle modules allows them to be easily combined to create new vision routines. It also simplifies runtime modification of the routines, as well as encouraging code reuse.

Once the routines have been requested, Gargoyle spawns vision routines as pipelines. It then passes the client a direct communication channel to the pipeline which the client uses to modify the pipelines as required and run them, receiving the resulting data from the pipeline directly. Figure 5.3 outlines this process. We now examine each of these components individually.

5.2.2 Server

The server is a common contact point for all of the clients and pipelines. When a client requests a new pipeline, the server creates a parser thread to interpret the client's requests. This allows the client to communicate directly with the vision routine. The client can then request changes in specific modules in the pipeline directly. The parser interprets the commands and the server constructs each module in its own thread. This multi-threaded

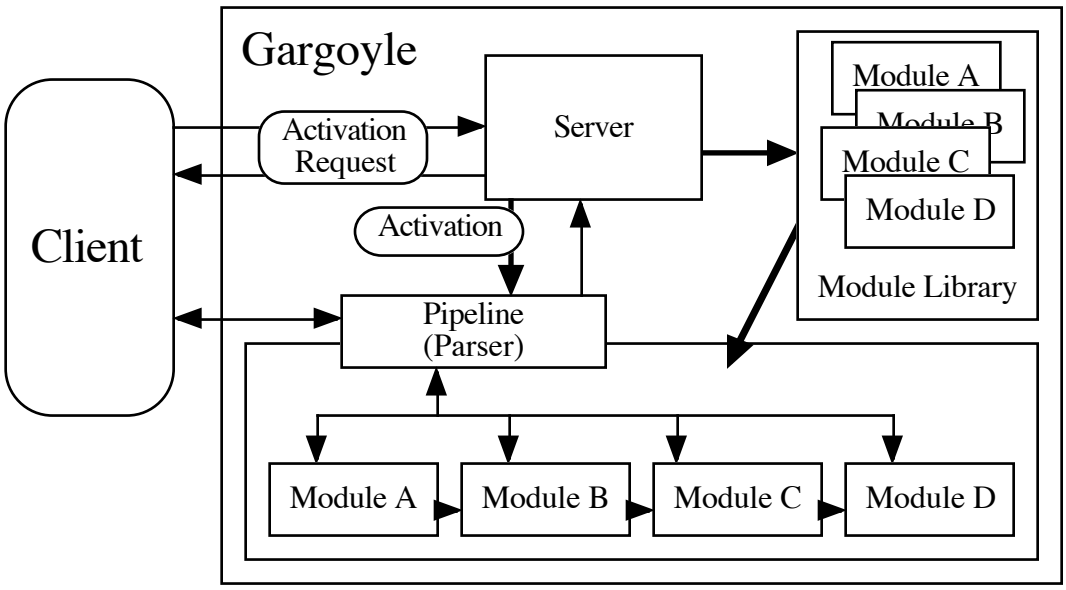

Figure 5.3: The flow of information and control in Gargoyle as it communicates with the client.

mode of operation allows pipelines to interact efficiently through shared memory without the high overhead of inter-process communication.

Setting up the different pipelines as threads also allows Gargoyle to implement efficient tracking. The client can build a tracking routine, and then let it run in a loop. For tracking to work properly, it must be fast and continuous. Since Gargoyle is multi-threaded, the client can allow the tracking pipeline to run continuously while building a pipeline for some other task. By using lightweight processes Gargoyle is able to start and stop vision routines with very little overheard, removing them from memory as necessary. This high degree of control allows very complex tasks which require many vision routines to be supported by a single Gargoyle server.

5.2.3 Pipelines

Pipelines are not actual threads of operation but rather a number of modules working in concert to perform some vision operation as shown in Figures 5.2 and 5.4. They are simply vision routines made up of a series of modules. The notion of pipelines is important, because a single client might have multiple vision routines running at the same time. For instance, when a tracking pipeline has been created, it needs to continue operating in order to not lose the location of the object being tracked. Since the pipelines are independent, the client can construct a new pipeline without stopping the tracking pipeline. This ability is crucial for our notion of general purpose vision. Pipelines allow the required vision tasks to be broken up into special purpose routines which can be turned on and off as needed. One of the advantages of special purpose routines that we rely on is that they can control motor skills directly. By allowing each of the routines to exist in its own space with its own communication channel to the client, one pipeline can be left on all the time while another routine is modified for a new task. The pipeline architecture allows this to happen.

A pipeline is made up of a sequence of modules. One of the pipeline's jobs is to maintain its modules. If the client needs a change in one of the modules, it sends that request to the pipeline which will make the change in the specified module as shown in Figure 5.3. A pipeline is also responsible for constructing and removing modules from itself. It does this by making requests to the server for new modules to be instantiated, and removes unused modules directly. One final task that it is responsible for is handing the results of the vision routine back to the client. Many modules may have results, but the pipeline allows the client to only have to deal with results at the level of vision routines.

5.2.4 Modules

Modules are the basic operating units of the pipelines. They pass information to each other through the pipeline in the way the client has connected them. Each module starts by reading some information, its *input*. Then, using that information, the module does some computation. Once the computation has been finished, the results, or *outputs*, can be set, which can then be used as inputs by another module. This flow of data is summarized in Figure 5.4.

Module to Module Communication

Outputs from modules are connected to the inputs of others to create pipelines. A module will run as soon as all of its inputs are filled. For an input to be filled, the output it comes from must have been set. Different modules may need different numbers of inputs depending on their functions. For instance, an edge detector would need a gray scale image as input while an object identification module might need an edge image, a color image, and a region of interest in which to search for objects. The data passed between modules is usually image data, or image based data, such as regions of interest.

All of the inputs and outputs of the modules are typed to prevent operations being done on the wrong type of data—for example, color operations being done to grayscale images. If the inputs of a module are not all connected then it will not run. Some modules—start modules—have no inputs. All of the start modules in a pipeline will run when the client tells the pipeline to run. Figure 5.4 illustrates the flow of image data through a completed pipeline.

The data is passed from one module to the next through shared memory for both speed and memory concerns. Since a single output can be connected to multiple inputs, this means that care must be taken to not overwrite an input image if the pipeline splits and

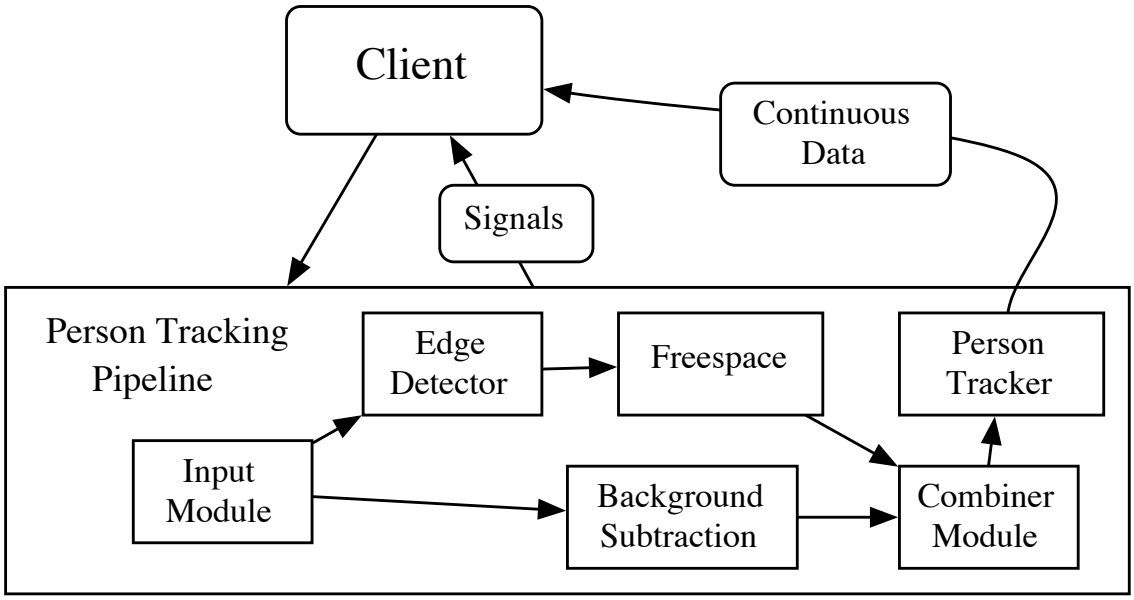

Figure 5.4: A representation of data flowing through a person tracking pipeline from one module's output to the next module's input. Control comes from the client and results are continuously returned.

uses the input image elsewhere. Because of this modules will often produce their output in a new image rather than directly overwrite the input image. However, if speed is the sole concern for a pipeline, all the vision operations can be done in place on the same image as it is passed through the pipeline. Many modules are parameterized to allow the data to be copied only if it is needed for the specific pipeline.

Finally, it is worth noting that some modules have multiple inputs or outputs, depending on what type of information they produce. For example, many modules take a region of interest as input in addition to an image. This way the module can limit the amount of processing it has to do by concentrating on a single region of interest. In terms of output, a color histogram module might create a binary image with the areas of correct color marked and a region of interest around the peaks of the color. Trackers might create the location of objects through regions of interest for a verifier further down the pipeline.

Module to Client Communication

Of course, many modules will also need to relay information back to the client. This is done through the pipeline's communication connection with the client. Since the module designer only cares about the vision routines and not the details of communicating to the client this has been made as transparent as possible—a single function call allows the pipeline to take care of all the communication. In general, module designers should be careful about how much information is sent back to the client. Ideally the module will be parameterized in such a way that only the required information will be sent back. The client may request information through the parameters, including requesting that no results be sent from the entire module.

Results can give the client information about the world for the client to react to, e.g. the results of a tracking pipeline might be used in a tight loop with an effector skill to produce some behavior. They can also give information about the functioning of the pipeline so that the client may reconfigure the pipeline to better accommodate a changed context. These are actually two distinct modes of communication from the client's perspective as shown in Figure 5.4. One is a regular mode of communication, which returns the results of the pipeline each time it is run, and the other is a signal indicating something is wrong. Often the client reasoning system only cares about the signal, leaving the regular communication to the task it is controlling.

In the Intelligent Classroom, the regular messages are passed directly to other skills. In order to facilitate this better, we are looking to expand Gargoyle to have an additional output module type. This module would send regular data through the pipeline to be used as needed. Then the direct communication would only be used to raise signals of abnormal behavior. The reasoning system uses these signals to either switch pipelines, or change how the pipeline is operating.

Client to Module Communication

In order to change how the pipeline is parameterized, the client affects the modules through the pipeline. Each module exposes some parameters for the clients to modify. These parameters are often tied to specific constraints so that the client can reason about what they should be set to. This allows the way in which individual modules behave to be altered at runtime. For example, if too much noise is being returned by an edge detector, the client could raise the threshold. This is how the client passes the context information onto the pipeline.

Parameters have default values and once set, they do not change until they are set again. Thus the client only needs to consider the parameters that directly concern the problem at hand. Any context that the module might receive from the client must be available through its parameters, or the configuration of the pipeline that the client set.

We have found that there are occasionally instances where the client needs to continuously update a parameter. This is very similar to the module sending data back to the client on every iteration. In the future the client will also be able to pass continuous information via input modules. Then parameters can be reserved for changing how the module operates rather than for continuously updated information. We have found that constantly updated information is generally positional information which is very amenable to use as an input.

Module Design

Modules are the key components of Gargoyle—they are where all of the vision happens. When a vision researcher programs a new module, much thought must be put into exactly what will happen inside it. Reusability is an important factor. In order to be reusable, the module should provide an amount of functionality that could be usefully reconfigured in a

pipeline. A module can be thought of as a single action on an image. The larger and more complex the action is, the less the client can do to manipulate the pipeline. For instance, a module that tracks and identifies a specific object would not allow the client to define how the object is being tracked, or when to identify it. Instead, vision routines should be broken up into smaller units which can be used in different ways, and even in different pipelines. In general, the fewer complex actions a single module makes the better, because these smaller modules are more likely to be reused, or swapped out of a pipeline for some other module which is better suited to the current context.

5.2.5 Libraries

Modules are extremely useful in terms of breaking down vision routines, and quickly developing new ones. They do have some disadvantages however, in particular the fact that they are limited in their interaction. Since the data flows in a single direction, upstream communication is difficult. Loops can be made for tracking purposes, but sometimes more complete control of a module is needed. Because of this we have provided a number of modules as libraries as well. The modules are then just simple implementations of these libraries, but if a more complex vision routine requires complete control of that processing unit, the library version can be incorporated directly into the module.

5.3 Gargoyle in use

Since Gargoyle is not particularly useful without a client, it is worth taking a brief look at the clients we have connected to Gargoyle.

In order to make designing vision routines easier we built a simple graphical user interface client which the designer can use to construct and try out vision routines on the fly without needing to implement a reasoning system. A picture of this GUI in action is

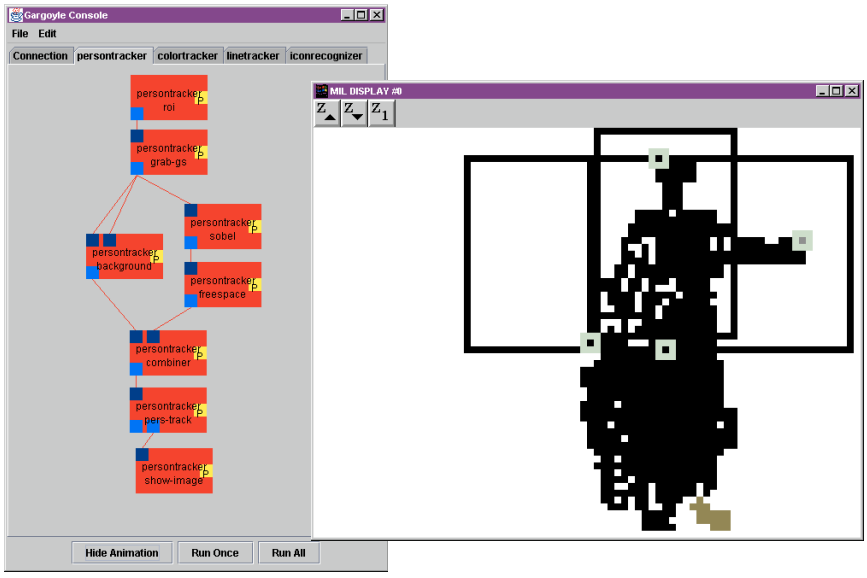

Figure 5.5: A graphical user interface gargoyle client which allows a user to easily build and configure pipelines.

shown in Figure 5.5 with a pipeline running, and some debugging output displayed. This helps the designer to test the routine and figure out what the constraints that the routine relies on are. It allows the designer to do any manipulation to Gargoyle that a client system could—build new pipelines, configure them, run them, and view the results of the pipelines running. We have found this to be an invaluable tool in designing vision routines.

Once the routines were developed, we used Gargoyle as the vision input for a number of different clients. Each of these clients are described in detail elsewhere in this dissertation; this is just to give an overview of where Gargoyle was used as an argument for its effectiveness at fulfilling both its design goals as well as the needs of a more general purpose vision system.

5.3.1 CHIP

While Gargoyle was developed for CHIP, it was never fully integrated with the robot. Initial experiments were done on CHIP by implementing Perseus as a Gargoyle pipeline. This proved to be successful, and the functionality of Perseus was implemented in Gargoyle [38], however as seen in Section 4.4.3, the integration between RAPs and Perseus was not tight. Thus, none of the constraints on the Perseus routine were ever implemented in RAPs, and therefore none of Gargoyle's more interesting capabilities were used.

5.3.2 Intelligent Classroom

The Intelligent Classroom was the first complete implementation of a vision system using Gargoyle. Chapter 6 discusses how the Classroom uses Gargoyle as well as the pipelines that were developed for the Classroom's tasks in detail. The Classroom uses pipelines of skills in its reasoning system, and meshes quite nicely with the Gargoyle way of doing things; it simply treats modules as skills. The Classroom connects them into Gargoyle pipelines which in turn are part of Classroom skill pipelines. To do this the Classroom passes all of the results of the vision routine directly on to the next skill in the Classroom pipeline. Finally, Classroom skills signal abnormal behavior similarly to how Gargoyle modules do, allowing the Classroom to use its reasoner for dealing with skills to deal with Gargoyle pipelines and modules.

5.3.3 Interactive Image Mosaic

One more use to which Gargoyle was put showed that it could easily be reused in other projects. A group of students built an Interactive Image Mosaic. In order to operate it needed to be able to track people, in addition to actually generating the Image Mosaic. The client they designed was a simple state machine, and used extremely little context.

However, they were able to quickly generate results by taking Gargoyle and using modules from pipelines built for the Classroom, adapting them for their new context. The Image Mosaic and the pipelines that it used are described in Chapter 7.

Gargoyle has been used by a number of clients to provide context-based, real-time information for specific goal-oriented tasks. The project that most completely implemented our methodology for computer vision using Gargoyle was the Intelligent Classroom. The next chapter looks at the Intelligent Classroom.

Chapter 6

The Intelligent Classroom

With the advent of Gargoyle and more robust vision routines, our research group began looking at ways to improve human computer interaction through computer vision and plan recognition [39, 40]. We began this line of research on Chip with the robotic waiter task discussed in section 4.4.4. We obtained encouraging results using the robot; however, the interaction required a large amount of cooperation on the part of the user. By forcing the user to make up for the robot's failings, the interaction became quite unnatural. It became clear that more complex modes of interaction were beyond the capabilities of our robot. This led us to look for new interaction tasks, and then to the Intelligent Classroom.

The Intelligent Classroom overcomes many of the disadvantages of the robotic environment while maintaining the key elements that this research requires. The Intelligent Classroom senses the world with cameras and microphones, similarly to how a mobile robot would, but does not need to worry about bumping into walls, since it is the walls. Being able to sense allows the Classroom to interact with the user in novel ways. Rather than having the user deal with operating VCRs and projectors, the Classroom can take care of those things for him. The very nature of being a classroom provides the Classroom with a

number of interesting ways to interact with the physical world and the user.

This chapter looks at the design of the Intelligent Classroom in terms of its entire research project, and then focuses on the specifics of its computer vision implementation. The Classroom is critical to our study of computer vision because it was the first complete implementation of Gargoyle with an autonomous controlling client, demonstrating the viability of the techniques for computer vision that we put forward. In addition to demonstrating our computer vision techniques, we have used the Classroom to demonstrate a number of interesting modes of human computer interaction, with the end goal of having the computer cooperate with the user in complex and interesting ways.

This chapter begins by describing the Classroom and its capabilities. Section 6.1 also examines the goals of the Classroom project. Section 6.2 takes a brief look at the hardware requirements for the Classroom. Section 6.3 looks at the planning and execution system that runs the Classroom, specifically the interaction between the high level reasoning and the vision framework—Gargoyle. Finally, Section 6.4 describes the tasks the classroom performs and Section 6.5 describes the different vision routines implemented for those tasks.

