THE UNIVERSITY OF CHICAGO

# GARGOYLE: VISION IN THE INTELLIGENT CLASSROOM.

A DISSERTATION SUBMITTED TO

THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES

IN CANDIDACY FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

JOSHUA D. FLACHSBART

CHICAGO, ILLINOIS

DRAFT OF November 20, 1997

## ACKNOWLEDGMENTS

I would like to thank Michael Swain for all of his support and ideas. Thanks also go to Peter Prokopowicz and Roger Kahn for all of their help and guidance early on. Additional thanks should go to Peter for writing Gargoyle and allowing me to take it where I think it needs to go. Special thanks go to Jim Firby for convincing me to do things right the first time, and if you later find that you did it wrong, fix it! Dave deserves some thanks for being the first person to read this paper. Finally thanks to Alain, Charlie, Edwin, Kris, Shannon, and Vassilis for making the lab a fun livable place to work.

# TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . . . ii

LIST OF TABLES . . . . . v

LIST OF ILLUSTRATIONS . . . . . vi

ABSTRACT . . . . . vii

CHAPTER

1. INTRODUCTION . . . . . 1

2. GARGOYLE . . . . . 3
   - 2.1 Motivation . . . . . 3
     - 2.1.1 The Client . . . . . 6
     - 2.1.2 Using Context to Improve Vision . . . . . 6
     - 2.1.3 Breaking Up the Task . . . . . 8
   - 2.2 Design . . . . . 9
     - 2.2.1 Overview . . . . . 9
     - 2.2.2 Server . . . . . 10
     - 2.2.3 Pipeline . . . . . 11
     - 2.2.4 Modules . . . . . 12
   - 2.3 Gargoyle in use . . . . . 14

3. PORTING PERSEUS . . . . . 16
   - 3.1 Perseus . . . . . 16
     - 3.1.1 How it works . . . . . 17
     - 3.1.2 The Pointing Task . . . . . 17
   - 3.2 Gargoyle modules for the pointing task . . . . . 20
     - 3.2.1 Input . . . . . 21
     - 3.2.2 Sobel and Floor Finding Module . . . . . 22
     - 3.2.3 Threshold . . . . . 22
     - 3.2.4 Background . . . . . 23
     - 3.2.5 Segmentation Map . . . . . 24
     - 3.2.6 Person Tracker . . . . . 24
   - 3.3 Comparison of Gargoyle to Perseus . . . . . 25

4. INTELLIGENT CLASSROOM . . . . . 27
4.1 Classroom Design . . . . . 27
4.2 Classroom Tasks . . . . . 28
4.3 Applicability . . . . . 29

5. CURRENT IMPLEMENTATION . . . . . 31
5.1 Controlling Gargoyle Automatically . . . . . 31

6. FUTURE DIRECTIONS . . . . . 33
6.1 Intelligent Classroom . . . . . 33
6.2 Gargoyle . . . . . 34

7. SUMMARY AND CONCLUSIONS . . . . . 36

REFERENCES . . . . . 37

# LIST OF TABLES


| Table | | Page |
|---|---|---|
| 1 | The main motivational elements behind Gargoyle. | 3 |
| 2 | How Gargoyle gets its context. | 7 |
| 3 | Component breakdown of Gargoyle | 10 |
| 4 | Module Communication Methods | 13 |
| 5 | Comparison of the different units in the system. | 21 |
| 6 | Advantages and disadvantages of Gargoyle and Perseus | 25 |
| 7 | Some possible tasks for the intelligent classroom. | 28 |
| 8 | Implemented Framings | 32 |

# LIST OF ILLUSTRATIONS


| Figure | | Page |
|---|---|---|
| 1 | A visual routine with interchangeable parts | 5 |
| 2 | Example Pipeline | 6 |
| 3 | General Architecture | 9 |
| 4 | Detail of client control of Gargoyle pipelines | 11 |
| 5 | Diagram of the pointing task implemented in Perseus. | 18 |
| 6 | A Gargoyle pipeline for tracking people | 21 |
| 7 | The outputs of from left to right: the floor, threshold, and background modules. | 22 |

# ABSTRACT

This paper looks at some technologies that were developed at the University of Chicago to see how well they will work as the basis for the vision aspects of our new project, the intelligent classroom. The paper starts with a look at Gargoyle, our real-time active vision framework and examines the features that make it a valuable resource to vision programers. It then looks at how Gargoyle has been tested and used so far. This experiment involved porting the Perseus real-time architecture's pointing task. Finally, the paper examines the intelligent classroom and shows the utility of Gargoyle by building on the results of the Perseus experiment. This paper also argues that for vision to be effective in the long term, it must be tightly coupled with a higher level planning or execution system.

# CHAPTER 1

# INTRODUCTION

This paper describes the research that is currently going on in the *intelligent classroom* project at the University of Chicago. The intelligent classroom is a project that will allow speakers to control multimedia elements of a classroom through natural speech and gesture. In order to do this, the room needs to be able to sense what is happening within it. This paper focuses specifically on the visual sensing aspects of the intelligent classroom, and the infrastructure that we have built to allow the classroom to "see".

Some work has been done recently[1, 3, 5, 7, 11] to allow computers to be visually aware of their surroundings. These visual tasks, however, can often be limited in scope and application by their lack of higher level knowledge and their inherent inflexibility. An example of this problem could be as simple as a program running when it is not needed, using up valuable vision processing time. If there was some way for the visual system to know that it was no longer needed, it could stop running and free up the resources that it was using. For example, the classroom's visual system might have a visual routine that looks for a specific gesture to indicate that the next slide should be shown. However, if the slide show is not being shown, then the system does not need to even bother looking for that gesture. Thus, because the system knows that it is not presenting a slide show, it can disable the part of the visual system that looks for the next slide gesture.

As well as providing better utilization of resources, this notion of smart visual systems can be extended to actually improve the performance of the system. For instance, one visual routine may be effective for a stationary platform, while another may be more effective on a moving platform. The intelligent classroom will know then it is moving its camera; and by using this information, the visual system

can decide which routine to use. This will produce more accurate results than if the visual system were unable to change how it operates.