6.1 Overview

The Intelligent Classroom is a lecture hall environment with audio-visual enhancements for a professor instructing a class or an invited speaker delivering a lecture. Oftentimes when a lecturer is faced with an array of audio-visual tools he can be overwhelmed by the complexity involved in figuring out how to change modalities for different parts of the lecture. If an audio-visual technician is available, then the set up and operation of the equipment can be taken out of the hands to the lecturer. The Intelligent Classroom is like that technician in that it takes the responsibility for operating the equipment in the room out of the hands of the lecturer and operates it itself.

When developing the Classroom, the key feature that we wanted it to have was the ability to assist a speaker delivering a lecture in the room. More than simply being a new set of buttons that the lecturer must learn how to use, the goal of the Classroom is to allow the speaker to go about his lecture as if there were a competent audio-visual assistant in the room controlling all of the auxiliary equipment for him. In order to assist the speaker the Classroom needs to be able to obtain certain information from the world. Specifically, the Intelligent Classroom has cameras to watch what the speaker is doing and a radio microphone set to listen to what he is saying. In this way it has sensors that obtain information from the real world, much like a robot. An assistant is not particularly helpful, however, if all it can do is watch. In addition to the ability to watch and listen to the lecture the Classroom controls its embedded audio-visual features, allowing it to play videos, change slides, turn different elements of the room on or off, and use a camera to intelligently film the lecture.

The key difference between the Intelligent Classroom and a normal audio-visual lecture hall is not in these things, however, but in the ability to help the lecturer without him having to ask for it. In addition to merely interacting with the world, the Classroom environment was designed to cooperate with the user. A competent audio-visual technician does not need to be told when to move the camera to point to a different part of the room, and can often follow along with a slide presentation without necessarily having to be told when to switch slides. In order to be able to exhibit this behavior the Classroom needs to know something about the task in which the user is engaged. Additionally, it must be able to tell where the user is in that task. In order to zoom in on the board where the lecturer has written so that the viewers of the film can read what is there, the Classroom needs to know that if the lecturer walks to the board and picks up some chalk he is probably about to write on the board.

In many ways the Intelligent Classroom in immersed in the real world, and

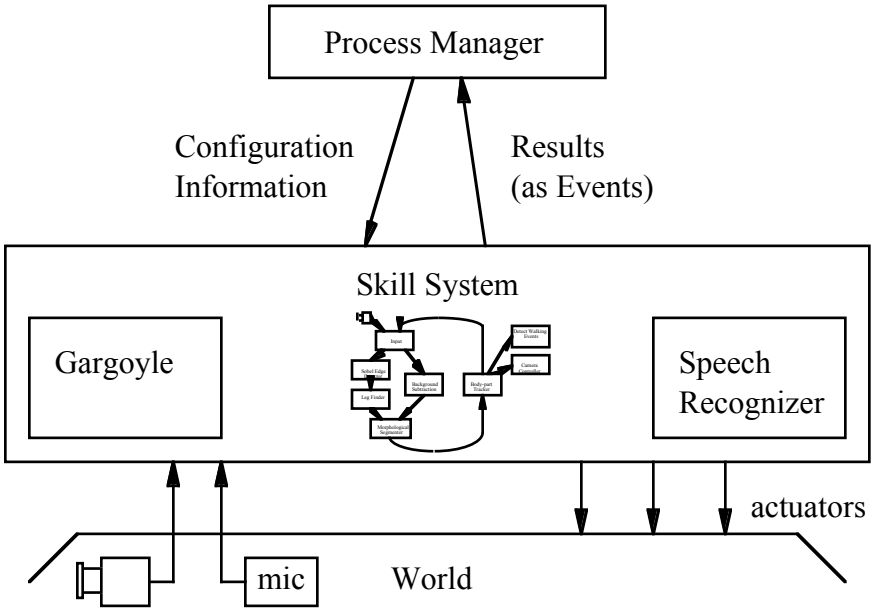

Figure 6.1: The system architecture diagram for the Classroom.

therefore has to deal with many of the same problems that a mobile robot would have. Since we had developed the Animate Agent Architecture, described in Section 4.3, to manage many of these problems, it was a clear choice for use in the Classroom. The high level system was redesigned to better support the plan recognition required by the Classroom to cooperate with the lecturer, and the low level vision systems were replaced with Gargoyle. The clean separation between high level reasoning and low level action and sensing skills remains, however, and provides the Classroom the ability to deal with the inherently noisy physical world.

The high level system is called the process manager and is equivalent to the RAP system in CHIP. This controls all of the planning and plan recognition. In addition, it configures all of the processes in the Classroom, and configures skills with Gargoyle pipelines to get data from and affect the world as shown in Figure 6.1. The reasoning system and slide changing system, Jabberwocky, are David Franklin's work and described in [41].

Figure 6.2: The Intelligent Classroom hardware stack.

6.2 Hardware

In order to accomplish all of these tasks the Classroom needs to have physical methods of interacting with the world. This section looks at the physical design of the Classroom. Of particular interest is the fact that these complex behaviors are accomplished with relatively inexpensive hardware. The lack of space and power constraints in the Classroom, as opposed to a robot, allowed us to use off-the-shelf hardware.

The Classroom has two main modes of sensory input, voice and vision. The Classroom currently uses three cameras to view the room. A cheap fixed black and white camera is used for some tracking tasks. In addition, we use two Sony D-25 pan tilt zoom color cameras for color video input. These cameras are used for making a film of the lecture and some active vision tasks. For voice input we use a radio microphone which the lecturer must wear, which connects to a standard PC running IBM's Via Voice, a speech recognition application.

Beyond just sensing the world, the Classroom interacts with the world through number of different devices. We have two VCR's that the Classroom controls, the filming

camera mentioned above, a projector, and a video crosspoint switch. The Classroom also has the ability to be easily extended to control any additional elements for which have appropriate control mechanisms.

Coordinating all of these devices is done through ethernet and serial connections. The computer vision and voice recognition skills each have a dedicated PC accepting their respective input and processing it for the process manager. These machines coordinate with the process manager through a local network. The process manager runs on a Macintosh along with the skill manager. The skill system can then control vision and speech recognition routines through the network and all the other devices via serial connections. Most of the serial connections are coordinated through a single AMX communications hub which parses commands for the respective devices and sends the appropriate commands. This provides good abstraction for adding more devices for the classroom to control. The AMX device contains a number of additional control outputs which the Classroom can utilize including infrared, serial, AMX control, and a number of analog controls as well. The physical Classroom layout is shown in Figure 6.3. The heart of this control mechanism is the process and skill managers.

6.3 Process and Skill Managers

The Classroom's reasoning system is itself an experiment. One of the goals of the Intelligent Classroom project is to develop a reasoning system that can interact with people in a natural manner. In order do this, the Classroom needs to be able to know what the user is doing. This means that in addition to planning for itself and executing those plans, the reasoning system must also be able to recognize the plans of the user. The systems used to accomplish these goals were a process manager and a skill manager developed by David Franklin.

Since we are dealing with the physical world we felt that using many of the

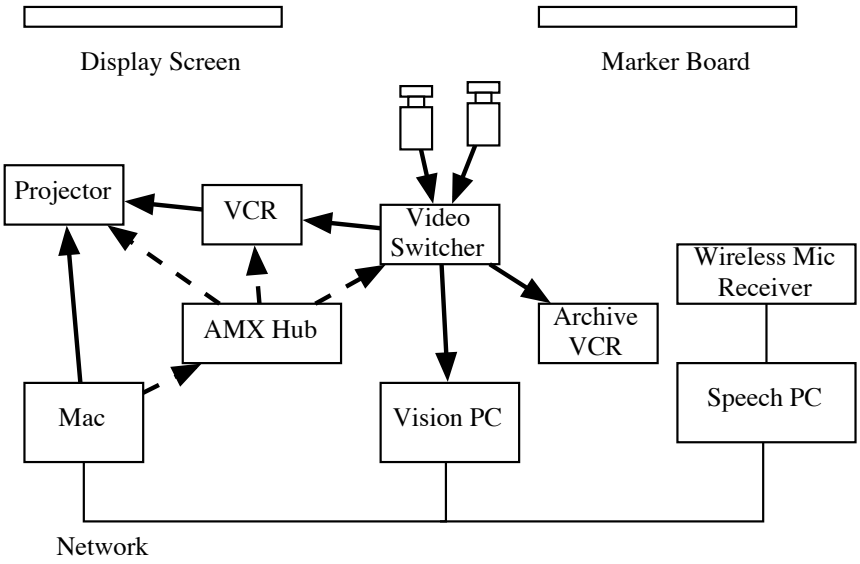

Figure 6.3: The connections between the different Classroom physical components.

techniques developed for the Animate Agent Architecture would prove helpful. Specifically, separating the low level systems which interact directly with the continuous world from the reasoning systems which operate in a much more discrete world. This separation can be seen in the system diagram in Figure 6.1.

In order to maintain this layer of abstraction, the Classroom's reasoning is divided into two separate segments. First the process manager, which is similar to the RAP system in Animate Agent Architecture, deals with planning, process recognition, and process monitoring. The process manager controls the skill manager, which handles the details of interacting with the physical world, and setting up all of the required communications channels. A complete description of these pieces of the Classroom can be found in [41]. Some understanding of these systems is required, however, because of the nature of the interaction between high and low level systems for the computer vision being discussed here.

6.3.1 The Process Manager

The discussion in Chapters 2 and 3 points out that the control mechanism for a computer vision system built using our methodology must be able to reason about the different constraints and be task based. In the Intelligent Classroom that control mechanism is the process manager.

In many ways the process manager is like the RAP system in the Animate Agent Architecture. It reasons about the goals that it is given and decides on the sequence of actions that it needs to do in order to accomplish those goals. This sequence of actions is called a process in the Classroom. By this definition a process is just a plan, but the distinction between them is important.

When designing the process manager for the Classroom, it was clear that it would need to have a plan recognition component in order to cooperate with the lecturer. Unfortunately the tasks that a lecturer might do in the Classroom proved to be similar, given few observed steps from which to reason, and standard plan recognition was insufficient. A process is different from a plan in that it is merely a set of steps, without an inferred goal. By using processes instead of plans the Classroom is able to simply observe what the lecturer is doing at any given moment, rather than constantly needing to update a complete notion of what goal state the lecturer has.

The process manager maintains processes for itself, i.e. tasks that the Classroom is engaged in, and processes for the lecturer, i.e. tasks that it believes the lecturer is engaged in. In order to have the ability to update its knowledge of external processes, as well as the progress of its own processes, the process manager is able to monitor all of the processes that it maintains. The Classroom can update what it believes the lecturer is doing by way of explanation. That is, the Classroom looks at the actions that the lecturer is making and picks a set of possible explanation processes from those actions. As time goes on and the

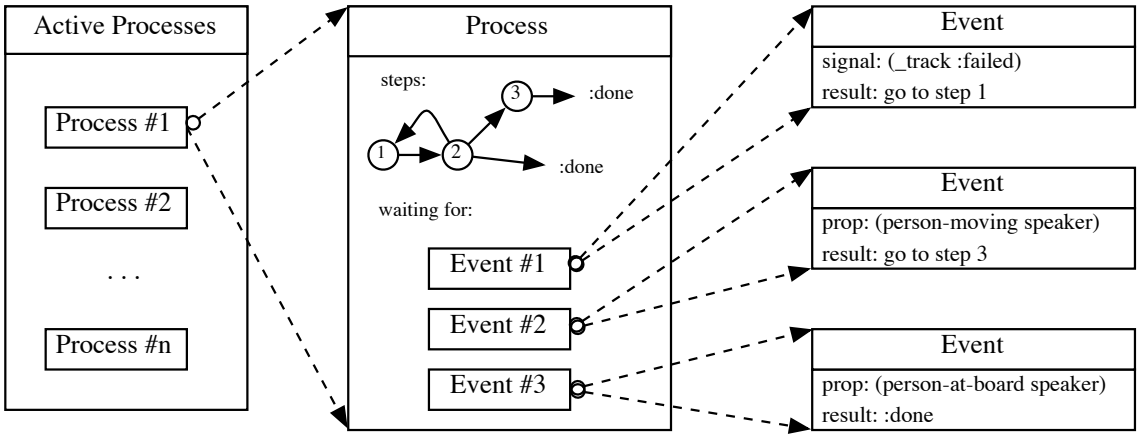

Figure 6.4: The process manager with active processes waiting for specific events.

lecturer does steps not in some of the explanations, the Classroom can eliminate incorrect explanations. Another way the Classroom eliminates processes is because they have not progressed properly.

A process contains a bit more information than just the steps that need to be accomplished. It also contains information on how that process is supposed to progress. This information includes things like ordering of steps, or lack thereof, and constraints on the tasks. One example of constraints that can be encoded into a process are time constraints, stating that a task should only take a specific amount of time, or should take at least a certain amount of time. Other constraints could include things like the location of the actors in the process, or some state.

Since the process manager uses the same representation for observing the lecturer, or more generally other agents, and for planning its own execution, it can also monitor how its own processes are progressing. For its own processes it executes the actions, for observed processes it observes the actions, but in both cases the Classroom monitors the progress of the processes.

This ability to monitor the process is critical for implementing our vision methodology.

The process manager monitors the progress of a vision routine that it instantiated via the skill manager, and makes sure that its constraints continue to hold. The process manager monitors the progress of a process two ways. First, it can receive a signal from subprocesses. In the case of a vision routine, the vision routine can signal the process manager with important information about the state of the world. Second, the process manager keeps important information in memory. Changes in memory can also signal constraint changes in the world from other sources which impact the running of vision routines. If the constraints that a vision routine relies on fail, the process manager makes appropriate changes to the vision routine.

It is important to note that while the process manager reasons about the sensing, and the constraints that it relies on, it does not in fact deal with the physical world directly. The process manager only deals with a clean abstraction of the real world. This means that there is an inherent difference between internal processes and the low level skills that interact with the world. We learned from the Animate Agent Architecture that this is a critical distinction. In order to keep this distinction clean, the process manager does not activate skills directly, but reasons about them and then communicates with a skill manager to take care of the skills.

6.3.2 The Skill Manager

In order to interact with the world in real-time, it is often necessary to set up tight sensing-acting control loops. These loops need to sense the world and then act on that data in order to accomplish some task. The skill system in the Classroom is very similar to that of the Animate Agent Architecture described in Chapter 4. This section looks at some differences in both the skill manager, and the way the skills it manages are used.

One major difference between the Animate Agent's skill system and the Classroom's

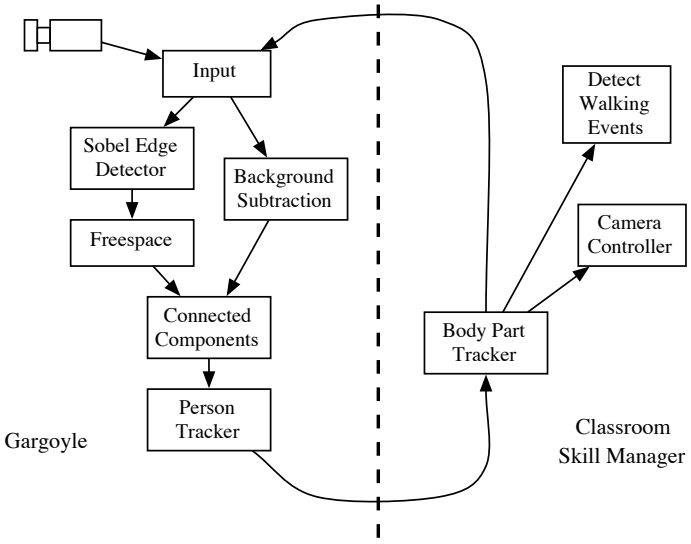

Figure 6.5: A complete skill pipeline spanning the Classroom's skill manager and Gargoyle.

is that the skills do not address global memory channels. Since the Classroom needs to maintain multiple external processes to reason about the speaker, this would be unmanageable. Instead, the Classroom's skills contain inputs and outputs which allow the skills to be hooked up in ordered sets to complete some computation. These skills can be parameterized, just like Gargoyle modules. In fact Gargoyle modules are treated exactly the same way as skills in the skill manager. This allows the process manager to reason about the vision at the module level if needed.

Much like Gargoyle modules, a set of Classroom skills linked together by inputs and outputs is called a pipeline. In fact, Gargoyle pipelines are often a part of Classroom pipelines. These skill pipelines can become large and heavily parameterized. In the Classroom, these pipelines are often reused with the same parameters, and sometimes by more than one process.

The other major difference between skills in the Animate Agent Architecture and the Classroom is that it provides a transparent communications mechanism between

different skills. On Chip the vision routines and the skill system were two completely separate entities. This made communications between skills simple since the off-board vision routines were not skills and all the skills resided on the same machine. In the Classroom, vision routines are treated as skills, but Gargoyle, where the vision routines run, is on a completely separate machine from the rest of the skills. To overcome this problem, the skill manager maintains communications channels between the skills no matter which machine they are on. It also decides which machine a skill should be run on when the process manager requests a new skill to start. Figure 6.5 shows a complete pipeline with some skills running as Gargoyle modules. The results from the Gargoyle modules are transmitted to the Classroom skills which in turn update the Classroom memory and take any required actions.

As one might expect, many of the skill pipelines that the Classroom uses involve a vision component for data input. Next we look at the tasks the Classroom does. In particular we look at the constraints the different tasks provide for the process manager to reason about. The rest of this chapter examines in detail the operation of the different vision routines that the Classroom uses.

6.4 Tasks

When a speaker is giving a talk in the Classroom, the tasks that the Classroom will do depend on which audio-visual elements the speaker chooses to use. The speaker can also easily opt out of using the audio-visual elements of the Classroom. The Classroom will not interfere with this and simply bottom out in its root skill which films the lecture. On the other hand if the speaker wishes to utilize some of the other features of the Classroom it will hierarchically engage more skills. The key is that any of these tasks can be utilized or not, as the speaker wishes. The Classroom exists to assist the speaker transparently, not to

Film the Speaker	
Entire room framing	none
Speaker moving framing	Centroid of the speaker Rapid updating
Speaker standing framing	Location speaker is standing Moving state of the speaker
Head shot	Location speaker is standing Moving state of the speaker Wether the speaker is gesticulating Rapid updating Possibly the location of a podium
Speaker writing framing	Location of the speaker (from centroid and height) Location of speaker's hands Location of the board
Video and Slide feed	How long to show the alternate feed

Figure 6.6: Summary of filming tasks, showing the information required to accomplish the given task.

get in the way of the speaker. With this in mind, we now examine some of the tasks which we have implemented in the Classroom.

6.4.1 Film The Speaker

One of the Classroom's primary tasks is to film the presentation that is being given. This film can then be archived for later use, sent off-site for remote presentations, or transmitted in real-time for distance learning. Producing a reasonable video of a lecture involves more than simply sticking a camera in the back of the room and hitting record. Unfortunately, this is what often happens due to the cost of employing a camera operator for every presentation that is recorded. The Classroom provides a superior recording of the presentation, compared to a static camera, with no work on the part of the speaker. The Classroom films the presentation using a pan-tilt-zoom camera unit as well as a video switch so it can record alternate video feeds as needed. The Classroom has a number of options of what to put on

Figure 6.7: A number of alternate framings for filming the speaker, clockwise from top left: generic, walking, writing, and gesturing.

the recording of the lecture.