The University of Chicago Artificial Intelligence Lab has long had the goal of realizing better computer vision through a combination of low level vision algorithms and higher level planning systems, such as RAPs on our robot. To this end we have created a general visual system that is run-time configurable by some remote agent, such as a planner. Gargoyle is new and the intelligent classroom will be the first system in which it is incorporated. Gargoyle is an important foundation for the intelligent classroom, so this paper begins with a look at Gargoyle, what it is and how it works, in order to provide a background of how we think about vision problems. The initial base of visual routines for the intelligent classroom are taken from Perseus[5], and are described in Chapter 3. The intelligent classroom is described in detail in Chapter 4.

# CHAPTER 2

# GARGOYLE

Gargoyle is a system developed at the University of Chicago to provide a unified structure in which to perform active vision. The Gargoyle framework has been developed with the intention of providing a system that will operate well under many different operating conditions. In order to understand why Gargoyle is helpful in our experiments, it is important to understand the motivation behind Gargoyle's design.

## 2.1 Motivation

The motivation behind Gargoyle is clear and based on our vision needs that were not currently met by other systems. Because Gargoyle runs in real-time, it is constrained by all of the speed requirements that other real-time vision systems are. We also have additional goals for the design of the system, which are enumerated in Table 1. This section looks at the reasoning behind each of these constraints.

The main reason for creating Gargoyle was to have a system that could be reconfigured dynamically to specific contexts by some client. This is crucial in the intelligent classroom project where many different visual contexts will be encountered. Typically, for a visual routine to work, it must be very specialized and constrained to specific environments.

| Design goals for Gargoyle: |
|---|
| To enable the dynamic creation and reconfiguration of visual routines specialized to context. |
| To enable code reuse and reduce debugging time. |
| To foster sharing between researchers. |

Table 1: The main motivational elements behind Gargoyle.

Visual routines often end up being constrained in this way because of two reasons. First, the visual routines are often written as monolithic programs which make them very hard to dynamically reconfigure at runtime. By monolithic, I mean a large single program that operates in a unchangeable linear manner. This is as opposed to Gargoyle, which can have the operating units reordered at run-time. The second reason is that the parameters which are configurable often depend on the environment and need to be tuned very carefully by the programmer ahead of time. A more general system would be able to determine the requirements for different environments and specialize itself as necessary, during runtime, without additional help from the programmer.

In order for the system to be able to tune itself to the specific demands of the current environment, it must understand what the environment is. The paradigm we use to allow the system to understand how to tune itself is that of exploiting *context* [10, 9]. By context we mean the situation in which the higher level system exists. For instance, in the intelligent classroom, the lecturer might be speaking from the lectern, or showing a video. These different contexts can provide a lot of information to the visual system on how to tune and reconfigure itself. To be able to exploit the context of the situation, the standard model of specialized, monolithic vision programming can not be used. Instead, Gargoyle employs a more general notion of what a visual routine is.

The other two design goals are based in a notion of how to write code. Monolithic visual routines are generally large and difficult to test and debug by their often non-modular nature. When writing large visual routines, elements of the code can often be embedded deeply in the code making them hard to reuse. Thus, large pieces of code can be written over and over again, since similar visual techniques are used in many different visual routines. In order to encourage code reuse and ease testing of new visual routines, the visual routines in Gargoyle are be broken up into logical elements, called modules, which can then be reused. The combination of these modules into a visual routine is called a pipeline. An example of how to break up a

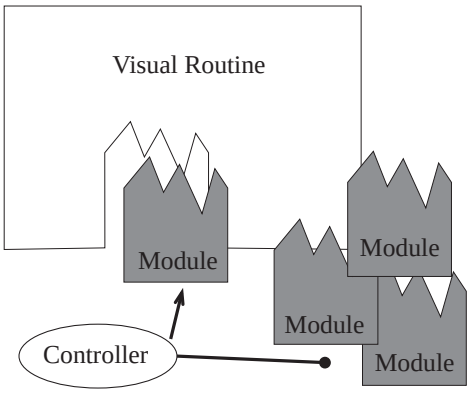


Figure 1: A visual routine with interchangeable parts

visual routine is seen in Figure 1. The visual routine is designed such that modules can be removed and replaced with other modules.

By breaking up the visual routine into basic elements, which we call modules, not only do we encourage code reuse, but we also provide a simple mechanism for the sharing of algorithms between researchers. The modules are written in machine-independent Java, which means that it is trivial for one researcher to incorporate another's modules.

The modules provided by Gargoyle allow a visual routine, which we will refer to as a pipeline, to be dynamically modified in non trivial ways. The operation of the elements 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. For example, Figure 2 shows an example of a pipeline that tracks some object using color or edges. Gargoyle provides a method to change which module, the edge tracker or the color tracker, will be used. This means that if one of the methods begins to fail, the inadequate module can be swapped out and replaced by the other one. Gargoyle also allows the parameters of the different modules to be changed during operation. This would allow the higher level system to fine tune the functioning of the modules to specific environments.

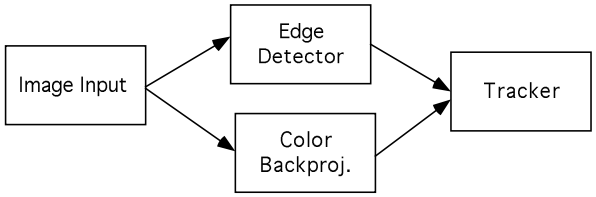


Figure 2: Example Pipeline

### *2.1.1 The Client*

The astute reader will have noticed by now that Gargoyle can not run on its own. It needs a "higher level system" to provide all of the knowledge that will make it powerful. This higher level system will be referred to as the client. The client can be a person setting up a single pipeline to test. The client could also be a reactive execution system which directly controls the system and reads information from Gargoyle and other sensors to gain information about the world. In any event, the client determines the behavior of the visual routines by deciding which modules to use, as seen in Figure 1.

If the client is a reactive execution system, as in the intelligent classroom, then it can use its knowledge about what is happening in the world to build better pipelines. In this case, the client may have access to information that Gargoyle does not have, for instance from other sensors, and can use that to build the pipelines. We refer to this type of knowledge about the world as *context* for the scene.

### *2.1.2 Using Context to Improve Vision*

The reason that Gargoyle allows the pipelines to be dynamically modified is so that they can take advantage of the client's context. Typically, the client will have some knowledge of the world around it, and can use that information to adjust the pipelines so that they can operate better in new contexts. There are two main types of context which the client passes on to Gargoyle: *task context* and *environmental context*. There is also *image context*, but that is normally internal to Gargoyle

| Context | Source |
|---|---|
| task context | client |
| environmental context | client |
| image context | other modules |

Table 2: How Gargoyle gets its context.

pipelines. Task context is information that the client knows about the task at hand which can be used to make the vision problem easier. For instance, if the intelligent classroom is showing a slide show, it can direct the pipeline not to look at the screen, since that will confuse our current pipelines. In the future, when we have more variety in our modules, it could replace the modules that are confused by the screen with one that is not. For instance, it could replace an edge detector with a motion detecting module. This notion of task dependence of vision algorithms is the key for active vision and one of the main reasons for the dynamic reconfiguration design goal.

Environmental context is a bit more subtle, and more interesting in that it tightly couples the client and Gargoyle. The reason that many visual routines need to be constrained and optimized so much is for environmental reasons. Visual tasks are very difficult and will perform much better if the environment can be constrained and the visual routine optimized for that specific environment [10]. In the client's world, state is modified by the information from Gargoyle, among other sources. The client could conceivably use this information to modify the pipeline for better performance. This is best shown by an example. One possible Gargoyle pipeline tracks a person in a scene using motion and color cues. The client can use its knowledge about the world to combine the different cues in the most advantageous way. For example, if the speaker is standing in front to the lectern and speaking, motion cues are not the best ones to use. The client could then reconfigure the pipeline so that the motion cues are not used. On the other hand, if a slide show is being presented, the lights will be dimmed and color cues will be less effective. The client could then use this information to change the pipeline configuration.

Environmental context also allows the client to modify the parameters of the pipeline. For example, if the lights are dimmed, the threshold on a sobel edge detector will need to be changed. In general these changes are simple, however, since Gargoyle allows the client to make such major changes in the pipeline, all changes are integrated much better than in a monolithic system.

The other type of context is *image context*. Image context is information based on the results of previous computation. In general the image context of a scene is held internally to a pipeline, but the client will build the pipeline so that image context will be used. For example, if you are searching for a specific object in a scene, some color analysis can be done to tell you where it might be, and then you only need to look in those locations. The pipeline shown in Figure 2 could use image context by letting the tracker define some reduced region in which a more computationally expensive visual task could operate.

### *2.1.3 Breaking Up the Task*

In order to speed up visual routine development and encourage code reuse as well as sharing between researchers, Gargoyle provides a method for breaking up the visual routines into elementary units. These units are called modules. An example of a pipeline broken up in this way can be seen in Figure 2. In this figure a simple visual tracking routine is implemented as a pipeline. By utilizing standard vision components like sobel edge detectors and color histogram back-projectors as cues for the tracker, very little additional coding needs to be done. Also, one would only need to worry about debugging the tracker if the other components have already been carefully tested. Breaking up the task in this manner also allows individual components to be tested more easily. As the developer of a component with some new algorithm in it, you can make yourself the client and try the component out in many different situations. Additionally when you are satisfied with the performance of the component, it can be easily distributed for use in other researchers' pipelines.

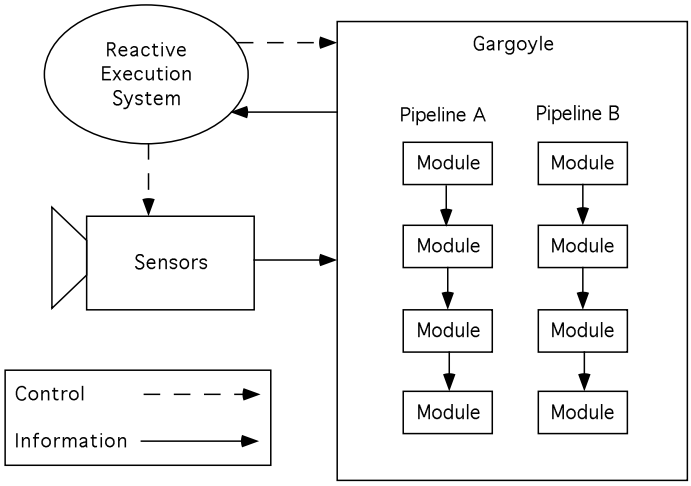


Figure 3: General Architecture

## 2.2 Design

Gargoyle is an active vision system that is run-time configurable and was designed to meet all of the goals laid out in Section 2.1. This section goes over how Gargoyle was implemented and how this implementation meets those goals.

### *2.2.1 Overview*

Gargoyle accepts commands from a remote client, which could be the reactive planning element of an agent, as seen in Figure 3, or a manual client run by the user. This constructs pipelines and manipulates them by sending messages to Gargoyle. The reactive execution system in the setup in Figure 3 would control the sensors, like the cameras, manipulate the pipelines, and interact with the world. For example, a speaker might move across the room while being filmed, and the client would use the information that Gargoyle passes to it to have the camera follow him.

The Gargoyle system meets its design goals by delivering an integrated framework for designing and using atomic visual units, called modules. An example of a module might be an edge detector or a color histogram back-projector, as in

| Thread | Function | Listens From | Talks To |
|---|---|---|---|
| Server | Controls communication<br>Spawns new threads | Client | Pipelines<br>Client |
| Client | Constructs and controls<br>Gargoyle pipelines | Pipelines<br>Server | Pipelines<br>Server |
| Pipeline | The pipeline is actually a collection of modules | | |
| Modules | Carries out visual tasks | Client | Client |

Table 3: Component breakdown of Gargoyle

Figure 2. Separating these elements into Gargoyle modules allows them to be easily combined to create new visual routines. This allows the routines to be modified, as well as simplifying and encouraging code reuse.

A breakdown of the different components for Gargoyle can be seen in Table 3. These are the functional elements of Gargoyle and will each be examined in detail.

### *2.2.2 Server*

The server is a common contact point for all of the clients and threads. It also spawns off all of the requested threads. When a client requests a new pipeline, the server creates a parser thread to interpret the clients requests. The client can then request that specific modules be constructed into a pipeline. The parser interprets the commands and the server constructs each module in its own thread. This multi-threaded mode of operation allows for the pipelines to interact efficiently without the high overhead of inter-process communication. This is important if there are limited resources and many visual tasks need to be accomplished at the same time, as is the case in many real-time environments.