With all of these methods available to the Classroom, it is possible that it will go overboard and produce a less useful video. We prevent this from happening by having a hierarchy of filming techniques, where the default is simply filming the entire room with a steady camera. This hierarchy is shown in Figure 6.6. In this way if all of the Classrooms sensors fail, and it is completely unable to understand what the speaker is doing, a reasonable, though poor quality, video is still produced. This is as opposed to trying to frame the scene in some inappropriate manner using noisy data. Examples of the different framing techniques that the Classroom uses are shown in Figure 6.7.

The next more specific framing technique follows the speaker with a slightly tighter shot. This technique involves simply placing the center of the speaker in the center of the frame, and following him as he moves. Since the Classroom knows where the speaker is and follows him with the camera, it can provide a tighter shot which allows the viewer to

see the speaker in more detail. In order to use this framing technique, the Classroom only needs to know where the speaker is. Therefore any of the person tracking techniques from Section 6.5 will provide the appropriate information. In fact, the amount of information required by this task is very small, it only requires the two dimensional location of the centroid of the speaker in image coordinates. In addition, there is a small time constraint on the results. If the speaker is moving quickly and the tracking algorithm is not fast enough, the Classroom may not be able to keep the speaker in the scene. If this happens, the Classroom can fall out to the default framing until the speaker slows down. Alternatively it can switch to a tracking technique with a higher frame rate.

This filming technique has proven to be quite robust, and produces a video of higher quality than a static camera. However, since the Classroom knows more about the its environment, it can provide a number of more specific framing techniques.

If the general technique follows the speaker as he moves about the space, an obvious additional filming technique is to change the framing when the speaker stops moving. When someone is talking while not moving, subtle body language is often important, be they facial expressions or arm gestures. The Classroom provides these better on the recording by framing the speaker much closer. Since the speaker has stopped moving there is a much smaller chance that he will suddenly move out of the frame. The Classroom relies on this to provide a tighter shot. This technique again relies on knowing where the speaker is in the image frame, so simple tracking techniques will work for this framing technique as well. Since this shot is too tight for following a moving person, if the speaker starts moving, the Classroom will revert to the less specific following shot framing.

Finally, the most specific speaker shot is a tight head shot. These are often used in television news casts when someone is delivering a report for a steady camera, and is very still. This behavior is sometimes seen in a lecture hall setting if the speaker is using

a podium. If the Intelligent Classroom has a podium, it can use this more precise framing when the speaker goes to the podium and stands still while lecturing. Determining when to use a head shot is very similar to determining when to use the previous framing technique. There are a few complications from a visual standpoint however. In order to do a head shot, the Classroom requires not merely the location of the speaker, but also the location of his head. Additionally, the previous framing technique includes the torso of the speaker which records the speakers gestures as well as his face. This is potentially important to have in a video of a talk, especially with more animated speakers. Therefore this technique should only be used if the speaker is not gesticulating. This means the Classroom requires the location of the hands of the speaker as well and provides another task constraint on the vision routine. With this additional information the Classroom reverts to a wider shot when the speaker starts moving his hands regularly, which the Classroom assumes should be in the video. These additional task constraints require that the Classroom change to the background subtraction tracking routine if it is not already using it. If the other constraints for that routine do not hold, then this framing technique is not available to the Classroom.

These framing techniques are used by the Classroom to produce a video which captures a much higher quality video than simply placing a camera in the back of the room. There is more to producing a quality video of a lecture, though, than simply focusing on the speaker at all times.

6.4.2 More Filming Possibilities

From a vision standpoint, filming the speaker is exciting because it utilizes many different vision routines. Of course, there can be times in a presentation when the focus of attention is not on the speaker. For example, if the speaker is using a video in his presentation, the Classroom will put that video feed directly into the recording of the presentation. In this

way the person viewing the presentation on tape will be able to see what the video is.

Similarly, since the Classroom knows when slides are being changed in a slide show, it can show the slide in the video feed, depending on how much text the particular slide has. The Classroom could also use tracking information about the speaker to know when watching the speaker is important. For example, if the speaker is particularly animated, it might want to hold off on showing the slide, or switch back to showing the speaker from showing the slide or video. Again, this is a hierarchical process where the basic action that the Classroom provides, showing videos and briefly flashing slides, produces a quite usable recording of the presentation. So far, all of the filming techniques shown are completely automated and require no extra work on the part of the speaker.

The Classroom also zooms in on things that the speaker has written on the board. If the Classroom knows where the board is, it can then zoom in on portions of the board that the speaker has written on so that it can be read in the video. This task requires much more information from the vision system than the previous tasks. In addition to knowing where the speaker is in two dimensions, it must also know if he is near the board, and then where he is writing on the board. The framing technique used makes sure that both the speaker and the board are in the scene while the speaker is writing. While this is going on, the Classroom tracks the speakers hand to see where he is writing. The filming camera is then directed to zoom in on the board where the lecturer is writing.

The more the Classroom knows about the lecture the better it performs, so the user can provide a script of the presentation to improve his results. If the Classroom knows that the speaker is going to be working at the board for a while, and then showing some slides, the recognition of those events becomes much easier.

Beyond simple scripting, additional knowledge could be used to generate new framing techniques. For example, if the speaker is presenting on anatomy, you could imagine

Operate Slides and Videos	
Use audio commands	Speaker must be using microphone headset.
Use Jabberwocky	Speaker must be using microphone headset Contents of slides must be known
Use virtual buttons	Location of the speaker (from centroid and height) Location of speaker's hands Location of the board

Figure 6.8: Summary of audio video controlling tasks, showing the task constraints.

him frequently referring to a skeleton. If this is the case, he may very well want the Classroom to zoom in on the specific anatomical element he is pointing at. All of the required vision and framing techniques already exist for this skill to be implemented. By simply giving the Classroom a small additional piece of knowledge it can accomplish wildly different tasks.

6.4.3 Operate Videos and Slides

In addition to simply filming the lecture, the Classroom controls all of its presentation components. Most of these are controlled through verbal commands. For example, the speaker can tell the classroom to play a video, or start a slide show and it will do so, starting VCRs and projectors as needed. This will also change the active filming technique. These tasks are are outlines in Figure 6.8. The important feature of these Classroom abilities from a vision perspective is that the Classroom knows about changes in the environment before they happen. It knows that part of the background is going to change, and that the lighting may change, and can take appropriate action for the vision routines.

In addition to controlling the Classroom through verbal commands we provide a number of nontraditional control techniques. For switching slides during a slide show the speaker may use the obvious "next slide" and "previous slide" commands, but he can also have the Classroom follow along with the talk and change slides as needed. The Classroom

reads the slides in the presentation and represents each of the slides based on the words it contains. It can then pick the slide that is most like what the speaker is currently talking about. By assuming that the speaker will go linearly through the talk, it can greatly improve its performance. But the speaker can also skip around in the talk by telling the Classroom to "skip to the slide that talks about...". We find this behavior interesting because it is something that a human operator would not be able to easily do. This was implemented by David Franklin and described more fully in [42].

Another alternate control mechanism the Classroom has is "virtual buttons". Since the Classroom has the ability to track what the speaker is pointing at and whether or not he is close to the wall, it can watch the user touch part of the wall that has been designated a control button. In this way the speaker can use non verbal commands to control the presentation. The first task constraint for this behavior is knowing where the user is pointing, which is very similar to filming him write on the board. Additionally the Classroom must know which section of the wall is a "button". More accurately, it must agree with the user on which part of the wall is which button. The Classroom has two techniques for sharing this information.

First, it can inform the user by projecting a button onto the video screen using its projector. We have actually used this functionality to create an entire menu system for turning on different Classroom behaviors as shown in Figure 6.9. This technique requires that the projector be on, and that the Classroom have the ability to draw onto the screen. This is not possible if there is a video playing through the projector, so we came up with an alternate way of sharing the information of where it button is.

Instead of having the Classroom always dictate where the button is, the user can actually tell the Classroom where the button is by drawing it on the board. In order for this to work, the Classroom needs two pieces of information. First, it needs to know where

Figure 6.9: Classroom generated and user generated virtual buttons in use.

the button is being drawn. It can gain this information with a tracking routine that follows the speaker's hands, just like when framing the recording for when the speaker writes on the board. Then once the icon has been drawn, and the speaker steps away from the board, it needs to read the icon off the board. There are a number of icon recognition routines that the Classroom has at its disposal to read the icon. Matching is a computationally expensive task, and this is only done when other tasks are not needed, normally in an initialization phase.

Finally, once the user and the Classroom have agreed on the location of the buttons, the Classroom needs to know whether or not the speaker is touching the wall at one of those locations. In order to do this, the Classroom must know the location of the user, whether or not he is at the wall, and where his hands are. This are the same task constraints as for finding the spot where the user is writing an icon on the board, and therefore uses the same vision pipeline.

6.4.4 Handle Initialization

As we saw in the "virtual button" task, a number of the Classroom's tasks require knowing specifics about the space in which it exists. In a set environment these locations can be assumed, and if the boards and projectors are bolted down, they will not change. In order to make the Classroom a more general system, we have also provided it with some tasks which allow it to initialize itself. Some of these tasks are very computationally expensive. In order to allow the Classroom to accomplish them without impeding other tasks which have time constraints, the Classroom uses an initialization phase. Depending on the information needed by the Classroom, it requests that the user point to them. For example, if it does not know where the white board is, it asks the user to point to the four corners of the white board. Similarly, anything else that the Classroom does know have a given location for, it can request that the user point it out.

This start up time is also the ideal time for the Classroom to acquire any additional information that it may need for some of its vision routines or other tasks. In the virtual button example given in the last section, we saw that the Classroom needs to know where the buttons are. The matching pipeline is very expensive, and can interfere with the filming task if run at the same time. So the user should draw the icon before the start of the lecture if he does not want it to interfere with the recording. Of course if he does draw it during the lecture, the Classroom can realize that the time task constraint will not hold when it is reading the icon, and use the most general filming technique. This would remove the time constraint and allow it to either stop the tracking pipeline altogether, or simply run both pipelines. There are many other pieces of information that the Classroom can obtain for the vision routines: color histograms of the user, height information on the user, or any other required information. In our Classroom most of the information that it needs is stored to simplify the startup process, and is stable because many of the room's

elements do not move.

We now look in detail at the vision routines we developed to accomplish these tasks. They were all implemented as Gargoyle pipelines which the Classroom uses to accomplish these tasks.

6.5 Vision Routines

The Classroom's vision routines are the first complete implementation of a vision system using Gargoyle. Each of the vision routines is a Gargoyle pipeline, designed for specific tasks in the Classroom. The Gargoyle architecture provides a framework in which we are able to rapidly develop those pipelines and expand our library as needed. We also found that many of the pipelines can be easily reused for a number of the Classroom's tasks. Looking at the list of tasks given in Section 6.4 we can see that many of the Classroom's tasks share task constraints. Since we thought about these ahead of time, we found our ability to reuse the pipelines was much improved over our experience with the robot.

We now examine each of the vision routines in detail, specifically noting the constraints that the routine relies on, the operation of the routine, and some examples of how the Classroom makes decisions about how to use the routine. The detail of how the different modules run are not particularly novel, however they are necessary to understand how the Classroom is able to turn this collection of routines into a useful vision system. Specifically, by understanding how the modules work, we can see how the Classroom is sometimes able to obtain the required information about the current context from the modules themselves.

Tracking the speaker in the Classroom is such a fundamental vision task for the Classroom that we developed two different routines for tracking people: background subtraction based and color based. The color based tracker is more efficient than the

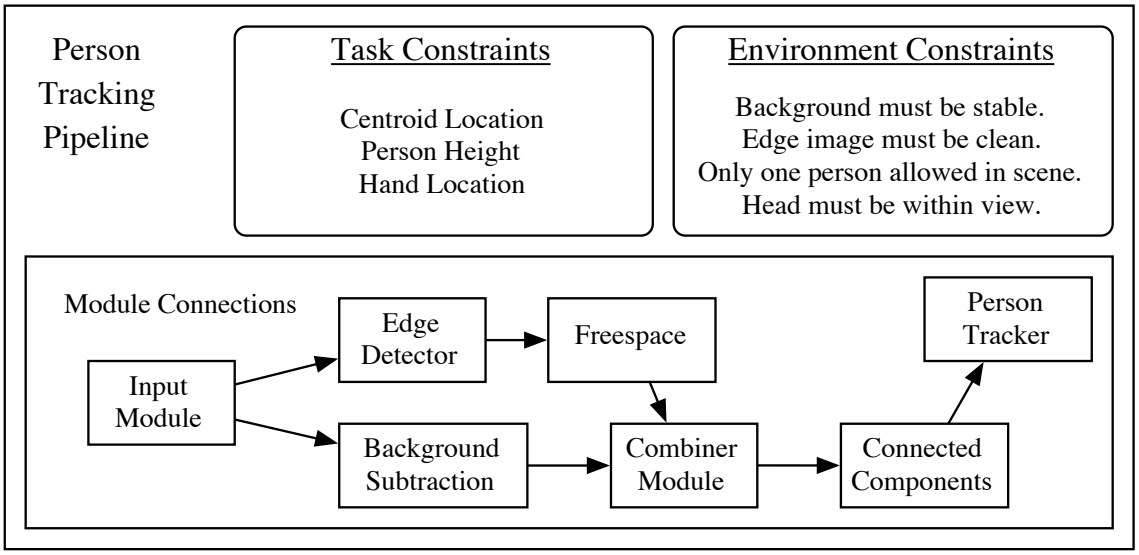

Figure 6.10: The background subtraction tracking pipeline and constraints.

background subtraction based tracker, and relies on fewer constraints. The background subtraction technique, on the other hand, provides more information about the location of the person being tracked.

6.5.1 Background Subtraction Tracking

The background subtraction person tracking pipeline is built up of a number of modules in Gargoyle. First it grabs grayscale images from the camera using a standard input module. This in turn hands the image off to the background subtraction module. This determines where the motion is in the scene, which is assumed to be the person being tracked, and is handed off to the person tracker module. The person tracking module does some morphological analysis of the person region and some rudimentary tracking to determine the most likely location for the features being tracked which it reports back to the client via Gargoyle. We now examine the operation of each of the modules.

Input Module

The input module takes images from the camera and converts them into a standard Gargoyle image on which the other modules can operate. This module is very straightforward, but has some complexities that are worth discussion since it is used by all of the pipelines. They all come from the fact that Gargoyle is meant to be portable, but grabbing frames for a video input source is inherently platform specific. Gargoyle gets around this problem by abstracting the frame grabbing away from the input module. This means that the same pipelines can be used with different equipment setups, although there is a danger of losing capabilities of the native frame grabbers with this abstraction. To get around this, the input module is configurable in a number of ways, which the frame grabber may or may or may not implement in hardware.

The first way in which the input module gives information to the frame grabber is by actually providing the client different input modules to pick from. This allows different input modules to be used to produce different types of images (e.g. color or grayscale). Some frame grabbers produce grayscale directly into memory while others need to have the data converted. By picking the needed input module, the frame grabber will utilize the appropriate hardware advantages where possible. Beyond simply choosing different modules, the modules themselves are also parameterized.

The input modules are parameterized on scale, size, and location. These parameters can sometimes be implemented in hardware as well, and when possible they are. Sometimes pipelines need information from only a subsection of the scene. Parameterizing the input module with that information will save internal memory, and potentially give some speed up to the digitization process. Additionally, different pipelines may require a specifically sized image in order to operate properly. If this is the case, then the input module can be parameterized to produce images of that size. In the other hand, if the other modules in

the pipeline are written in a more general manner then the imaged can be sized based on other concerns. For example, if the current hardware setup prefers specific sizes and scales, those can be chosen for performance improvements. Similarly, most pipelines are sped up with smaller images. If the smaller image contains enough information for the pipeline to provide the information required of it, speed improvements can be gained by using smaller images.

In the Classroom we use quite small images, only 160×120. This may seem small, but for the task constraints that the pipeline has been given by the Classroom, this resolution provides ample information. However, the rest of the modules in the background subtraction person tracking pipeline were written in a general manner with respect to image size and scale. This means that if more precision were needed for, say, tracking the hands more accurately, higher resolution images could always be used, at a cost to speed. So for this pipeline, the grayscale input module produces images as requested by the Classroom and hands them off to the background subtraction module.

Background Subtraction Module

The background subtraction module does the initial localization of where a person could be in a scene. This module was inspired by the Perseus implementation on the robot [57]. This module operates like a motion detector with localization improvement. Instead of simply differencing consecutive frames to find motion in the scene, it actually builds up a representation of the background in the scene. By assuming that the background of the scene is relatively stable, this module is able to compare a given image against the background representation to determine which parts of the scene are foreground. Any portions of the scene which do not fit into the background representation are assumed to be foreground. The Classroom makes the constraint that there be only one person in the scene, so with

Figure 6.11: The functioning output of the background module and a person "burning" into the background on the right.

some exceptions this foreground will be the speaker. It should be noted that the one person constraint is a constraint on the pipeline, but not on this module. The major environmental constraint that this module relies on is a stable background. We will now look at how the module builds and uses its representation of the background, followed by some of the ways the Classroom deals with constraint failure on this module.

The background of the scene is represented in a retinotopic manner, that is each pixel of the image plane $I(x, y)$ maps to a single point in the background representation $B(x, y)$. Each of the pixels is then represented by a histogram of the values at that location over time. The resolution of the histograms is the same for all of the pixels in the image, but can be set depending on the needs of the pipeline, by a given bin width w. So this histogram $\mathbf{H}(x, y) = (b_1, \ldots, b_n)$ where $n = maxval/w$. The width of the bin determines the sensitivity of the background subtraction module to change in the image. In our case, $maxval = 256$, and we tend to use a bin width of 16, so $w = 16 \rightarrow n = 16$. So each pixel has a histogram with sixteen bins.

In order to obtain the histograms, this module watches images over time. Each time the module receives a new image, it updates the histogram for each pixel with their

respective values.

$$\begin{aligned} \forall n \neq \frac{I(x,y,t)}{w}, \mathbf{H}_n(x,y,t) &= \mathbf{H}_n(x,y,t-1) \\ \mathbf{H}_{\frac{I(x,y,t)}{w}}(x,y,t) &= \mathbf{H}_n(x,y,t-1)+1 \end{aligned}$$

This allows the histograms to build up a representation of the history of what the background has looked like over time. If the scene is relatively stable, this will allow the histogram to build up a peak which represents a range w of the most likely values of the background at that point. The rest of the histogram also gives the system information about what other values could be likely in the case of an oscillating background. We discuss these features later when talking about using the background representation to find foreground.

Before we can leave the construction of the background representation, there are two additional problems that must be discussed. First, the above algorithm resists change over time. That is to say, as t grows, with a stable $I(x,y)$, if some change happens it becomes harder for that change to affect the system. In fact, if we are using the maximum bin as the background, in the worst case, where the new background value has never been seen before, the time until that value is the new background will be $\Delta t = \max \mathbf{H}_n(x,y,t)$, that is, the largest value currently in the histogram. We have found that even a stable background will change over time for a number of reasons. These include people leaving or entering the room, changing the amount of light reflected into the background. Additionally as the speaker moves around the scene blocking different portions of the background the background will change from the camera adapting to the new white balance environment. In order to overcome this problem, we add an additional step to the algorithm for updating the background.

$$\begin{aligned} \text{If } \mathbf{H}_{\frac{I(x,y,t)}{w}}(x,y,t) &> b_{max} \\ \text{then } \forall n, \mathbf{H}_n(x,y,t) &= \frac{\mathbf{H}_n(x,y,t)}{2} \end{aligned}$$

Basically if the value of the just-incremented bin exceeds a maximum value, b_{max}, then we halve the value of all the bins. This allows the histograms to still represent the relative value of the values' histories, but still maintain a maximum time to change. In this case $\Delta t = b_{max}$. This means that the parameter associated with b_{max} is simply the longest amount of time that should pass before a change in the background becomes the new dominant background value. This leads to the second problem, which is the fact that if the foreground stops changing, it becomes background.

This feature is actually generally desired. In the Classroom, if a speaker brings an object into the scene and places it in the image, then it will become become the background over time. If the object does not ever move, this makes sense in terms of the vision task. Unfortunately, if the goal is to track the person, as it is in the Classroom, this can be detrimental if the person stands in one place for an extended period of time. In this case, the person will become part of the background. In order to prevent this from happening, the background subtraction module can be parameterized to not update the background representation. If the person stands in one place for an amount of time approaching b_{max}, the Classroom turns off the background subtraction.

One item of note here is the fact that the background subtraction module is not able to turn off the background updating on its own. It has no understanding of what its retinotopic output is being used for, and thus does not know when the person is standing still. Only the Classroom's reasoning system has this information, derived from later modules in the pipeline.

Stopping the background representation from updating prevents the person from

being "burnt into" the background, but it does mean that the background can not change any more. Since the background does change over time, for time of day and other lighting effects, this module can not be used for extended periods without the background being updated.

Now that we have examined the representation of the background, we need to look at how that representation is used to determine where the foreground is. The first concern is how the foreground will be represented for use by other modules. The representation used in the Classroom is a retinotopic map with a binary value at each image point, either foreground or background. An alternate representation is to give a value at each image point representing the probability that that point is foreground. We now look at three methods of obtaining these representations.

First we use a simple comparison. For every pixel we take the value with the largest bin from its respective histogram,

$$\forall n \exists n_{max}, \mathbf{H}_{n_{max}}(x,y,t) \quad = \quad \max \mathbf{H}_n(x,y,t)$$

If the value of the pixel for the current image falls into the bin with the largest value for that histogram, then that pixel is background. That is, $I(x,y,t)$ is foreground, iff $n_{max} \neq \frac{I(x,y,t)}{w}$. This method is nice because it is extremely fast to compute, but it has some disadvantages. If the background is slowly changing and is at a boundary (i.e. it has just changed) noise will cause the value of that pixel to vacillate over the bin boundary. This can cause a background pixel to become foreground, even though both of the bins that it vacillates between have a high count.

Another discrimination method gets around this problem by allowing more that one bin value to be the background. It does this by assigning a percentage p of the current maximum bin value to be the cutoff for a bin to be background. In this case, $I(x,y,t)$ is

foreground, iff $\mathbf{H}_{\frac{I(x,y,t)}{w}}(x,y,t) < \mathbf{H}_{n_{max}}(x,y,t) \times p$ It is important to note that this value must be proportionate to the current maximum, not the global maximum b_{max}. By dividing the histogram as the background settles, the actual value of the maximum bin goes up and down, but the proportions of all the bins relative to each other remains correct. This allows us to use the above equation to achieve a gentle transition from one background to another.

Finally, if the pipeline requires more detailed information about the foreground, rather than just a simple threshold, the background subtraction module can return an image with a percentage at every pixel representing the likelihood of that pixel being background:

$$P(I(x,y,t)) = \frac{\mathbf{H}_{\frac{I(x,y,t)}{w}}(x,y,t)}{\sum_n \mathbf{H}_n(x,y,t)}$$

This information can be combined with input from other modules and then run through a threshold module to make a more complex determination of where the foreground is. In the actual implementation of the Classroom, we stick to the simplest pipeline that meets all of the constraints to give the information in the fastest possible manner. Once this representation of foreground has been made, it is passed on to a connected component module.

Connected Component Module

Once the foreground has been extracted from the scene, the pipeline performs a connected component analysis on it. This aids in determining which parts of the foreground are actually the person. By examining the different connected pieces of the scene, the person tracker is better able to deal with noise.

The operation of the connected component module is itself fairly straightforward. It is a simple blob coloring algorithm. We scan the input in left to right, top to bottom order. The required input for this module is a binary image. The value 1 is colored, and

the value 0 is not. The output of this module is the input with each connected region, or "blob", is assigned a unique number. While doing this, this module also accumulated a number of statistics about the different regions, such as its size.

The coloring algorithm works in the following manner. The current color c is set to one, and the colored images is cleared, $\forall x, y, C(x, y) = 0$. Using appropriate boundary conditions, the following algorithm is used. If the current pixel is not foreground, that is if $I(x, y) = 0$ then go on to the next pixel. On the other hand if the current pixel is foreground, $I(x, y) \neq 0$, then we use the following set of equations to determine what the blob's color is.

$$
\begin{aligned}
I(x, y-1) = 0 \;\&\; I(x-1, y) \neq 0 &\rightarrow C(x, y) = C(x-1, y) \\
I(x, y-1) \neq 0 \;\&\; I(x-1, y) = 0 &\rightarrow C(x, y) = C(x, y-1) \\
I(x, y-1) = I(x-1, y) = 0 &\rightarrow C(x, y) = c \;\&\; c = c + 1 \\
I(x, y-1) = I(x-1, y) = 1 &\rightarrow C(x, y) = \min C(x-1, y), C(x, y-1)
\end{aligned}
$$

In the final case, where both pixels have a color, then additional work needs to be done. Both $C(x-1, y)$ and $C(x, y-1)$ are the same color now. We keep track of this information in a vector of all the colors. Once the above step is done, then this vector must be updated, $\mathbf{R}_{C(x-1,y)} = \mathbf{R}_{C(x,y-1)} = C(x, y)$. Once the algorithm is complete, the vector $\mathbf{R}$ is updated iteratively $\forall c, \mathbf{R}_c = \mathbf{R}_{\mathbf{R}_c}$ until each $\mathbf{R}_c$ has its "canonical" value.

Once this coloring algorithm has run, there will actually be more colors than regions. In order to solve this problem, we run through the image a second time, using the color vector to look up the true color of that region. While $C(x, y) \neq \mathbf{R}_{C(x,y)}$ update the color so that $C(x, y) = \mathbf{R}_{C(x,y)}$.

If the performance of this module degrades, there are a number of options

available to the Classroom. First, the background could have just changed on a large scale, rendering the background subtraction tracker useless. This tends to happen when a person walks into the scene for the first time causing a global white balance change in our Sony cameras. Another possibility is that a shade was opened changing the nature of the lighting in the room. Both of these are relatively small changes in background and will quickly be compensated for. However, the Classroom must know for how long the data quality will be poor. This module provides that information by informing the Classroom of the largest region size. The Classroom then ignores the tracking information until the size is reasonable for a person. It can speed the recovery process by having the background subtraction module run as quickly as possible, thus updating the background representation without bothering to send data to the rest of the pipeline. If, on the other hand, the data is not noisy and the Classroom requires faster tracking, the background representation can be sent directly to the person tracking module. This has the disadvantage of not localizing the large person regions, so the location information returned by the pipeline will become much more susceptible to noise.

One final way in which the Classroom can control the connected component module is to actually have it perform a first pass filter on the data. There will often be small foreground noise regions in the image. Since this module calculates the size of the region when recoloring, it can be parameterized to eliminate regions below a certain size. This will occasionally make the head disappear, so if a stable head location is needed, the full version must be used, or it must be told the location to not filter.

Finally, a retinotopic map representing the uniquely colored regions is handed to the person tracker module.

Person Tracking Module

Once the foreground regions have been labeled and the noise removed, it is up to the person tracking module to determine which blobs are part of the person, and where the hands and head are. This module actually uses a representation of what it expects a person to look like in order to make this determination. All of the components of what exactly a person is are parameterizable, however since they are all percentages based on the size of the person, they are fairly generic. In fact we have been able to use the same parameters on completely different scale images.

In order to find the person it must first determine which foreground region is the main part of the person. The first thing it does is throw away regions that are too large or small to be a person. It then uses the environmental constraint that people stand on the floor to begin searching for the person region. If it has been running for a while, it will already have an estimated location for the person region from tracking the previous locations of the user. Assuming that it has the tracking information, it chooses the region closest to the estimated person location. Otherwise it will simply choose the largest person-sized region in the scene, on the assumption that there is only one person-sized foreground object in the scene. Once it has done this it uses its person representation to determine if any other regions belong to the person.

The person representation relies on a number of constraints. First, the person must be standing on the ground. This means that the head is on top of the body, and the arms are at the side of the body. The specific regions in which the person tracking module searches for the head and hands are determined by the location of the central region, and percentages of its height. The head is the top most point of the region within the head search range. This prevents noise from above the person from causing too much disturbance in the head location. Once the head location has been found, a more accurate person height

is obtained and appropriate locations to search for the hands are generated. The hands in turn are understood to be the end of extremities from the body. Therefore if the person is blocking his writing with his body, the Classroom will not be able to see it. This is an acceptable constraint since the Classroom only wants to know when the person is writing where it can zoom in for filming skills, or when he is pointing at a location on the wall for virtual button skills. Other skills, pointing at classroom elements for example, would require different modules since the arm might not be away from the body from the camera's perspective.

Once the person tracking module has acquired the person, it improves its performance by tracking the person to estimate where he will be in the next frame. Our tracking technique is another example of needing to understand the constraints on the system in order to produce effective results. We have found that when the speaker is delivering a lecture his motion can be quite erratic. Because of this, unless he is standing in one place the prediction would never work for more than a few consecutive frames. In particular, the hand locations were almost impossible to track. With a frame rate less than ten frames a second, the user's hand could move from one end of the search region to the other. In the end, we decided to use a very simple prediction algorithm which was extremely cheap to predict the general location of the person since further processing is done—we concentrate on tracking the centroid of the person, since that will give the search locations for the head and hands.

The tracking mechanism this module uses looks at only the two previous locations of the person to determine the probable next location. It uses a simple linear assumption, $\Delta P = P(t) - P(t-1)$ and the estimated next location is $P(t+1) = P(t) + \Delta P$. This ridiculously simple tracking method works because of two assumptions. First, the person is most likely to do in the next moment what he was just doing in the previous moment. Any more complexity will be finding patterns where there are none. Second, when the person

is moving around the room, his motion tends to be extremely regular until he stops. One special case is when the speaker is standing still. Random noise can move the centroid of the speaker around to a small degree (standard Gaussian noise). In order to prevent these small aberrations from sending the tracker off in an incorrect direction, the successive deltas can be run through a simple low pass filter. We are looking at using Kalman filters [47] to reduce the noise further. Our current method prevents small amounts of high frequency noise from affecting the tracking when the user is standing still. One last point about the simplicity of the tracking algorithm is that is it only used to reduce the search time for finding the person region. Even if the tracker fails the module will still function, albeit a bit slower, by iteratively expanding the search area. This module shows that a simple speedup can often be the most effective.

Once the locations of the head, hands, and centroid have been obtained, they are sent through Gargoyle to the Skill Manager to be used by the next skill in the pipeline.

Alternate Modules

Examining the background subtraction tracking pipeline, it is clear that the Classroom has many options for updating how the pipeline is running. From changing parameters, as in the maximum bin height for the pixel histograms, to changing how a routine operates, like when it turns off the background updating. It also can make structural changes to the pipeline, like removing the connected component module for a speed increase. We have found this pipeline to be so fundamental to the functioning of the Classroom that we have developed a number of other variants on this pipeline.

We can add an edge detector module to increase the localization accuracy of the person. One problem we have is that the person can cast shadows on the wall which throw off the tracking. Since the back wall of the Classroom is textureless and the person's

shadows tend to be soft, the output from this module is combined with the output from the background subtraction module, and a new person location image is created with only regions that are both not background and have edges handed to the person tracking module. This provides much better localization on the hands and legs, but is more computationally expensive.

Another problem we have found is that if one of the fluorescent ceiling lights happens to be in the scene, it will flicker causing the background subtraction routine to think that the light is foreground. In order to solve this, we combine the output from the background subtraction module with a simple threshold module. The light clips at the white end, and is easily filtered out.

Finally, if noise needs to be filtered out, but the connected component module is too expensive, it is replaced with a simple morphological erode module.

These alternate formulations of the person tracking routine are simple alterations that change how the routine operates without changing the information that the routine returns. That is, the alternate modules use different environmental constraints, but still use the same task constraints. Currently the Classroom only uses these alternate modules when specifically instructed to do so. We want to have it alter the vision routines by its own constraints in the future. Another formulation of the tracking routine which does alter the task constraint is color tracking of the person.

6.5.2 Color Tracking

The background subtraction pipeline just described provides accurate data about the location of the hands and head and centroid of the speaker. It is able to do this by using a very strong environmental constraint: the background must be relatively stable. The Classroom has a number of modifications it can make at its disposal in order to account for small

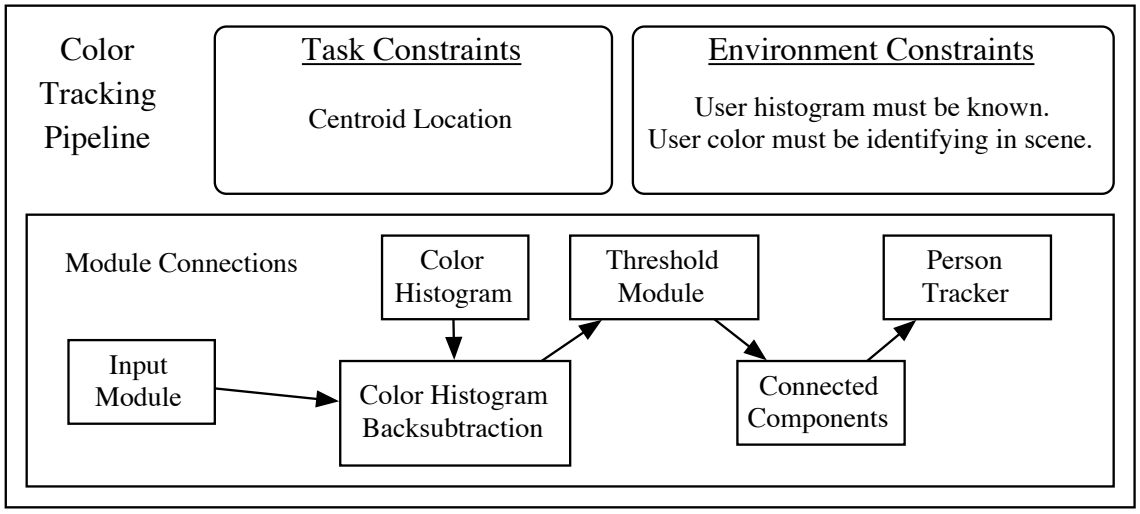

Figure 6.12: The color tracking pipeline and constraints.

changes in the background; however, there are times when that constraint does not hold. The most obvious example of this is when the camera is moving. In our setup, the vision camera seldom needs to move, since we have a second camera for filming; however, if the user walks out of the view of the stationary camera, a movable camera will obviously need to be used to track the user. Additionally, there are instances when the background is constantly changing because of lighting conditions outside the control of the Classroom, such as light from a window on a windy day. Clouds passing in front of the sun can make regular changes in the background. For all of these reasons, we built an alternate tracking pipeline, this one based on color.

In addition to using different environmental constraints, the color tracking routine operates on a different set of task constraints. Due to the imprecise nature of the color tracking, exact location for different body parts are harder to achieve, although it does provide good localization of the centroid. We now examine how this routine operates in detail, again with an eye toward understanding the constraints on which the pipeline relies. Many of the modules are the same module used in the background subtraction pipeline. In

this case we only look at how they interact in this pipeline and refer to Section 6.5.1 for details on the operation of that module. This pipeline consists of six modules, a color input module, a color histogram matching module, a threshold module, possibly a connected component module, and finally a person tracking module.

Color Input Module

The color input module is parameterizable in the same way as the gray scale input module. Since color matching is a local operation, and we have already seen that none of the other modules in this pipeline rely on a specific scale or size, specific regions of the scene can be used, and smaller scales can be used in order to speed up the processing. Once the color image has been obtained it is passed to the color histogram matching module.

Color Histogram Matching Module

Using color for tracking requires that two constraints hold. First, the model of the object to be tracked, that is its color histogram, must already be known. Second, those colors must be relatively unique in the scene, that is color must be a good discriminant if you are going to use color to track the object. The first constraint is interesting because that is entirely dependent on the system using the pipeline. Basically this pipeline can not be used unless the histogram of the object to be tracked has already been obtained. The Classroom must therefore have a color histogram of the speaker before utilizing this pipeline. This is a crucial point, but in order to describe the operation of this module we will assume that the Classroom already has that model of the speaker. The Classroom monitors the second constraint in much the same way that it monitors the stable background constraint, by observing the output of the pipeline. We will examine both of these constraints more later, first we examine the operation of this module.

This module utilizes a model of the object to be tracked based on its color histogram. The histogram is a three dimensional space, with each dimension divided into n bins giving n^3 total bins in the histogram. Each possible color maps into these bins in the usual manner by taking the value of each axis of the color space and determining which bin it goes into for the corresponding axis of the histogram.

$$C_{r,g,b} \rightarrow \mathbf{H}_{C_r/n, C_g/n, C_b/n}$$

The value at each position in **H** represents the percentage of the object that contains that color. Color histograms have been used for object recognition and localization before, initially by Swain and Ballard [85], and a number of techniques for using this color histogram model of the object were available to us. We implemented two methods, a simple percentage threshold method, and a slightly more general color histogram backprojection method.

The threshold method is extremely cheap computationally, requiring only one pass through the image. At each pixel in the image, the value of that color is looked up in the histogram. Then the corresponding location in a single band image is set to be the value retrieved from the histogram. This new image contains the relative likelihood that each point is part of the tracked object. This simple algorithm works better when the tracked object has only a few colors. If this is true, then simply thresholding these percentages will give the location of the object. This is fast and effective in the Classroom if the speaker is wearing a solid-colored shirt. If this is the case, the centroid of the shirt, and thus the speaker's torso, is easily found. Unfortunately it leaves smaller sections of the histogram, say the hands and head of the speaker, as much less likely to be the foreground, as seen in Figure 6.13. One possible solution is to find different histograms for different body parts, though this increases complexity. Another solution for more accurate color tracking is to use color histogram backprojection.