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. In the monolithic scheme, the client

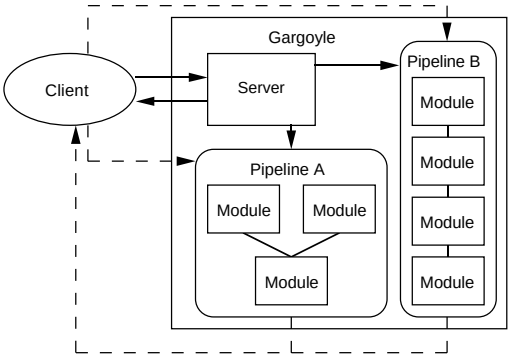


Figure 4: Detail of client control of Gargoyle pipelines

would have to run two different programs to do this, and maintain all of the necessary communications channels.

### *2.2.3 Pipeline*

Pipelines are not actual threads of operation but rather a number of modules working in concert to perform some visual operation. Modules are described in Section 2.2.4. The notion of pipelines is important, because a single client might have multiple visual operations happening at the same time, like the tracking example in Section 2.2.2. When this is the case, the different pipelines can be treated by the client as separate visual processes. 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. The client then has the option to work on constructing a new pipeline without stopping the tracking pipeline. This ability is crucial in the intelligent classroom where there are many visual operations that might need to happen at the same time. For instance, the speaker will always need to be tracked; while at the same time other operations may need to be performed, like watching which way he is looking, or observing his hand gestures. Both of these actions could be performed in other pipelines while the tracking loop was still running without the immediate attention of the client.

Pipelines also provide a simple way of thinking about results rather than methods. A single pipeline might have modules that provide a number of possible methods of accomplishing the same task, as described in the tracking example in Section 2.1. This would allow the client to specialize the pipeline to operate better given the current conditions.

### *2.2.4 Modules*

Modules are the basic operating units of the pipelines. They pass information to each other through the pipeline in the order in which the client has connected them. Each module starts by reading some information, its *input*. Then, using that information, the module does some computation on it. Once the computation has been finished; the results (or the *outputs*) can be set, which are then used by another module. The different modes of communication for the modules are summarized in Table 4.

Outputs from different modules are connected to the inputs of modules that are currently active. These connections define the pipeline, and tell the inputs where to get their data from. A module will run as soon as all of its inputs are filled. For an input to be filled, its connected output must have been set by the output's module. Different modules may need different numbers of inputs depending on the function of that particular module. 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 inputs can only be connected to outputs of the same type. 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.

Once a module has run it fills its output slots with the image information it has generated. A module might have multiple outputs, depending on what type of information it produces. For example, a color histogram module might return a binary image with the areas of correct color marked and a region of interest around

| Comm. Port | Information | Connector |
|---|---|---|
| Pipeline Input | Image Information | Other Modules |
| Pipeline Output | Image Context Info | Other Modules |
| Parameters | Task Context Info | Client |
| Results | Vision Processing Results | Client |

Table 4: Module Communication Methods

the peaks of the color. Trackers might return the location of objects through regions of interest for a verifier further down the pipeline, or as results to the client.

The module's results that interest the client are sent back. In general, modules should be careful about how much information is sent back to the client. The client may request information through the parameters, and may also request that no results be sent from the entire pipeline. Results can give the client information about the world for the client to react to. They can also give information about the functioning of the pipeline so that the client may reconfigure the pipeline to better accommodate the current state. This is how the visual information is currently handed back to the client. This information can be used by the client to figure out what context it is in.

While working on the intelligent classroom, a number of flaws have been found with this method of information exchange with the client. First, it does not provide any way for Gargoyle to synchronize with the client. This is very important for tracking routines which rely on the client for information. Another method of communication, based on input and output modules, will replace the results communication method. These modules are fully described in Chapter 6.

Modules also have parameters which may be set by the client. 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 task context onto the pipeline. The client can also change the configuration of the pipeline, if the results have indicated that some change is warranted. 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. In the future the client will also be able to pass context via input modules. Unlike parameters, input modules will provide information that changes each time through the pipeline.

Modules are the key components of Gargoyle, as 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, visual routines should be broken up into smaller units which can be used in different ways, and even 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.

## 2.3 Gargoyle in use

To ensure that Gargoyle had been developed properly for our vision needs, a number of experiments were done using the system. Gargoyle is still under development and the following chapters are as much experiments to determine the viability of Gargoyle as they are vision projects themselves. Chapter 3 describes the initial experiment of implementing the Perseus pointing task with Gargoyle. That experiment was done during the base development of Gargoyle and many of the problems that we discovered resulted in some of the design considerations discussed in Section 2.1. It also provided many of the base modules that are now being used in the intel-

ligent classroom. Chapters 4 and 5 discuss the intelligent classroom, and how it uses Gargoyle to obtain excellent visual results.

# CHAPTER 3

# PORTING PERSEUS

The first experiment I did was a port of the Perseus system to Gargoyle. Perseus is a complex real-time visual system developed by Roger Kahn[5, 6]. This port provided a look at the proper way to implement visual routines in Gargoyle, as well as providing a base of modules to use in the intelligent classroom. It also brought up the issue of what computation happens where. This is a crucial question if the Gargoyle system is going to provide any value over monolithic systems. The goal is to have the appropriate amount of work being done in each module to allow and encourage reuse. To provide a good base of modules for the intelligent classroom, modules must provide general functionality that can be reused in other domains. This chapter will examine Perseus and how the different elements from Perseus map into Gargoyle modules.

## 3.1 Perseus

Perseus is a large system that runs on specialized hardware and implements a visual routine which tracks people in real-time and recognize gestures. We chose this person tracking system to be the test visual routine for a number of reasons. Since it was already developed we had something with which to compare the Gargoyle version. It also uses a large number of basic visual routines which make good modules in Gargoyle. Perseus interfaces with a higher level planning system, which fits into the Gargoyle concept of a client. Finally, many of the tasks involved in the intelligent classroom require the speaker to be tracked, so the person tracking visual routine provides a nice basis for the intelligent classroom project.

### *3.1.1 How it works*

Perseus divides up all of its operation into six different logical sections: feature maps, object representations(ORs), markers, visual routines, a segmentation map, and a long term visual memory(LTVM). These sections can be seen in Figure 5. The feature maps are retinotopic interpretations of the scene that have had some simple visual operation done to them. These are then made available to the different ORs. The ORs gain some higher level understanding of the scene by examining the feature maps. They then write the information that they find into a global segmentation map. This map provides for communication between the different ORs, since some ORs need information from the other ORs. The ORs are specified by the visual routines. Visual routines in Perseus are instructions on how to use the information from the other elements. This is as opposed to the notion of visual routine in Gargoyle, where the routine is the actual pipeline itself. The Perseus visual routines get their information from the LTVM and the reactive execution system. However, all of the changes that happen once the routine is running come from the visual routines themselves, or the LTVM; not the reactive execution system. If runtime changes need to be made, the visual routine will make them and the LTVM will store them for later use. No task context can be added to the visual routines and no changes can be made once they are started. The ORs store information in markers to which the visual routines then have access. The markers exist continually and are updated by the ORs so the visual routines can get the newest information whenever they need it.