Color histogram backprojection, developed by Mike Swain [84], provides more

Figure 6.13: Results from the color histogram backprojection module.

detailed tracking of the object in the scene. Rather than simply looking at the histogram of the object being tracked, it also compares it to the histogram of the given image. This takes into account all of the colors of the tracked object rather than just the dominant ones. In order to do this, a new histogram is made, the ratio histogram. The model histogram M is divided by the histogram of the current image $I(t)$ to generate the current ratio histogram,

$$\forall i, j, k, \left(\mathbf{R}_{i,j,k}(t) = \frac{\mathbf{M}_{i,j,k}}{\mathbf{I}_{i,j,k}(t)} \right)$$

This histogram is then backprojected into the image in the same way that model histogram was in the threshold variant. In this case, if the likelihood of a pixel being chosen as part of the foreground depends on not just how much color is in the model, but also how much is in the image. If they both have the same amount, then the likelihood that it is the object being tracked is much higher. While this allows more precision in the tracking of parts of the person, it also means that it is more susceptible to small noise that may correspond to a color that is only somewhat represented in the model. Fortunately the rest of the pipeline cleans up most of that noise, since this probability image is going to be fed to the person tracker module.

Now that we have examined how the histograms are used, we must ask how the histograms are generated. The simplest method is to have a pregenerated histogram for the

speaker. If this is not available, the system must generate the histogram from the image somehow. In order to generate the histogram, the system is fed color pixels known to be of the object to be tracked. For the Classroom this means that the speaker must somehow be segmented. If this is not available, the classroom can automatically generate the histogram during the initialization phase by requesting that the user stand in a specific location in the initialization phase so his color can be extracted. A less intrusive method uses the fairly precise results of the background subtraction module to segment the person and then feed those to a histogram generator as described in Section 6.5.3. Once the histogram has been made for a specific run, it generally does not need to be rebuilt for the rest of the lecture.

Once the probabilities that each pixel are the person have been found, that probability image is passed on to the threshold module.

Threshold Module

Since the color histogram module returns a probability image and the person tracking routines need binary images, the image is run through a simple thresholder. This routine must be parameterized by the classroom based on the nature of the histogram if the simple histogram matching routine is used. This is because the more colors the object has, the weaker the strong probabilities will be. Deciding how the threshold must be tuned can be done by examining how noisy the output of the connected component module is.

Connected Component Module

Just like in the background subtraction pipeline, this module is used to help localize the person. Generally in the Classroom the speaker is such a different color from the rest of the room that this step can actually be done away with. This is partly because of the uniformity in the colors in the Classroom, but also because of the poor nature of the

information being given by the color tracker. If the task constraints call for precise hand and head locality information, the backprojection method suffices, but a better tracking pipeline would be preferred. Figure 6.13 shows the poor color results for arm and head tracking of the histogram threshold method. The connected component module passes its labeled image on to the person tracking module.

Person Tracking Module

This person tracking module works in many ways like that in the background subtraction module, however it must be parameterized slightly differently for use in the color tracking module. First, since the color tracking is generally only useful for the person's centroid, the additional body part searches (hands, head, and feet) can be turned off. Second, if the connected component module is not used, the person tracker simply returns the centroid of the color regions. Finally, if it is being used on a moving camera, the person location will only be given in image coordinates, and if absolute coordinates are needed, additional processing will be needed (which will require the camera's physical parameters). Since the goal of centroid tracking is to just keep the camera on the person anyway, we have found that a simple relative orientation skill loop works quite well.

6.5.3 Automatic Color Histogram Generation

In order to use the color based person tracking pipeline, the system must have acquired a color histogram of the user. In our experiments we have used pre-built histograms, however we have also experimented with a number of alternatives, which would allow the Classroom to build one as needed. Having the Classroom automatically generate the user's histogram allows less configuration time as well as dealing with unexpected changes in the histogram, for instance, if the user removes a jacket part way through a talk.

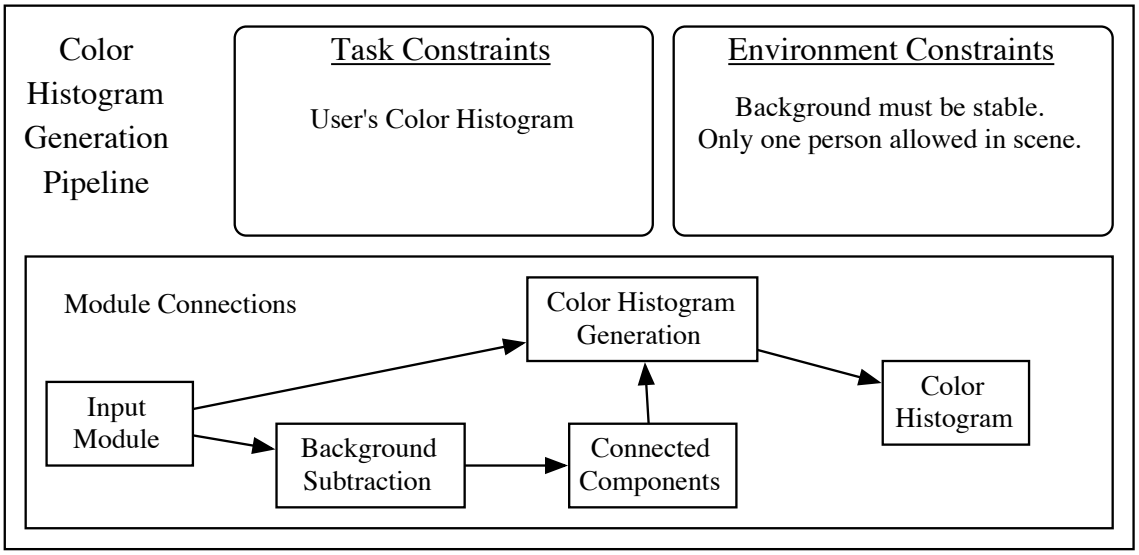

Figure 6.14: The pipeline and constraints for the pipeline that uses background subtraction to automatically generate a color histogram of the user.

Color Input Module

The color input module generates images that are passed to two modules, the background subtraction module, and the color histogram generation module. Since the background subtraction module requires grayscale input, that branch of the pipeline is run through a color to grayscale converting module.

Background Subtraction Module

If the background subtraction tracking routine is already running, it uses the previously generated background eliminating the usual startup time. Other than that, this module is identical to the one used by the background subtraction person tracking pipeline. The output of the background subtraction module may have some noise as usual, however for this purpose, obtaining the complete extent of the person is less important that making sure that all the pixels designated foreground are part of the user. In order to aid in this,

the background subtraction module output is passed to a connected component module.

Connected Components Module

The connected components module is the same one used with the background subtraction person tracking pipeline. In order to lessen the effects of false positives in the color histogram, the connected components module is parameterized to ignore all but the largest region. This can eliminate some poorly connected foreground components, like the arms or legs, however the color tracking pipeline only provides centroid information. This means that a histogram covering the bulk of the user's colors, not necessarily including all the colors is sufficient.

Color Histogram Generation Module

The color histogram generation module takes a color image, reduced color space used by the color histogram back projection module, and the region of interest to histogram. In this case, the region of interest is the foreground region recovered from background subtraction and cleaned up by the connected components module. In order to generate a complete histogram, it needs to obtain a minimum number of pixels from the color images. This is done very quickly by allowing the pipeline to run for a few consecutive frames. We have found that as few as two to three frames are required to generate a new object histogram that will give good results (about a tenth to a fifth of a second). Once the histogram is built, it can be saved to disk for later use.

When the camera in the Classroom is pointing straight forward in the Classroom, we have found that the majority of the pixels are the white color of the back wall. An alternate method of building a color histogram uses this constraint to build a histogram by adding all of the colors that are not white. This assumes that the person is the only non

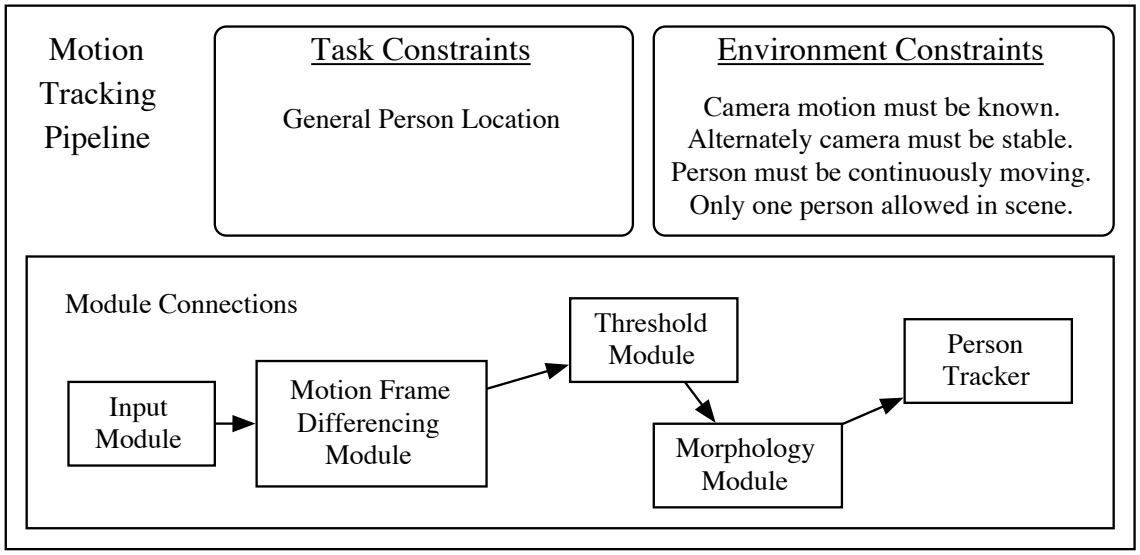

Figure 6.15: The pipeline and constraints for the motion tracking pipeline.

white object in the scene. This alternate version of this pipeline proved useful for the Image Mosaic project described in Chapter 7.

6.5.4 Motion Detection

The color tracking pipeline is very effective for finding the centroid of the person, when two constraints hold: the speaker is wearing colors that are fairly unique in the scene, and the Classroom has segmented the color histogram for the user ahead of time. There are times when these constraints do not hold, but the centroid of the person still needs to be known. We have developed yet another vision pipeline for tracking the person, this one based on motion. The motion pipeline is very similar to the color tracking pipeline. The key difference is that instead of using color histograms, motion is used. The obvious constraint on using this pipeline to track people is that the person must be continuously moving for this routine to work. Since motion in the scene is ephemeral, this can not be used for complete person tracking. In fact the only information it returns is the centroid of

motion, which may very well not be the center of the person.

The motion detection module operates with a simple frame difference,

$$M(x, y, t) = |I(x, y, t) - I(x, y, t - 1)|$$

This provides an estimate of locations in the image where there is motion from a stable camera. This algorithm is very susceptible to noise. An alternate method would be to use motion fields which could be correlated to camera motion to improve the localization of the speaker.

6.5.5 Template Matching

The final Classroom pipelines we discuss in this dissertation are inherently different from the others. The other pipelines all have the capability of controlling some process. This means that they must adhere to the real-time constraint. We use template matching to identify icons that the speaker has drawn on the board, as well as to initialize the location of white board if it is not already known. Since both these tasks return a single piece of information that the Classroom then enters into its memory (unless the speaker draws on that portion of the board again) it can be somewhat more computationally expensive without hurting the overall performance of the Classroom. This is good due to the expensive nature of template matching.

The Classroom has two main options for template matching pipelines, Hausdorff matching and the Hough transform. These are both well understood techniques [53, 9]. An optimized version of the Hough transform is used to find the board and squares surrounding the icons, which the Hausdorff technique is used to identify the icon, once it has been localized by the Hough transform.

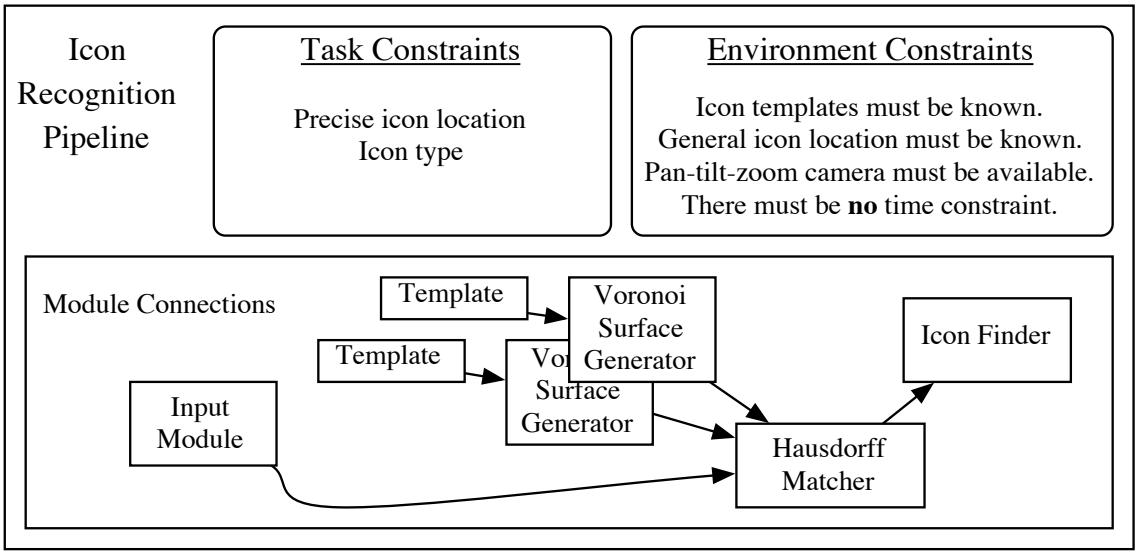

Figure 6.16: The pipeline and constraints for the Hausdorff matching icon finding pipeline.

Hough Transform

The Hough transform module takes the binary output from an edge detection module. This is the scanned for squares by finding vertical and horizontal lines. The parameter space is simply the x location of vertical lines and the y location of horizontal lines. The parameter vectors are then generated as follows:

$$\forall x, \mathbf{X}_x = \sum_y I(x,y) \text{ and } \forall y, \mathbf{Y}_y = \sum_x I(x,y)$$

Then the parameter vectors $\mathbf{X}$ and $\mathbf{Y}$ are used to determine where the vertical and horizontal lines in the scene are. This is actually fast enough that it can be used to visually servo the camera over the icon for recognition by the Hausdorff matcher since at the low resolution of a wide angle shot, the icon is unrecognizable.

Hausdorff Matching

Once the icon has been localized, Hausdorff matching is much cheaper because the scale and orientation are known. To match the saved template, the Hausdorff matching module has two options. It can use a single template for the icon and attempt to match the entire thing, or it can use multiple icon elements, such as corners, to find a range of icons. The matching process works as follows.

Every point in the image I covered by the template T at a given translation t. The points covered by the template in the image are $I(T \oplus t)$. For each point in the template, find the closest point in the image then calculate the Euclidean distance between them. The largest of these is the directed Hausdorff distance from the template to the image, and a measure of how well the template matches the image:

$$h(T, I(T \oplus t)) = \max_{a \in T} \min_{b \in I(T \oplus t)} |a - b|$$

Total Hausdorff distance is the maximum of the directed Hausdorff distance from the template to the image, and vice versa:

$$H(T, I(T \oplus t)) = \max\left(h(T, I(T \oplus t)), h(I(T \oplus t), T)\right)$$

Both the input image and the template are inputs to this module. In order to speed up this calculation we use a Voronoi surface which is precomputed for the templates. We use a small set of pregenerated templates, however "learning" new templates is simply a matter of drawing the new icon on a blank board. This module returns an image which holds the Hausdorff distance at each point. The two different formulations of this module either use this directly to determine the type of the icon, or combine the output from a number of Hausdorff modules to determine the locations of features of the icons to allow for more variation in the shape. Since the number of icons that we recognize is low, discriminating between them using a straight matcher is effective.

6.6 Final Thoughts on the Classroom

Intelligent environments has been an area of very active research in the last decade. From television studios [74] to playrooms [12] many researchers have looked at putting more smarts into the world around them. Most of these systems utilize vision in one manner or another. They all rely on constraints in their environment to be able to have effective vision. What sets our system apart is that it actually has an understanding of what those constraints are, and how to change its sensing when those constraints fail. We feel that this methodology provides numerous advantages in terms of expandability of these systems. Intelligent environments are the ideal location for reasoning about the constraints used by the vision system, because of the very nature of the environments: they are engineered and can be well understood. These are great advantages for our methodology and allow it to succeed.

The Classroom has proven itself to be an excellent environment for both plan recognition and vision research. The routines written for the Classroom have been used many times and even in different locations. We have taken the Classroom hardware to conferences and demonstrated it in different situations and configurations. In one situation the side of the environment was open to viewers, changing the vision environment considerably. However, the fundamental tasks have remained the same. In an effort to expand the test of our vision system, we built another vision system which required similar information, but for very different tasks—the Interactive Image Mosaic.

Chapter 7

The Interactive Image Mosaic

The Interactive Image Mosaic is an electronic art installation built by undergraduates at Northwestern University using Gargoyle. They built is using many existing modules and one module built specifically for this project. The system constructs an image mosaic which is an image made up of many smaller images which, when viewed from a distance, resembles another image. In this way it is similar to grade school students pulling apart magazines to find the right picture to paste into a mosaic. It constructs this mosaic by using a piece of art as the base image that it is trying to duplicate in the mosaic. It then takes images from a camera and compiles those images into a mosaic that resembles the original work of art, as shown in Figure 7.1.

The students provided a mechanism that allows the viewers to interact with the artwork and feel like they are a part of it. In order to make the interaction more interesting than simply making the viewers image part of the mosaic, they allow the viewers to record their thoughts. Then when the new viewer's image replaces an old one, the system plays back the previous viewer's recording, allowing the new viewer to respond if he wishes. To make the interaction as natural as possible they decided to use computer vision to control the

Figure 7.1: A diagram of the eventual installation, as it should appear in a museum, with a user interacting with a partial mosaic by way of a camera positioned to film a person viewing the artwork.

interaction. Building the required routines would have been a difficult and time consuming task, well outside the scope of an undergraduate project. However, by using the Gargoyle framework they were able to make use of a number of modules built for the Intelligent Classroom.

7.1 Background

While the desired interaction with the system is fairly straightforward, it still requires a good deal of vision processing. The Gargoyle pipelines which we had designed for the Intelligent Classroom were able to be quickly reconfigured with this new set of tasks, and integrated into the new system. By combining modules built for the Intelligent Classroom and an additional image mosaic module built for this task into routines tailored to their tasks, we were able to quickly construct a set of special purpose routines suited for the

new tasks and constraints. Using these routines and constraints, the students were able to combine them into a sophisticated system which easily allows users to interact with it.