### *3.1.2 The Pointing Task*

It is easiest to understand how Perseus works with an example which will be followed through in Section 3.2. The pointing task was the original task for which Perseus was built. Figure 5 shows the breakdown of the parts of the pointing task that each element of Perseus does. The pointing task is a visual routine which identifies a person in a scene, and then tracks the person to see if he makes a recognized gesture.

Figure 5: Diagram of the pointing task implemented in Perseus.

The routine waits for someone to enter the scene and then tracks their head and hands until the pointing gesture is recognized. The reactive execution system[1] starts the system by requesting the detect pointing system to run with a specific person object representation. The other object representations – background, lights and floor – fill the segmentation map with the information that the person OR uses to find the person in the scene. The different ORs use the intensity, edge, and motion feature maps as necessary. Once the person OR finds a person in the scene to track, the hand and head markers are updated, so that the detect pointing visual routine can use the markers to determine if the person is pointing at something. Only once the pointing is detected does the visual routine return the information to the reactive execution system.

The pointing task lends itself well to use in the intelligent classroom as well as implementation in Gargoyle. In fact, some aspects of the pointing task are better suited to Gargoyle than Perseus. For example, when the reactive execution system tells the pointing detection routine to run in Perseus, it uses motion to decide when a person has entered the scene, and when it should start tracking. This provides no feedback to the execution system. In Gargoyle, the execution system can decide for itself when to look for a person by constructing a motion detection pipeline itself, or by using other context clues that Gargoyle does not have.

For the port, some of the elements of Perseus were broken up into Gargoyle modules. These modules were then used to create the Gargoyle visual routines which are at the heart of the intelligent classroom. The exact operation of each of the Gargoyle modules is described in Section 3.2. A loose mapping from Perseus feature maps and ORs to Gargoyle is given with the descriptions; however, due to the modular nature of Gargoyle, some elements do not have a direct mapping to Gargoyle modules.

[1]The reactive execution system used by Perseus in the pointing task is Jim Firby's RAPS[2]

## 3.2 Gargoyle modules for the pointing task

Table 5 shows the different modules that were written to do the job of the different Perseus feature maps and ORs. The description of Perseus in Section 3.1 shows that there is not a direct mapping from Perseus ORs and feature maps to Gargoyle modules; however many of the functions did map well, and some of them ended up being organized in a more appropriate manner for the task at hand. The concept of a Perseus visual routine[2] was no longer needed in Gargoyle. Since Gargoyle is more tightly coupled with the reactive execution system, the pipeline is controlled directly by the execution system, and the locations are reported directly back to the execution system. Similarly the markers are no longer needed, since the reactive execution system can request the information directly from the pipeline when it needs it. The LTVM is actually split up into the modules and the reactive execution system. In Perseus it would maintain any changes that may have been made to the specification of the object representations. In Gargoyle the client may want to apply information that the visual system does not even know about to the operation of the pipeline, so it makes more sense for the client to remember any changes in a long term sense. An example of this could be if the person object representation developed a color histogram for a specific person. The client would know to use that histogram if it was looking for the same person again, whereas the pipeline has no notion of that sort of higher level task. While the routine is running, the specialization information is stored in the parameters of the specific modules.

The rest of Perseus: feature maps, ORs and the segmentation map, map fairly directly into modules. The pipeline constructed from those modules is shown in Figure 6. In Gargoyle, since any module may be reused by another pipeline, we make no distinction between the ORs and the maps. The feature maps happen earlier in the pipeline, but since their outputs can be linked to any number of module inputs, there is no need for a distinction between feature map modules and other modules.

[2]It is important to note that this paper uses visual routine to mean two different things. The first is a series of commands that retrieve some information from the scene. The other, as it is being used here, is an element of Perseus which controls how information in Perseus is being used.

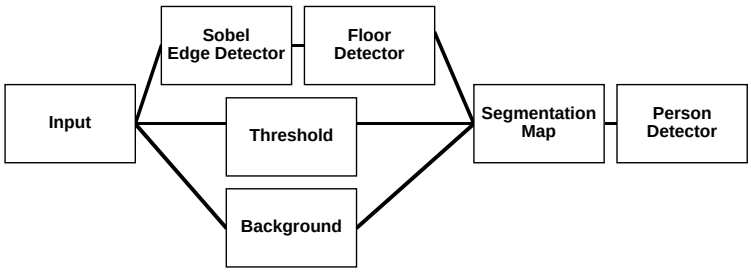


Figure 6: A Gargoyle pipeline for tracking people

| Module | Input | Output | Parameters |
|---|---|---|---|
| Input | None | GSImage | Region |
| Sobel | GSImage | Binary Image | Threshold |
| Free Space | GSImage | Binary Image | Horizon |
| Threshold | Binary Image | Binary Image | Threshold |
| Background | GSImage | Binary Image | Bin Params |
| Segmentation Map | 3× Binary Image | Segmentation Map | None |
| Person Tracker | Segmentation Map | | Person Params |

Table 5: Comparison of the different units in the system.

Table 5 shows all of the modules created for the Perseus pointing task in Gargoyle. The person tracker corresponds to a combination of the functionality of the Person object representation and the markers. The rest map directly to the Perseus object. The following sections describe the operation of each of these modules in detail.

### *3.2.1 Input*

The input module has the functionality of the intensity map. It grabs grayscale frames and gives the requested region to all of the other modules to start the pipeline running. The client may set the region to a subset of the image if there is some constraint on where the person can be. This will reduce the amount of computation that all the other modules will need to do.

Figure 7: The outputs of from left to right: the floor, threshold, and background modules.

### *3.2.2 Sobel and Floor Finding Module*

The sobel module has the functionality of the edge feature map, and the floor finding module has the functionality of the floor OR. The sobel module runs a sobel edge detector on the image provided to it by the input module. This edge image is then fed to the "Free Space" module. This module looks for free space on the floor by looking for edges up from the bottom of the screen[4]. This finds that part of the floor which is not covered by the person or some other object. This is needed because the background module will sometimes notice shadows on the floor cast by the person. It will only search for edges up to a horizon. An example of the output of this module can be seen in Figure 3.2.3. The horizon is a parameter that can be set by the client depending on the angle of the camera.

### *3.2.3 Threshold*

The threshold module is a very simple module that outputs a binary image of its input image, where values above the threshold are 1 and those below 0. The threshold value is set as a parameter. The threshold has the same functionality as the lights OR. The lights can cause blooming and clipping in the input images which confuse the background module, so the threshold can be used to remove those sections.

### *3.2.4 Background*

The background module contains the functionality of the background OR. This module works by observing a scene over time and segmenting out those pixels that are not changing. This is accomplished with an array of histograms, one for each pixel in the input image. When a new image is input to the background module, the value of each pixel is marked in the corresponding histogram. In this way the background module keeps track of the different values it has seen at each pixel. Over time, a histogram will accumulate peaks if its pixel does not change in successive input images. This peak is called the *mode* of the histogram. If a pixel in the input image corresponds to the mode in its histogram, then the pixel is considered to be background.

There are a number of important considerations that must be taken into account for the background to work. First, the background may change over time, a new object may be set down in the background. This object will gradually become part of the background as the histogram shifts to prefer the value of the object. However the time that it takes for the mode to shift to the new value is dependent on how much the histogram has been built up already for the other value. If it has been built up for a very long time, it is hard to change. In order to mitigate this problem, there is a maximum value parameter which defines how large a particular bin in a histogram may become. When it reaches that level all of the bins in that histogram are halved to maintain the proportionality of the histogram. This value defines the maximum amount of time that it can take for the background to shift to take a new object into account. Another consideration is the fact that there might be small amounts of motion in the background, due to lighting fluctuations or other noise. To take this into account, the width of the bins can be set by another parameter.

A final consideration is that this technique of background detection assumes a constant background and a stable camera position. It is often true that an agent may want to look at something else and then look back at the person. Gargoyle is well suited to this task, since the client can stop the pipeline with the background module

while it looks away and restart it when it looks back. Unfortunately the precision of such saccadic movements is often poor. Therefore we implemented an additional feature in our background detector. If requested, the background module will search in a small area around the center of the input image for the best match. The transformation with the best match is chosen as the correct one and the background module will use that in the future.

The output of this module can also be seen in Figure 3.2.3. One may notice the shadow on the floor around the person is also moving. This is why the output is combined with the output of the floor module.

### *3.2.5 Segmentation Map*

The segmentation map combines the outputs of the three object representation modules into a form that the person tracker can use. In general it will combine any number of images with binary information in them, so that all of the information can be obtained off a single image.

### *3.2.6 Person Tracker*

The person tracking module communicates directly with the client and combines with it to accomplish the tasks of the visual routines, markers, and the person object representation. The input to the person tracker module is a segmentation map containing the background, the lights and the floor. The basic assumption is that the person is going to be the only major moving object in the scene. The person throws shadows which are removed by the floor, and the motion caused by the flickering of the lights is removed by the lights. The only remaining object is the person. The actual segmentation is done on this connected component.

The segmentation searches for the top center extremity as the head, and similarly on the sides for the arms, and on the bottom for the feet if requested. These locations can be compared against the height and width and centroid of the person

| Perseus | Gargoyle |
|---|---|
| Faster specialized hardware | Cheaper off the shelf hardware |
| Highly optimized C++ code | Understandable, reusable Java modules |
| Fast access global data | Multiple execution threads |
| Excessive specialization | Tight coupling with client |

Table 6: Advantages and disadvantages of Gargoyle and Perseus

region to make sure that they are reasonable. A number of other constraints are placed on the relative locations of the points to improve the ability of the module to track the person.

Once the points have been found, future searches for the person can be constrained during the tracking process to be near the old point, to speed up the tracking process. For instance if the centroid is moving to the right, the head and hand search regions can be moved appropriately. We can also predict where the centroid will next appear under a linear movement assumption, using a simplification of the recursive least squares method. Since the motion taken by a person is not particularly regular, it doesn't make sense to use the full recursive power of this method. So instead, we only look at the last few points where the person was, since those are more likely to determine the next point. The client, of course, can also provide information to help the tracker reduce its search problem.

## 3.3 Comparison of Gargoyle to Perseus

Gargoyle and Perseus have a number of different advantages over each other, which are inherent given the designs mentioned above. Some of the more interesting differences are mentioned in Table 6. Many of Perseus' advantages come from the fact that it was written in C++ and very highly optimized to run efficiently using our DataCube MV200 vision processing computer. While the specialized hardware and code optimization allow the routines to run in real-time, they cause a number of disadvantages as well. For one thing, the large amount of space taken up by the

MV200 and support computers means that none of the vision processing could be done on board a mobile robot. It also means that the code must be used on this very specialized platform, and is very hard to use again or share with other researchers.

As smaller personal computers become faster and more powerful, we feel that vision processing can be moved onto the cheaper platforms without too much loss in real-time capability. The Gargoyle version of the pointing task runs on a dual processor Pentium 133 MHZ with only a small decrease in performance from the Perseus version. Also, since it is written in Java, it is portable to other platforms, and in fact has been ported to the Apple Macintosh. The only non-portable pieces are the frame grabber and any native methods that have been written for speed considerations.

The main advantage of Gargoyle over Perseus is the module paradigm. This leads to the results shown in Table 1. The modular nature of Gargoyle combined with the inherent portability of Java provides an excellent way for researchers to share their work and reuse parts of old pipelines in new ones.

Finally, the advantages of having tight coupling of the client to Gargoyle cannot be overstated. Gargoyle allows the client to modify the operation of the pipelines dynamically during runtime, whereas Perseus has no mechanism for modifying the visual routine once it has been started. If we are to ever have a system that can usefully understand a general scene it will need to be very tightly coupled with a higher level system. Only by allowing the client to automatically specialize the system to the current context will we be able to break away from the long standing problem in vision of over-specialization to strong assumptions.