The interaction begins when the viewer steps in front of the installation as diagrammed in Figure 7.1. The viewer is presented with an artwork and a computer generated mosaic of the artwork made up of previous viewers. Once the system has noticed that the user has stepped into the scene, it takes a picture and decides where to place it in the mosaic. When one viewer replaces another, it plays the recording made by the previous person if available. It then records any response that the current viewer may have. We will look at the vision routines required by this system in a moment, but first it is worthwhile to look at the reasoning required by this system. As it turns out, very little reasoning is required at all. This reasoning is easily represented by a simple state machine, which determines which action to take next.

The class designed and implemented the complete system, although it was never shown in a museum environment. They concentrated on the video and reasoning aspects of the project, and simply used Gargoyle as a part of their system. A quick glance at the system diagram in Figure 7.2 shows how this was used to make a complete vision system using Gargoyle for a completely new task.

7.2 Execution System

The tasks required by the Image Mosaic system are quite linear in nature, and don't require a powerful reasoning system to change routines on a moments notice. The system layout for the Interactive Image Mosaic project is shown in Figure 7.2. The students were also interested in using speech recognition for selecting videos to play back, so the reasoner starts that system when requested. Instead of building a complex reasoning system to decide which actions to take next based on the current state of the world, the students built

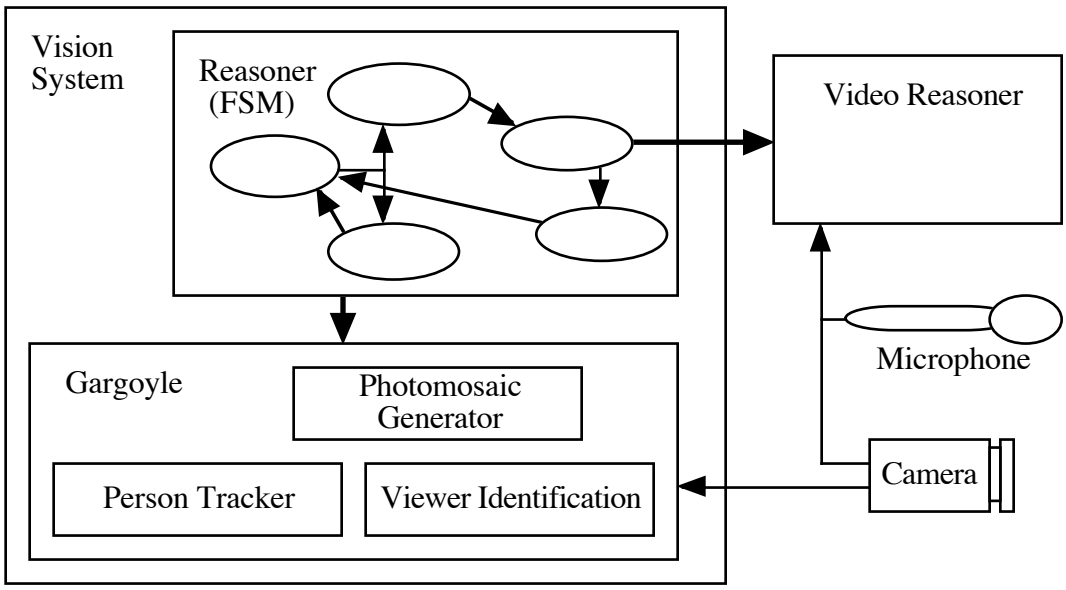

Figure 7.2: The system architecture for the Interactive Image Mosaic project.

a simple state machine, shown in Figure 7.3. The machine waits for specific events to send it to the next state. When it switches to the new state, the actions that the old state was taking are stopped, and the actions specified in the new state are started.

For example, suppose the system is waiting for a user to enter the scene in the start state. It waits until the viewer identification routine (described later in Section 7.3.1) signals that a person has entered the field of view. It then stops the viewer identification routine, and starts the picture placement routine. All of the states and their respective vision routines, along with the possible transitions are shown in Figure 7.3. Note that once a person has entered the scene the system watches for the person to leave the scene in parallel with whatever else it is doing, so it can abort operating and return to the waiting state as soon as the person leaves.

This simple state based execution system only uses task constraints to decide which special purpose routine to run. However, even though the environmental constraints

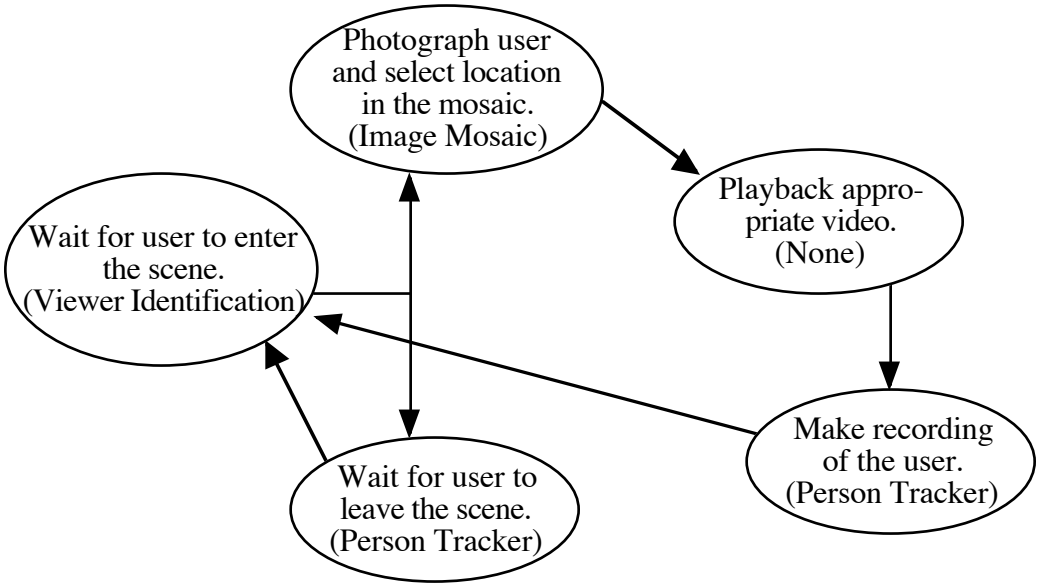

Figure 7.3: A simple state machine for controlling the Interactive Image Mosaic, showing which routine is used for each state.

are not represented in the state machine, they are still used by the system. Since the exact structure of the state machine depends on the installation, which routines go with which state will also depend on the constraints inherent in the installation. The installation is very constrained in terms of lighting and the location of people relative to the camera. This means that many of the constraints are enforced by the structure of the world rather than reasoned about. For example, in some installations a colored background is used. If this is the case, a color-based viewer identification routine is used, because the stable background color constraint will hold. However if this is not the case, the less accurate motion based viewer identification routine used in the Classroom can be used instead. Thus the constraints of the vision routines are still used, just not encoded in a reasoning system; rather, they are encoded by the person doing the installation when it is being put together. Even very simple planning mechanisms can still provide good vision, as long as you are careful with the constraints.

The next section looks at the vision routines used by the Interactive Image Mosaic project.

7.3 Tasks and Vision Routines

Once the state machine was built, it was a simple matter of connecting it to the Gargoyle server to use the special purpose vision routines for the different tasks. Many of the tasks require routines which were easily constructed out of existing modules built for the Intelligent Classroom. This section examines those routines and how they were designed using the known constraints on existing routines and pipelines. The execution system uses the results from these pipelines to switch states or to control physical elements of the installation. Figure 7.3 shows which pipeline is used for each step in the state machine. Information on the Person Tracking pipeline can be found in Section 6.5.1.

7.3.1 Viewer Identification

Before the system can start interacting with a user, it needs to know if there is a user in the scene. Since the space where the viewer can interact with the system is limited, and has a set background, there are two simple routines which it uses to recognize that a person is standing in front of it: motion detection and background color subtraction.

If the background color of the image is known, then the color histogram background subtraction module can be used to determine when a person enters the scene. The pipeline that is built to utilize this module needs to be slightly different from the color tracking pipeline in the Classroom. Rather than using the centroid of a positive threshold based on the known histogram, a negative threshold is used, and the region that is not the known color is where the person is. Figure 7.4 shows the alternate color tracking pipeline. When a set amount of the known color background changes to a new color, then the pipeline informs

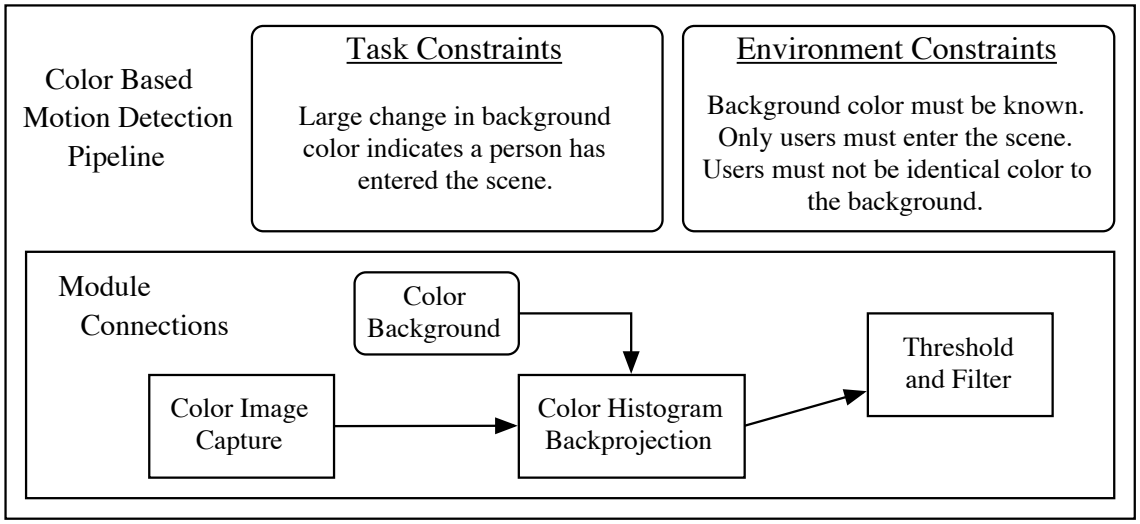

Figure 7.4: An alternate implementation of the color tracking pipeline built to determine when someone has walked in front of the camera.

the state machine, and it enters the new state. The amount of different color is computed by the "Threshold and Filter" module which thresholds the color (doing some morphology to eliminate noise) and sends the message to the state machine when it determines that a person has entered the scene.

Depending on the installation, the background color may not be known. In this case the motion tracking pipeline from the Intelligent Classroom, as seen in Section 6.5.4 is used instead. That pipeline is altered slightly with the addition of the Threshold and Filter module where the Person Tracking module would otherwise be. With motion tracking it is more difficult to know when a person has left the scene and a new one entered, but it is still sufficient for informing the state machine to begin the process of adding a new viewer to the mosaic.

Once the system knows that there is a person standing in front of it, it takes a picture and integrates it with the mosaic.

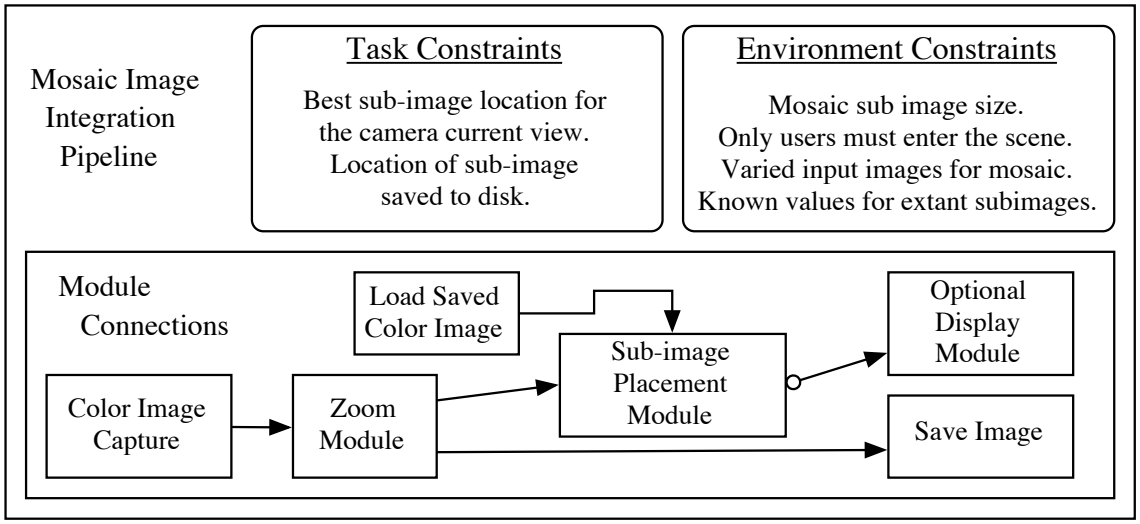

Figure 7.5: A pipeline for figuring out the best location for an image in a mosaic representing another image.

7.3.2 Mosaic Image Integration

The image integration step is actually quite expensive, and while there was not a pipeline already built for this step, many of the modules built for other routines were able to be reused. In fact the only module that needed to be written for this routine was the image mosaic image location selection module. The rest of the modules shown in Figure 7.5 were already built, and combined to form this pipeline.

This pipeline works by loading the artwork from disk, and taking a picture of the viewer. The zoom module scales the image of the viewer to be the correct size for the image location module. That is, the scaled image must be the same comparative size as it will be in the final mosaic. This allows the mosaic construction module to compare the captured image directly to the different possible placement locations on the stored artwork.

A number of different methods for constructing image mosaics have been developed using different image matching algorithms. One popular method is using wavelet based image matching as described by Jacobs [54]. Other methods are suggested by Finkelstein

and Range [29], Hausner [49], Kim and Pellacini [61], and the patented techniques by Silvers [82, 81]. The module built for this pipeline uses a much simpler color and shape matching technique inspired by the non-wavelet based method used in the Metapixel system by Mark Probst.[1] We experimented with the wavelet comparison method as well as the above method, however similar results were achieved with both methods in terms of visual appearance. The comparison metric was implemented as a library making it easy to exchange comparison functions if the wavelet method is desired.

By comparing the current image, I with the artwork, A, at every possible offset, t, the image mosaic module picks the best possible location for the image in the mosaic. Each possible mosaic image location is given a value, $V_t(I)$, depending on the current image. In our system the offset is constrained to be an integral of the image we are placing:

$$t = (n \times (I_{x_{max}} \bmod A_{x_{max}}), n \times (I_{y_{max}} \bmod A_{y_{max}}))$$

The image is zoomed to be smaller than the artwork by an integral factor, making this work evenly. The simple comparison that we use is:

$$V_t(I) = \sum_{x=0}^{I_{x_{max}}} \sum_{y=0}^{I_{y_{max}}} |[A \oplus t](x, y) - I(x, y)|$$

The location with the smallest $V_t(I)$ is then chosen. Optionally, if the new score is higher than the previous score, the next best location can be chosen. This method is less than optimal, however the images that are used to generate the mosaic are not known a priori so the optimal solutions are not possible our this case.

Since most of the images coming into the system are of people standing in front of the camera, the system may not have a sufficiently varied set of images to generate a good mosaic. This is especially true as the mosaic is being filled the first time. One solution

[1] http://www.complang.tuwien.ac.at/~schani/metapixel/

Figure 7.6: The progress of building an image mosaic, some images are replaced, and some new ones are added.

is to look for subimages from the camera rather than using the entire image. Figure 7.6 shows that as a mosaic is made, some images may be replaced rather than unused locations being filled. Forcing the routine to fill the image first works, but may result in poor quality mosaics. The solution the class uses is to alter the mosaic generation slightly. Rather than use the current image directly, they filter the image for the background color (if it is known). This color is then filtered out, and just the person is superimposed onto the image location to make the mosaic. This enhances the viewer's ability to control where his image is placed by moving his body into different positions in the scene.

Finally, the system may use an external display, in which case it will use the sub-image locations returned by the pipeline to build the mosaic on its own. On the other hand, an optional display module can be used instead to allow Gargoyle to display the mosaic.

7.3.3 Filming the Viewer

While the system is deciding where to place the image it captured of the user, it also makes a video recording of the viewer giving his thoughts on the piece. Depending on the installation the filming may be as simple as starting a recording process, or it could need to determine where the user is, and then point the camera at the appropriate location. Filming is a task which we already implemented with a slightly different set of constraints in the Intelligent Classroom using the person tracking algorithm described in Section 6.5.1. By figuring out when the different constraints hold in this new system, we were able to make use of the same routines. With a fixed camera, however, no vision routine is needed. This part of the project is mostly devoted to speech recognition, to determine which other recordings are most related to the viewer's, so the vision routines have not yet been integrated.

7.4 Conclusions

This project has gone through a number of different phases. It started as something fun to do with Gargoyle and was installed at the front door of the InfoLab. It then became a quarter project for the undergraduate students, John Didion, Ken Suh, Muon Van who turned it into a real system. It was later revived as a project looking into using voice recognition to get the context of video clips with Jon Card, Jason Friedlander, Shovahn Rincon, Adam Tarkowski, Angelo Gonsalez, and David Smock. I want to thank all of the undergraduates for their work. It was very exciting to see different people use Gargoyle as part of their project, and to consider the possibilities of what they could use it for. It was especially exciting to see someone use it as a tool to accomplish their goals, since that is what it was designed for.

One direction we would like to take this work, is to make the image mosaic

pipeline more active. Rather than simply taking the picture that is in front of it, have it search the room for colors and other features that it knows it needs. This could be a very interesting use of Gargoyle, allowing it to heavily modify that pipeline as it runs.

The work done so far shows that by thinking about the constraints on vision routines and pieces of vision routines it is easy to reuse them. The vision system that we designed for the Classroom was easily reused in a different environment in a simple and straightforward manner. This was due to the fact that the constraints were understood, and the routines were broken up in such a way that they could be reconfigured for different environments.

Chapter 8

Related Work

The computer vision methods used in this dissertation cover a wide range of research topics, from robotics to intelligent environments to computerized art. In addition, the thesis: that computer vision can be made more successful by reasoning about explicitly represented constraints to decide which special purpose vision routines to use when, covers the entire range of computer vision research. Many other researchers have used special purpose vision routines to achieve their goals. Indeed, we rely on this research to provide routines which utilize different constraints in order to cover as many situations as possible. It is this very breadth of special purpose routines which allow our methods to be viable.

Because of this there is a good deal of work from many areas of computer science which relates to our methodology, from work on new computer vision routines, to work on new reasoning systems. This chapter examines some work in both of these research areas. In particular we examine vision systems which utilize vision to solve specific high level tasks. We also look at classes of reasoning systems used in real world architectures and how suited they are to reasoning about vision. Finally, we examine a number of other computer vision frameworks which have been developed and compare them to Gargoyle.

All of the vision systems that we examine in this chapter use constraint-based, special purpose routines. The reasons for this are discussed in Chapter 2. Robotics and intelligent environments are two areas that use special purpose vision to achieve their goals. In robotics, there are many different tasks that the system is trying to achieve, so they will often use multiple special purpose routines to accomplish their goals. Looking at the field of intelligent environments, this seems to be less true. Rather than solve a number of different vision tasks, they seem to stick to tracking people. This means that while the range of vision tasks they accomplish is smaller, the complexity of the routines and constraints that they rely on is higher.