# CHAPTER 4

# INTELLIGENT CLASSROOM

The testbed that we are developing for the combination of Gargoyle with a real-time reactive execution system is called the "Intelligent Classroom". The idea is that the multimedia classroom observes the actions going on in the room and controls the multimedia devices in the room accordingly. This is an interesting problem, because it requires the system to run in real-time, and provides a number of different contexts in which the system must operate. It also provides interesting and complex visual problems which will test the limits of our visual technology. This chapter discusses what the intelligent classroom is, and why we picked it as a research domain. Section 4.2 examines some potential tasks for the intelligent classroom, and Section 4.3 looks at the applicability of the combination of Gargoyle and a reactive execution system to these tasks.

## 4.1 Classroom Design

In our intelligent classroom, the room will automatically control the multimedia elements which a lecturer might want to use. A reactive execution system will effect the changes to the world that the user requests in a natural manner. The intelligent classroom will take the place of classroom technicians who do things like film the lecture, play videos, or change slides (see Sec. 4.2). In order to accomplish all of these tasks the room will need to be equipped beyond what is usual for a multimedia classroom. It will need cameras to observe the speaker, computers to run the visual and planning tasks, and control access to all of the multimedia devices. One advantage of the classroom environment over the robot platform is that the main camera used by the visual routines does not necessarily have to move.

| Task | Control | Task Context? |
|---|---|---|
| Filming the speaker | Continuous | No |
| Controlling lights | Singular | Yes |
| Running slide shows | Continuous | Yes |
| Starting videos | Singular | No |

Table 7: Some possible tasks for the intelligent classroom.

Multiple modes of sensing will be useful to the system. Visual gesture recognition is important for a number of the possible tasks that the room will implement; however, gestures are often not the most natural way to communicate intention. Microphones that allow verbal commands to the system will be needed for the interface to the room to be totally natural. In fact, using multiple modes of sensor input provides yet another method for the reactive execution system to determine the context that the speaker is currently in, or going to be in.

## 4.2 Classroom Tasks

There are a number of tasks that could challenge our system in different ways if the system is to control all the multimedia devices available to the speaker. Some examples are shown in Table 7. The table shows the task and what type of control it requires. It also shows whether or not the task in which the execution system is engaged can provide useful context to Gargoyle.

One major task that has many applications beyond the classroom is filming subject. Classes could be filmed for review by students who missed them, or for transmission to students at remote sites, allowing a teleconferencing model of teaching. If the speaker is being filmed, the camera requires continuous tracking of the speaker as he moves around.

In order to make the end result more viewable some notion of framing should be included. Automatic framing of scenes for television shows has been studied[8]. The intelligent classroom, however, will not require the manual assistance that these

systems do, since the classroom environment provides a good deal of framing clues that generic television shows do not. For instance, if the speaker is not moving, zooming in on the head is helpful to capture facial expressions. Also, if the speaker is pointing to something on the board or writing, it would be useful to zoom in on that so that the person watching the film could see the information written there.

A much simpler task would be controlling the lights for the speaker. If there are a number of different lights in the room, the speaker could verbally request that the lights be brought up or down and then point to indicate the specific set. In this case the system does not even need to look for that gesture until signaled verbally. There is also a fair amount of task context that can be brought to bear on the situation. For example, you wouldn't want to put the lights up when someone is in the middle of a slide presentation, or when a video is running.

Slide shows and videos could also be automated, getting rid of the need for multiple, unwieldy remote controls. The speaker could signal for a presentation and point to the television or the slide projector. Once in the slide show context, the system would be able to activate specific routines that look for gestures and listen for cues that would indicate that the speaker wanted the next slide. For example, the words, "the next slide" or some gesture could be used. The video would not need this, since it could just play through to the end. Alternatively, it could allow a video context which would let the user stop, rewind and fast forward.

There are obviously more tasks that could be done – for instance filming questions from the audience – but this small set of tasks covers a wide range of possible problems, and provides a useful starting point for research in natural classroom interfaces.

## 4.3 Applicability

The classroom problem provides a number of interesting tasks for our system to solve. It requires the use of a number of different contexts; and there are different modes of operation for the system to utilize, like the video mode, or the

slide show mode described in section 4.2. It also provides a changing environment that the system can control to some extent. For instance, different versions of the same pipelines may be needed in low light conditions, when the lights are lowered.

A simple modification to the classroom idea allows even more task context to be brought to bear. If the user is willing to provide a script for the system to read, then the context assignment would be even easier. The system would only have to match the current actions to the location in the script. For example, if the slides are in Powerpoint, then the script might be encoded in the notes section. Such a script could range from simple notes annotating a slide show to complex instructions, depending on the user. The end goal, of course, is to allow a totally novice user, such as a visiting guest lecturer, to come in and make full use of the technology.

With this goal in mind, it is clear that the system will eventually have to have complex understanding of the scene. The classroom provides a rich, varying, real environment that stretches the bounds of technical ability, for the reactive execution system, Gargoyle, and some sort of higher level plan recognition system. The intelligent classroom environment will be a good test for all of the tools that we have developed.

Finally, it is clear that these tasks have wider application than simply classroom use. There are many computer vision applications for which context could help solve problems. Teleconferencing is one area in which many of the same solutions could be used. Other applications with similar needs are smart houses – where the house can watch what people are doing and control the environment – or security monitoring, among other things.

# CHAPTER 5

# CURRENT IMPLEMENTATION

The current system is a very limited intelligent classroom environment. The reactive execution system has no physical control over the room. Instead, the reactive execution system informs the user of the actions it would take if it could. The user is a lecturer, giving a talk that is to be transmitted via video. As in Chapter 4, the real classroom would direct the video camera to frame a specific shot in the scene, whereas our limited classroom marks the framing of the shots using a Gargoyle display module. This is only a test system to demonstrate the feasibility of using the Gargoyle system for this task.

## 5.1 Controlling Gargoyle Automatically

This is the first time that we have implemented the full system with Gargoyle and a reactive planner. All of the experiments in section 3.1 were run by a manual client. One of the main advantages of using Gargoyle in the classroom situation is its ability to interact closely with a reactive planner; so one of the first tasks was to test this ability. The task that we set it to was the previously discussed task of shot framing.

To frame the shot, the system needs to know where the person is and what the person is doing. Since the observing camera will most likely be stationary for most of the classroom tasks, the background model of person tracking discussed in section 3.2 is an appropriate one. The client sets up the pipeline exactly as described for the pointing task. The information about the body parts' locations is then used to determine appropriate framing for each frame.

| Speaker | Meaning | Action |
|---|---|---|
| Standing | Lecturing | Zoom in on head |
| Waving Arms | Gesturing | Zoom on upper body |
| Moving | Moving | Zoom out |

Table 8: Implemented Framings

The possible framings are shown in table 8. To determine what the speaker is doing, the client tracks the lecturer's body parts similarly to the person tracker module. However, the client can make smarter assumptions about the scene, depending on what the speaker is doing. If the speaker is simply standing and lecturing, then an actual recursive tracking method can be used, since the movement will be regular. Similarly if the speaker is gesturing, without moving his body, the client does not need to track the hands in any real way until they stop moving. Actually, since the modules do most of the tracking internally for this task, the client really only needs to see if the body parts are moving outside of some threshold. Then the tracking only needs to be done when the part is not moving, to allow for noise and slow small motion on the part of the speaker.

The different states that the speaker may be in are chosen with the concept of good framing in mind. If the speaker is just standing and talking, it is best to zoom in on the face to get as much facial expression as possible. On the other hand, if the speaker is gesturing, then the video camera should zoom out to show the gestures to the viewer. Finally if the person is moving around, it is best to zoom out to allow for imprecise tracking of the person as he moves. The tracking of the center to predict the speakers motion while he is moving is identical to that of the person tracker.

Another Gargoyle module was written to show the framing that the client requested. It simply draws a box requested by the client onto the input frame.

# CHAPTER 6

# FUTURE DIRECTIONS

This project is currently still in the experimental stages and there are many more lines of investigation to pursue. This chapter examines some of those.

## 6.1 Intelligent Classroom

The next step to take in the intelligent classroom project is to recognize other gestures, and make the client more adaptive to the environment. This will allow us to do the other tasks that were described in chapter 4. To provide a more natural interface to the classroom, it will be important to incorporate some simple verbal command mechanism into the system. This will also add a number of context clues to ease the visual tasks.

The basis for doing all of the suggested tasks exists in Gargoyle already. Pointing can be recognized using the line through the head and the hand that is pointing, if the speaker is pointing perpendicularly to the camera. Other simple gestures could also be recognized in this manner. If more specific gestures were needed the new visual routines would be built. These routines could be directed to specific locations by the client, using results from the other routines. For instance, if a hand gesture needed to be recognized, a pan-tilt-zoom camera could be trained on the hand, using the information from the person tracker. Then a new hand gesture recognizing pipeline could be used.

Focus of attention is actually a very interesting problem in the classroom scenario. Beyond the more standard issues of knowing where in the scene to search for objects, there is also the filming issue. If the class is being filmed for transmission or later viewing, there might be a number of possible sources of input; for example, the

speaker, the audience, a slide, or a video. The question then arises of which source to record. To know which input to use, some knowledge will be needed about what the intention of the speaker is. Building a client that is able to recognize the speaker's goals would aid this task greatly. This would allow the system to figure out what the audience is supposed to be watching at any particular moment. Thus, if the speaker started a video, that could be recorded. Then as the speaker narrated over it, the system could determine whether to continue recording the video or switch back to the speaker camera.

## 6.2 Gargoyle

While Gargoyle is fairly complete and capable, as shown by the Perseus experiment, there are and will undoubtedly be some weaknesses in the design that we did not foresee. The initial experiments combining Gargoyle with the client for the intelligent classroom have shown a number of issues that will be dealt with in future versions of Gargoyle.

The current key problem is communication between the modules and the client. Gargoyle provides a very rich method of allowing the client to communicate with the pipeline, but it does not provide a similarly rich environment in the other direction. The only way that a module can communicate with the client is by a direct write function. Each time an individual module runs, this message will be sent. If a module is in a tight loop, the only way for the client to indicate that it is ready to hear from the pipeline is by setting a parameter on the sending module. A much better solution would be for there to be two communications modes: one in which the communication is continuous as it is now, and one in which the communication is served on request by the client. In the latter case, Gargoyle would keep the most recent transmission from the pipeline and send it through only when it was requested.

Another improvement could be made in the way that Gargoyle tracks. Tracking currently happens in one of two ways: internal to a module, or via the client. Through the client, one module tells the client the current location of the

tracked object. The client can then set the region of interest around the predicted location of the object for the next run through the loop. The other method has the tracking going on internally in the module. This has the disadvantage that the savings in computation from tracking is only gained in a single module. If there are multiple modules that feed only into the tracking module, they could also be sped up by the focus of attention provided by the tracking information. It would be nice to be able to have that kind of advantage without the need for the client to do all of the work. This will require some method for upstream communication within the pipeline itself. If the client could tell the tracker to feed its tracking information to a specific module upstream, that would allow the client to set the entire pipeline tracking an object quickly while it did something else.

Obviously there are probably other improvements or issues that we have not yet thought of, that will need to be dealt with as they arise through use and further implementation of the system.

# CHAPTER 7

# SUMMARY AND CONCLUSIONS

There are many things still to be done on the road to the complete intelligent classroom. However, I believe that we have already made the first steps that will get us there. As Java virtual machines become better, and faster personal computers with more CPUs start coming out, the performance of Gargoyle should reach real-time speed without sacrificing on any of the algorithms.

In the Perseus example, we have shown that Gargoyle is a useful tool for quickly creating complex visual routines for a real-time environment. This, combined with the portability of Gargoyle, will hopefully encourage others to use it. However, the real advantages of Gargoyle will not be seen until it has been combined with a client that takes full advantage of the context of a scene. That is what will show Gargoyle to be very useful and much more powerful than stand-alone vision applications.

# REFERENCES

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

[2] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale, 1989.

[3] David Franklin, Roger E. Kahn, 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.

[4] I. Horswill. Polly: A vision-based artificial agent. *American Association for Artificial Intelligence*, 1993.

[5] Roger E. Kahn. *Perseus: An Extensible Vision System for Human-Machine Interaction.* PhD thesis, The University of Chicago, August 1996.

[6] 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.

[7] David Kortenkamp, Eric Huber, and Peter Bonasso. Recognizing and interpreting gestures on a mobile robot. *American Association for Artificial Intelligence*, 1996.

[8] Claudio S. Pinhanez and Aaron F. Bobick. Intelligent studios: Using computer vision to control tv cameras. *IJCAI Workshop on Entertainment and AI/Alife*, April 1995.

[9] P. N. Prokopowicz, Roger E. Kahn, R. James Firby, and Michael J. Swain. Gargoyle: Context-sensitive active vision for mobile robots. *American Association for Artifical Intelligence*, 1996.

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

[11] 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.