8.1 Automated Control

The following systems use some sort of reasoning system, be it learning or planning, to decide which routines are most appropriate for the given situation. Different systems place different emphasis on reasoning about how to sense, or controlling the context in which the routines are operating. In every case however the systems switch which routine they are using for given tasks, with more or less reasoning.

8.1.1 University of Rochester: Driving System

Gabris Salgian at the University of Rochester has recently built an autonomous driving system which controls a photorealistic virtual reality driving simulator [79]. Driving is an interesting domain in which to reason about how to sense because there are many sensing tasks that one may need to accomplish while driving. Specifically, their system detected a number of features important to driving such as traffic lights and stop signs. In addition, they also detect looming objects, such as other cars or pedestrians. Each of these tasks was accomplished with a dedicated vision routine, or set of routines.

Their system operates in four cleanly separated layers. They use basic features which in turn are fed into the different visual routines. The visual routines in turn drive the behaviors which are selected by a high level scheduler. The high level scheduler decides which behaviors to give more time to, depending on what the state of the world is.

The routines in this scenario are all tied to specific behaviors. This means that visual routines are activated and deactivated by the routine scheduler. Many of the visual routine results are utilized by multiple behaviors allowing for some simplification. This system is an interesting example of special purpose vision routines being used to accomplish specific tasks, however, unlike the Intelligent Classroom they are for the most part constrained to using all of the routines at all time instances in their current system. As they expand the set of behaviors they have implemented, the timeslices available for the scheduler will diminish. This is as opposed to our system which modifies the operation of the actual visual routines based on the environment in addition to the task.

8.1.2 MIT: Kidsroom, Smart Studio

A large amount of work in computer vision has been coming out of MIT relating to constrained environments. These systems have shown effective use of special purpose vision in constrained domains. Each of them shows an interesting use of constraints. This section briefly looks at each of these systems.

Kidsroom

The Kidsroom is an interactive environment in which children are engaged in a participative storytelling activity [12]. The room interacts with the children through a number of well placed cameras and screens on the walls of the "bedroom". This interaction is led by the system which has a story that it runs the children through. It engages the children by

having them play out certain rolls in the story. The children do not, however, lead the story.

This is a very interesting tactic, in terms of constraints in the environment. It allows the system to control the context in which it is operating. When it tells the children to perform a specific action, it only needs to be able to respond to that action, or a lack thereof. This is also used to aid the computer vision used to interact with the children.

This has some of the same requirements that we do. For example, if they know to look for specific actions on the part of the children, they can match the motion to specific motion templates [27]. However, all of the context is used in expectation. If the story requires that the children stand at a specific point and jump, the system knows that it needs to look in that point using a jump action template. By telling the children where to go and what to do, the problems of reasoning about how to sense, becomes simply using the correct routine for the requested situation.

Smart Studio

Some earlier work by Bobick and Pinhanez used the same idea of controlling the action, but instead of having the computer control the action, it is run by a human in in the loop. The Smart Studio system aimed at generating television quality film of a cooking show [74]. The cooking show provides a well constrained environment in terms of computer vision. Each of the cameras was well placed for each of the required shots, as is usual for a television studio. The computer vision task is then reduced to looking for specific motions at specific points, and panning and zooming as appropriate.

With a number of cameras, the reasoning of what to look at is a nontrivial problem. The Smart Studio approach to this problem is to put a director in the loop. Rather than having the system decide what to look at, it offloads that computation to the

person in the loop. The director issues directions to the system which it can then implement. For example, by telling a specific camera to zoom in on a requested area, the vision system only needs to deal with looking for that area from the requested camera. They also rely heavily on the fact that specific items will always be in specific locations from a given view. This allows high quality films to be made with few operators.

The notion of using a human to provide direction is very exciting, and could clearly be used in the Classroom. Some unpublished work is currently being begun by Tsotsos at York University along these lines. By having the human being filmed be able to attract the "attention" of the system, they hope to improve the automatic filming task.

8.1.3 RECIPE

Arbuckle and Beetz developed this system which is the closest to Gargoyle [5, 11]. RECIPIE and Gargoyle share many features, including modularity in the vision routines and run-time reconfigurability among others. It's internal communication system is not particularly rich, however, and the ease of reuse is not necessarily there. The system is available as a C++ class library, with each module being a class. In order to create a new routine, each required class must be instantiated, and messages queued up for the appropriate classes. This system allows the modules to be abstracted away from the hardware drivers, however control of individual routines become more difficult.

8.2 Automatically Building Routines

One of the directions we were initially looking at with Gargoyle modules was the ability to automatically generate routines for the specific task and environment. We ended up not following that route, however some researchers have built systems which use machine learning techniques to generate special purpose routines for specific tasks. While these

systems do not switch routines based on the context, they do build routines for that context ahead of time.

8.2.1 Colorado State: ADORE

Draper, Bins, and Baek have developed the ADORE system to allow for easy creation of new vision routines for specific tasks [28]. They felt that while there were many frameworks which allowed new routines to be constructed, there were none that helped the vision routine designer pick which operators were appropriate for their new routine. Their complaint was that the construction of new routines was quite ad-hoc. The goal of their system is to provide a theoretically sound method of generating new vision routines.

In order to accomplish this goal, they needed a system which would adapt to the current situation. Their system models the object recognition problem as a Markov Decision Process. It then uses this with a set of labeled samples to learn the optimal strategies for the given recognition problem.

While learning the optimal recognition strategies beforehand has advantages in terms of accuracy, this system not intended to be used outside of the object recognition task. This is due to the cost of learning the new routines. It also requires a good training sample. Both of these make it not amenable to real time vision tasks.

8.2.2 Carnegie Mellon University: Learning Routines

Martin Martin and Moravec at Carnegie Mellon University used genetic algorithms to evolve new visual routines [69]. This system was genetic algorithm based and did not use explicit constraints to reason about how to build the constraints. It broke the vision routines up into atomic units which it would then use as the operations with which to drive the genetic algorithms. This is an interesting problem area because it it possible for this system to

develop routines which adhere to constraints on an entirely new environment. Unfortunately their experiments have shown that it is not an appropriate technique for online learning. The routines must be learned offline with a large data-set of positive and negative results.

8.3 Combined Estimation

Beyond selecting specific routines to use, some systems rely on a multiplicity of routines running at the same time to produce effective results. Figuring out how to combine these routines often leads to an interesting vision problem.

8.3.1 MIT: Lightweight Robotic Navigation

The earliest work in computer vision which provides direct information for a system embedded in the real world was in navigation for autonomous agents. These systems were required to rely on strong constraints about the world. Ian Horswill's Polly [51] system was one of the first systems to use explicit constraints. This system was described in Section 2.1.3. This system would engage many different navigational behaviors, however, it only used a single vision technique for navigation. More interesting was the fact that when it engaged in non navigational tasks, like accepting commands from people, it would use different vision techniques thus allowing task constraints to control which routine was running.

Liana Lorigo extended this work to utilize a number of different lightweight routines for the same task [64]. These routines were combined in such a way as to provide accurate obstacle avoidance. Patterns are matched to find the first obstacle, which are each returned as an array of safe distances. These arrays are then combined to determine which direction is the best. This differs from our method since it does not deal with the case where one of the routines fails because its constraints fail. In this case, if the routine hallucinates an obstacle, it will still be combined with the output of the other routines,

potentially leading to false readings.

8.3.2 MIT: Pfinder

MIT Produced a number of systems that heavily rely on explicit constraints to produce fast accurate results. In particular Christopher Wren's work on the DYNA system and the very prolific Pfinder person tracking system which that work came from [25, 24, 90, 91, 92, 89]. These systems provide very precise person tracking routines. In order to do this they rely on many strong constraints. Specifically they create models of the people they are tracking and use a number of vision techniques to follow the model. The data from these routines are then combined using condensation estimation techniques to determine the location of the person and the parameters for the model. This creates a very precise measure of where the person is how his body is oriented. This system uses constraints to improve the results of their vision processing, however rather than reasoning about the constraints, they explicitly encode the physical constraints into their condensation estimation techniques.

8.3.3 University of Massachusetts Amherst: Adaptive Systems

Zhu, Karuppiah and others have examined using software mode changes to improve continuous motion tracking [60]. They have developed a platform for building systems which modify how they operate while they are running called the University of Massachusetts Self-Adaptive Software (SAS) platform. They use multiple sensors to overcome the problems with one or the other. By having this redundancy they are able to achieve effective person tracking in their environment. Their system does not stop at sensor fusion, however. They are also interesting in determining when specific sensors are failing which can allow for the information to be aggregated more appropriately.

8.4 Vision Routine Studies

In order to build systems that reason about when to use vision techniques, it is important to have a good understanding of what techniques exist for a given set of tasks and how they operate. Fortunately some tasks in computer vision are growing quickly enough that there has been interest in studying them and defining the space of constraints that they use.

One of the most important vision routines for all of our systems is person tracking. By knowing where the user is and what he is doing, whole new modes of interaction are open to computer systems. Because of this much work has been done on tracking humans as they move about in a scene. J.K. Aggarwal from the University of Texas at Austin explores the complete space of these systems in his excellent review of the field [1]. He divides the problem space into tracking human motion, and recognizing human activity, in which the body parts of humans are tracked. While both forms of tracking use constraints, the latter uses explicit constraints about the possible orientations of the different body parts to inform the vision how to operate. Another survey of motion analysis by G.M. Gavrila from Damier-Benz breaks human motion analysis down by three-dimensional and two-dimensional, with or without models [45]. Some of the three-dimensional techniques are an attempt to get back to complete reconstruction, however, in order to achieve good results these model based techniques often end up relying on very strong constraints.

Finally, T. Moeslund gives a very complete survey of computer vision based human motion capture [71]. The level of precision required in human motion capture means that many techniques are used to further constrain the problem. Of particular interest is his list of assumptions that these systems make, or for our purposes, constraints that the system's vision routines rely on. He groups them into three main categories: movement assumptions, environment assumptions, and subject assumptions. Examples of movement assumptions include, one person in the scene, no camera motion, constant camera motion,

etc. Environment assumptions include things like static background and known camera parameters. And finally subject assumptions include things like known start position and special colored clothing.

This catalogue of assumptions is very exciting for us because it implies that it may be possible to enumerate constraints for a given task.

8.5 Reasoning and Execution Techniques

Throughout this dissertation we have stressed the need for a reasoning system to control the vision. Since our vision systems interact with the world, classical planners which are unable to deal with changes in the world (see, for example, Nilsson's seminal work on Shakey [73]) are unsuitable to our vision methodology. Fortunately much work in the field of autonomous robotics in the past decade has developed a range of more reactive reasoning architectures as discussed by Arkin and Maes in their overview books [6, 66].

8.5.1 Subsumption Networks

Early work by Brooks, which took a dramatic step away from static plans, rejected planning entirely [15, 18]. The subsumption networks proposed by Brooks have been implemented in many different systems, from insect like robots like Ghengis and Atilla [16] to more complex humanoid robots, like Cog [19]. These systems operate by directly attaching sensing to action and using input from the different sensors to control how the actuators run. In this way any vision routines written for a subsumption architecture are necessarily going to be special purpose. However they are going to be strongly tied to the specific behavior that they are designed for, and the process of rewiring the network so the same routine can be used for another behavior proves difficult. Nonetheless, vision has been used successfully to control subsumption networks [80, 51]. While this bottom up approach provides a wealth

of useful skills, it does not provide the necessary cognitive control for our general purpose system. Perhaps the most important contribution of this work to our own is the notion that powerful behaviors can be built by relying on string task constraints.

8.5.2 Action Selection

Along a similar line of research, Pattie Maes considers action selection [65] as an alternative to subsumption networks. Her action selection mechanism allows the system to have goals, and decide which behavior to use when. Rather than simply activating a behavior based on the current input, their activation is also influenced by the current goal of the system. This allows the system to have much more direct control over the task context, and therefore a good basis for accomplishing our goal of more general purpose vision. In fact, the finite state machine used in the Interactive Photomosaic project, resembles an action selection network.

Rosenblatt and Thorpe from CMU implemented another method of action selection by allowing the different low-level behaviors to vote for which higher level tasks to do, the Distributed Architecture for Mobile Navigation [78, 77]. This is really a cross between action selection and subsumption, but still lacks the high level understanding for more complex tasks.

8.5.3 Three Layer Architectures

The work that most closely resembles ours, however is the work that has been done on so called "Three Layer Architectures". Bonnasso, Firby, Gat, Kortenkamp, Miller, and Slack wrote an analysis of three tiered architectures [14]. Their system includes a set of reactive skills, a sequencer that activates skills to create a network of skills that accomplish specific tasks, and a deliberative planner. Deliberation gives partial task ordering. Sequencer gives

instantiated tasks. Skills give sensor readings and actuator commands. This one is critical.

The Classroom's skill system is directly derivative of RAPs [32], as described in Chapter 6.

8.6 Computer Vision Frameworks

A large portion of this work was the design and development of Gargoyle. Since one of the main goals of this research is to make a system that is easily reusable and applicable to more contexts, it is important to have a framework that allows the routines to be reused and support the ability to change how they are operating at run-time. When we started this project, we found no frameworks that supported this behavior. More recently though, a number of frameworks have appeared which are worth examining.

Some of the oldest computer vision frameworks are designed to simplify the creation of vision routines. These functioned by allowing the user to connect prebuilt pieces of code in a visual manner to generate novel vision routines. The following frameworks provide a "high level programming language" for visual routines, but do not provide methods of controlling routine structure once they are constructed.

8.6.1 Khoros

Khoros was one of the the first graphic user interfaces for creating vision routines [76]. This system breaks routines up into "glyphs" which are instantiated in C libraries. This allows Khoros to be used to rapidly prototype the vision routines. Once they function properly, the routines can be compiled into C code which is then used in the overall system. The advantage of this technique is the speed of the resulting system and ease of constructing new vision routines. The notion of modules in Gargoyle was heavily influenced by our

experimentation with Khoros. This visual nature of the glyphs also influenced the design of the GUI Gargoyle client. Unfortunately it was not amenable to being reconfigurable, a primary concern for Gargoyle. It also lacked the ability to run in tight control loops, being designed more for single image use, rather than real time computer vision.

8.6.2 CVIPtools

CVIPtools, which stands for Computer Vision Image Processing tools, is another library for constructing new vision routines much like Khoros [88]. Developed at the University of Southern Illinois, CVIPtools differs from Khoros in that it comes with a number of utilities which allow direct manipulation of images using the available vision operators. This system provides an excellent learning environment, and a good environment for using trial and error methodology to creating new routines. It also allows the constructed routines to be exported for use by other programs.

8.6.3 IUE

Image Understanding Environment was developed by ARPA in order to provide a common development environment for image understanding work [62]. It is mostly a cross-platform library of vision operators, however it also incorporates a number of interesting reasoning operators as well, mostly focused on three dimensional reconstruction.

Chapter 9

Conclusions

Many vision researchers have built successful systems by relying on constraints in the world in order to narrow the space in which their routines need to operate. These special purpose routines are more efficient and more reliable in the context in which they operate by sacrificing generality. This dissertation argues that operating within a given context does not necessarily need to imply a loss of generality. In order to utilize the context in the first place, it must be understood. We believe that by leveraging that understanding, more general purpose vision can be obtained. If a system understands when to use which special purpose routine, it is a general purpose system.

With this goal in mind, this dissertation presents our work on three different systems which successfully accomplish a range of tasks in the physical world: Chip, the Intelligent Classroom and the Interactive Image Mosaic. These systems owe much of their success to the vision routines which they utilize. These routines have all been tailored to solve their specific tasks in the environments in which the tasks are to be accomplished. However, we are able to use them in this larger range of tasks and environments by having the system understand the constraints on which the routines rely and then choose

appropriate routines as required for the task the system is attempting to solve.

In addition, we believe that understanding the constraints and reasoning about them provides a method of utilizing existing work in computer vision to build more robust vision systems for specific tasks. As computer vision is more and more widely used in real-world tasks it becomes important to be able to use the vision systems built for specific tasks more broadly. Many researchers now take advantage of the structure of the world around them to provide simplifying assumptions for their vision routines. These constraints allow their systems to function and we feel provide hooks to expanding their usefulness to less well constrained environments.

By following the Methodology given in Section 1.5 we have built a number of systems which are able to successfully use computer vision to solve specific tasks. Understanding the constraints on the vision routines has allowed us to reuse vision routines in novel environments with a small adaption time. Starting with Chip, then using those routines in the Intelligent Classroom we have shown this methodology to be effective. Moving them to the Interactive Image Mosaic we have maintained the same base of routines in widely varying environments showing generality. We are encouraged that this type of system can be used to create quite generally useful vision systems.

9.1 Building Useful Systems Incrementally

One of the most interesting features of this line of thinking, and indeed our implementations, is that when developing more robust systems, they continue to be functional at each stage of development. By understanding the constraints of each added vision routine, the researcher is able to expand the world in which the system as a whole can operate. Since those routines are built to solve specific problems, each additional problem or environment is tackled, one routine at a time. This allows a system that reasons about when to use which routine to

expand the set of tasks or environments in which it can operate.

The improvements provided by reasoning about how to use the routines do not allow our systems to cover the entire constraints space, even in their respective controlled domains. That is, we have not achieved a complete general purpose solution, but we have made progress in that direction and built useful systems. This research does not explain how to do general purpose computer vision. What it does is give a path that we can follow toward general purpose vision while continuously providing a useful set of tools.

In order to be a useful tool our vision system must take advantage of the structure that the world gives us. Others have argued for the need to study the environment in which our systems will operate; we argue for the need to embed those routines in a system which understands that environment. We need to pay attention to that structure so that we can continue to use the work we already have. Chapter 8 shows that many researchers rely on the constraints inherent in the world. These routines are effective in their specific domains, however they will fail when the context changes, and are difficult to expand to other domains. By making these constraints explicit and reasoning about when to use the different special purpose routines, we are able to add more routines to our system as required by an ever-expanding task and environment set.

9.2 The Future of This Work

The work presented in this dissertation has given us a number of encouraging results. However, the ideas of generality presented here require ongoing refinement and experimentation. This section examines some future directions and open questions, for both the vision techniques presented here in general as well as the Classroom in particular.

9.2.1 More Vision Routines

The obvious extension is to expand the set of vision routines that the Classroom uses, to accomplish more tasks or cover a wider range of environments. Since the classroom is an interactive environment, we are particularly interested in routines which would enhance it's interactive nature. Most human visual interaction is with faces, so it would be nice if the classroom could interact with faces.

Eye and face tracking is currently quite an active field. Combining those techniques into the Classroom would allow a number of interesting filming techniques. For one, it would allow the Classroom to follow faces when it was doing a close up, rather than simply waiting at the top of the body. Following the theory of extracting only the needed information for a specific task, doing a head shot only requires knowing where the face is, not where the entire body is. Face recognition would also be useful for identifying users, and allowing the system to maintain preference profiles and have more information about what that specific user might need. When used in conjunction with schedules and connected class material, this could become quite powerful.

9.2.2 Constraint Representation

One of the key aspects of our system is that the constraints that the system uses need to be explicitly represented in order to be reasoned about. Unfortunately, it can be difficult to determine in advance the range of constraints that a given routine uses. This leads to an underspecification of the constraints on which a system relies.

Some subfields of computer vision are rich with systems providing a fertile resource for studying the range of possible constraints. This could potentially lead to a better understanding of the types of constraints that vision routines rely on. This, in turn, could help vision researchers avoid the problems of underspecifying the constraints that

the routines rely on. A number of groups have made some analysis along these lines as discussed in Chapter 8. However, there is currently no uniform method of describing those constraints.

One possible solution to this problem would be to define a language for expressing the constraints on a vision routine, independent of the explicit internal representation. This language could then be used to describe when and where the particular vision routines should be used. The constraint descriptions would then be distributed with the vision routines. Integration with a complete system would simply involve selecting and implementing the constraints useful to the given task. It would also help define the constraints which must be enforced.

It is worth noting that while this constraint description language could be the explicit representations, different systems will have different methods of representing their constraints. Therefore it is important to keep the description at a high level if the routines are to be reused.

Once the range of constraints is well understood, it would be interesting to study how to obtain information about that state. In the Classroom all of the constraints that we use are able to be extracted directly from the routines themselves or the internal state of the Classroom. There are many constraints however, whose state is nontrivial to ascertain. Knowing the range of useful constraints may lead to discovering the set of routines which contain information about those constraints. This would be extremely helpful in designing new systems.

9.3 Vision and the InfoLab

The mantra of Northwestern University's Intelligent Information Laboratory (the InfoLab), where this work was done, is "frictionless access to information". Mainstream InfoLab

projects like Watson and XLibris [20, 23] use the context in which people are operating to give them the information they need when they need it without them having to ask for it. These systems do this by operating in an information environment, be it web pages that a person is surfing, a book they are checking out from the library, or a paper they are writing.

The Classroom uses the context of a lecture to help the lecturer without having to be explicitly asked to do so. Instead of existing in electronic form, it exists in the real world, but the notion of allowing computers to take actions for you still exists. In an electronic task, such as writing a paper or surfing the web, it is not easy to get appropriate information for the user without understanding what it is that that user is doing. However by using the context in which the user is operating the InfoLab has shown that computer systems are able to provide useful information with a minimal amount of "smarts".

We believe that this is even more important in the world of computer vision. The InfoLab has shown that computers can be proactively useful in everyday tasks for which people use computers. XLibris begins moving that proactivity away from a computer environment and into the real world. Computer vision techniques are a clear next step in that direction, allowing the computer to be proactive in determining the needs of the user given the context in which he is operating.

As computer and camera systems become more ubiquitous and people become more connected, this line of research should prove quite exciting. Combining multiple sources of information will provide more information about the context the user is living in, which makes the job of the computer vision easier, and in turn allows us to provide more timely and useful information to the user.

9.4 Final Thoughts

Computer vision is being used more and more in the real world, from surgical assistants to automated vehicles driving on our highways to face recognition systems monitoring our security. It is quickly becoming a part of the mainstream consciousness. As vision researchers, we need to be honest in describing the capabilities of our system. When a system which is described as a solution for a given problem fails because one of its required constraints fails, the response is to say that the system is useless rather than that it was applied inappropriately. Until we reach the point when computer vision based systems know that they are failing and can report that rather than simply failing, it is incumbent on vision researchers to understand which constraints our systems rely on and advertise them.

Computer vision is a useful tool. However, it is a picky tool that defies generality. By carefully studying how to use that tool in different situations, we hope to make the tool more useful by both making it more robust for the task for which it was built, and easier to reuse.

Humans use vision to solve almost any task. As the set of vision routines that are available to the community expands, more and more tasks will be solvable using computer vision. In order to be truly successful, the people implementing these systems will need to understand the limitations of the computer vision they are using. It is our hope that this ever expanding set of vision routines will be easily applied to these new situations. By utilizing them in systems that have an understanding of the world, knowing where and when different routines should be used, we believe that these routines will more easily accomplish interesting tasks than were possible before.

We see a bright future for computer vision in solving tasks that were not considered before. Before this is possible, we need a set of vision tools which can be easily

and robustly applied to these problems. It is our hope that this dissertation presents a direction to follow in providing those tools.

Bibliography

[1] J. K. Aggarwal and Q. Cai, *Human motion analysis: A review*, Computer Vision and Image Understanding: CVIU **73** (1999), no. 3, 428–440.

[2] Phillip Agre and David Chapman, *Pengi: An implementation of a theory of activity*, Proceedings of the National Conference on Artificial Intelligence (AAAI-87), American Association for Artificial Intelligence, AAAI Press / The MIT Press, July 1987.

[3] Yiannis Aloimonos, *Purposive and qualitative active vision*, Proceedings of the 10th International Conference on Pattern Recognition (Atlantic City, NJ), June 1990, pp. 346–360.

[4] ———, *What i have learned*, CVGIP: Image Understanding **60** (1994), no. 1, 74–85.

[5] Tom Arbuckle and Michael Beetz, *Recipe - a system for building extensible, run-time configurable, image processing systems*, CVPR, 1998.

[6] Ronald C. Arkin, *Behavior-based robotics*, The MIT Press, 1998.

[7] Ruzena Bajcsy, *Active perception*, Proceedings of the IEEE **76** (1988), no. 8, 996–1004.

[8] Ruzena Bajcsy and K. Goldberg, *Active touch and robot perception*, Cognition and Brain Theory **7** (1984), no. 2, 199–216.

[9] Dana H. Ballard, *Generalizing the Hough transform to detect arbitrary shapes*, Pattern Analysis **13** (1981), 111–122.

[10] ———, *Animate vision*, Artificial Intelligence **48** (1991), 57–86.

[11] Michael Beetz, Tom Arbuckle, Armin B. Cremers, and M. Mann, *Transparent, flexible, and resource-adaptive image processing for autonomous service robots*, European Conference on Artificial Intelligence, 1998, pp. 632–636.

[12] Aaron Bobick, Stephen Intille, Jim Davis, Freedom Baird, Claudio Pinhanez, Lee Campbell, Yuri Ivanov, Arjan Schutte, and Andrew Wilson, *The kidsroom: A perceptually-based interactive and immersive story environment*, Presence: Teleoperators and Virtual Environments **8** (1999), no. 4, 367–391.

[13] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack, *Experiences with an architecture for intelligent, reactive agents*, Journal of Experimental & Theoretical Artificial Intelligence **9** (1997), no. 2/3, 237–256.

[14] R. Peter Bonasso, R. James Firby, Erann Gatt, David Kortenkamp, David P. Miller, and Mark G. Slack, *Experiences with an architecture for intelligent reactive agents*, Proc. of the Int. Joint Conf. on Artificial Intelligence, 1995.

[15] Rodney A. Brooks, *A hardware retargetable distributed layered architecture for mobile robot control*, Proceedings of the IEEE conference on Robotics and Automation, IEEE Press, 1987, pp. 106–110.

[16] ———, *Robots that walk*, Proceedings of the IEEE International Conference on Robotics and Automation, April 1989.

[17] ———, *Elephants don't play chess*, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back (P. Maes, ed.), MIT Press, 1990, pp. 3–15.

[18] ———, *Intelligence without reason*, A.I. Memo 1293, MIT AI Laboratory, April 1991.

[19] ———, *Building brains for bodies*, Autonomous Robots **1** (1994), no. 1, 7–25.

[20] Jay Budzik, Kristian Hammond, and Larry Birnbaum, *Information access in context*, Knowledge Based Systems **14** (2001), no. 1–2, 37–53.

[21] David Chapman, *Vision, instruction, and action*, MIT Press, 1991.

[22] Silvia Coradeschi and Alessandro Saffiotti, *Anchoring symbolic object descriptions to sensor data. problem statement*, Linküping electronic articles in computer and information science **4** (1999), no. 9, 1–5.

[23] Andrew Crossen, Jay Budzik, Mason Warner, Larry Birnbaum, and Kristian Hammond, *XLibris: An automated library research assistant*, Proceedings of the Conference on Intelligent User Interfaces (IUI-2001), ACM Press, 2001.

[24] Trever Darrell, Pattie Maes, Bruce Blumberg, and Alexander Pentland, *A novel environment for situated vision and behavior*, Workshop On Visual Behaviors: Computer Vision and Pattern Recognition (1994), 68–72.

[25] Trever Darrell and Alexander Pentland, *Space-time gestures*, Computer Vision and Pattern Recognition (1993).

[26] P. Davidson, *Toward a general solution to the symbol grounding problem: Combining machine learning and computer vision*, AAAI Fall Symposium Series, Machine Learning in Computer Vision: What, Why, and How?, AAAI Press / The MIT Press, 1993, pp. 191–202.

[27] James W. Davis and Aaron F. Bobick, *The representation and recognition of action using temporal templates*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR '97), The Institute of Electrical and Electronics Engineers, 1997.

[28] Bruce A. Draper, Jose Bins, and Kyungim Baek, *ADORE: Adaptive object recognition*, ICVS, 1999, pp. 522–537.

[29] Adam Finkelstein and Marisa Range, *Image mosaics*, Proceedings of the EP 98 and RIDT 98 Conferences, Electronic Publishing, Artistic Imaging and Digital Typography, March 1998.

[30] R. James Firby, *Adaptive execution in complex dynamic worlds*, Ph.D. thesis, Yale, 1989.

[31] ———, *The crl manual*, Animate Agent Project Working Note 3, University of Chicago, February 1995.

[32] ———, *The rap language manual*, Animate Agent Project Working Note 6, University of Chicago, March 1995.

[33] R. James Firby, Roger E. Kahn, P. N. Prokopowicz, and Michael J. Swain, *Collecting trash: A test of purposive vision*, Workshop on Vision for Robots, August 1995.

[34] R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain, *An architecture for vision and action*, International Joint Conference on Artificial Intelligence, August 1995.

[35] ———, *Collecting trash: A test of purposive vision*, Workshop on Vision for Robots, August 1995.

[36] R. James Firby, Peter N. Prokopowicz, and Michael Swain, *Ai-based mobile robots*, AI-based Mobile Robots (David Kortenkamp R. Peter Bonasso and Robin Murphy, eds.), MIT/AAAI Press, 1997.

[37] R. James Firby, Peter N. Prokopowicz, Michael J. Swain, Roger E. Kahn, and David Franklin, *Programming chip for the ijcai-95 robot competition*, AI Magazine **17** (1996), no. 1, 71–81.

[38] Joshua D. Flachsbart, *Gargoyle: Vision in the intelligent classroom*, Master's thesis, University of Chicago, 1997.

[39] Joshua D. Flachsbart, David Franklin, and Kris J. Hammond, *Improving human computer interaction in a classroom environment using computer vision*, Proceedings of the Conference on Intelligent User Interfaces (IUI-2000), 2000.

[40] David Franklin, *Cooperating with people: The intelligent classroom*, Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), American Association for Artificial Intelligence, AAAI Press / The MIT Press, 1998.

[41] ———, *The intelligent classroom: Competent assistance in the physical world*, Ph.D. thesis, Northwestern University, 2001.

[42] David Franklin, Shannon Bradshaw, and Kristian J. Hammond, *Beyond "next slide, please": The use of content and speech in multi-modal control*, Working Notes of the AAAI-99 Workshop on Intelligent Information Systems, American Association for Artificial Intelligence, AAAI Press / The MIT Press, 1999.

[43] David Franklin, Roger E. Kahn, Joshua D. Flachsbart, Michael J. Swain, and R. James Firby, *Happy patrons make better tippers: Creating a robot waiter using perseus and the animate agent architecture*, International Conference on Automatic Face and Gesture Recognition, 1996.

[44] Erann Gat, *Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots*, Proceedings of the National Conference on Artificial Intelligence (AAAI-92), American Association for Artificial Intelligence, AAAI Press / The MIT Press, 1992.

[45] D. M. Gavrila, *The visual analysis of human movement: A survey*, Computer Vision and Image Understanding: CVIU **73** (1999), no. 1, 82–98.

[46] James Jerome Gibson, *Perception of the visual world*, Houghton Mifflin, 1950.

[47] Mohinder S. Grewal and Angus P. Andrews, *Kalman filtering theory and practice*, Prentice Hall, 1993.

[48] A. Hanson and E. Riseman, *Visions: A computer system for interpreting scenes*, Computer Vision Systems (A. Hanson and E. Riseman, eds.), Academic Press, New York, NY, 1978, pp. 303–334.

[49] A. Hausner, *Simulating decorative mosaics*, Proceedings of ACM SIGGRAPH 2001 (New York, NY) (E. Fiume, ed.), ACM, ACM Press / ACM SIGGRAPH, 2001, pp. 573–580.

[50] David Hinkle, David Kortenkamp, and David Miller, *The 1995 robot competition and exhibition*, AI Magazine **17** (1996), no. 1, 38–45.

[51] Ian Horswill, *Polly: A vision-based artificial agent*, Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), American Association for Artificial Intelligence, AAAI Press / The MIT Press, July 11–15 1993.

[52] ———, *Specialization of perceptual processes*, Ph.D. thesis, Massachusetts Institute of Technology, 1993.

[53] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge, *Comparing images using the hausdorff distance*, IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993), no. 9, 850–863.

[54] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin, *Fast multiresolution image querying*, Computer Graphics **29** (1995), 277–286.

[55] Roger E. Kahn, *Perseus: An extensible vision system for human-machine interaction.*, Ph.D. thesis, The University of Chicago, August 1996.

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

[57] Roger E. Kahn and Michael J. Swain, *Understanding people pointing: The perseus system*, International Symposium on Computer Vision (1995), 569–574.

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

[59] Roger E. Kahn, Michael J. Swain, P. N. Prokopowicz, and R. James Firby, *Gesture recognition using the perseus architecture*, Computer Vision and Pattern Recognition, June 1996.

[60] Deepak Karuppiah, Patrick Deegan, Elizeth Araujo, Yunlei Yang, Gary Holness, Zhigang Zhu, Barbara Lerner, Roderic Grupen, and Edward Riseman, *Software mode changes for continuous motion tracking*, Proceedings of the International Workshop on Self Adaptive Software, 2000.

[61] J. Kim and F. Pellacini, *Jigsaw image mosaics*, Proceedings of ACM SIGGRAPH 2002 (New York, NY), ACM, ACM Press / ACM SIGGRAPH, 2002.

[62] Charles Kohl and Joe Mundy, *The development of the image understanding environment*, Proceedings of Computer Science and Pattern Recognition (CVPR94), IEEE Computer Society Press, 1994.

[63] David Kortenkamp, Peter Bonasso, and Robin Murphy, *Artificial intelligence and mobile robots*, MIT Press, 1998.

[64] L. Lorigo, R. Brooks, and W. Grimson, *Visuallyguided obstacle avoidance in unstructured environments*, Proceedings on Intelligent Robots and Systems, 1997.

[65] Pattie Maes, *Situated agents can have goals*, Robotics and Autonomous Systems **6** (1990), 49–70.

[66] Pattie Maes (ed.), *Designing autonomous agents*, The MIT Press, 1991.

[67] David Marr, *Vision*, W. H. Freeman and Company, San Francisco California, 1982.

[68] David Marr and H. K. Nishihara, *Representation and recognition of the spatial organization of three dimensional shapes*, Proceedings of the Royal Society London **B** (1978), no. 100, 269–294.

[69] Martin C. Martin, *The simulated evolution of robot perception*, Ph.D. thesis, Carnegie Mellon University, 2001.

[70] Maya M. Mataric, *Behavior-based control: Main properties and implications*, Proceedings of the Workshop on Intelligent Control Systems, May 1992.

[71] T. Moeslund and E. Granum, *A survey of computer vision-based human motion capture*, Computer Vision and Image Understanding: CVIU **75** (2001).

[72] Vishvjit Nalwa, *A guided tour of computer vision*, Addison Wesley Publishing Company, 1993.

[73] N. Nilsson, *Shakey the robot*, Tech. report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1984.

[74] Claudio Pinhanez and Aaron F. Bobick, *Intelligent studios: Using computer vision to control tv cameras*, Proceedings of the Workshop on Entertainment and AI/Alife, 1995.

[75] Peter N. Prokopowicz, Michael J. Swain, and Roger E. Kahn, *Task and environment-sensitive tracking*, Workshop On Visual Behaviors: Computer Vision and Pattern Recognition (1994), 73–78.

[76] J. Rasure and S. Kubica, *The KHOROS application development environment*, Experimental Environments for Computer Vision (Ney Jersey), World Scientific, 1994.

[77] Julio A. Rosenblatt, *Damn: A distributed architecture for mobile navigation*, Ph.D. thesis, The Robotics Institute, CMU, 1995.

[78] Julio A. Rosenblatt and Charles E. Thorpe, *Combining multiple goals in a behavior-based architecture*, proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems (Los Alamitos, CA), vol. 1, IEEE Press, August 1995, Human Robot Interaction and Cooperative Robots, pp. 136–141.

[79] Garbis Salgian, *Tactical driving using visual routines*, Ph.D. thesis, University of Rochester, 1998.

[80] K. Sarachick, *Characterizing and indoor environment with a mobile robot and uncalibrated stereo*, Proceedings of the IEEE International Conference on Robotics and Automation (Scottsdale, AZ), 1989.

[81] Robert S. Silvers, *United states patent number 6,137,498*, October 1997.

[82] Robert S. Silvers and M. Hawley, *Photomosaics*, Henry Holt and Co., 1997.

[83] Thomas M. Strat, *Natural object recognition*, Springer-Verlag, 1992.

[84] Michael J. Swain, *Color indexing*, Ph.D. thesis, University of Rochester, 1990.

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

[86] Michael J. Tarr and Michael J. Black, *A computational and evolutionary perspective on the role of representation in vision*, CVGIP: Image Understanding **60** (1994), no. 1, 65–73.

[87] Shimon Ullman, *Visual routines*, Cognition **18** (1984), 97–159.

[88] S. Umbaugh, *Computer vision and image processing: A practical approach using CVIPtools*, Prentice Hall, New Jersey, 1998.

[89] Christopher Wren, *Understanding expressive action*, Ph.D. thesis, Massachusetts Institute of Technology, 2000.

[90] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alexander Pentland, *Pfinder: Real-time tracking of the human body*, Media Laboratory Perceptual Computing Section 353, Massachusetts Institute of Technology, 1995.

[91] ———, *Pfinder: Real-time tracking of the human body*, IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997), no. 7, 780–785.

[92] Christopher Wren and Alex P. Pentland, *Understanding purposeful human vision*, Proceedings of the IEEE International Conference on Computer Vision, 1999.

[93] Shujun Zhang, Geoff D. Sullivan, and Keith D. Baker, *The automatic construction of a view-independent relational model for 3-d object recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993), no. 6, 531–543.