THE USE OF CONTEXTUAL CLUES IN REDUCING FALSE POSITIVES IN AN EFFICIENT VISION-BASED HEAD GESTURE RECOGNITION SYSTEM

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Brian M. Blonski

COMMITTEE MEMBERSHIP

TITLE:	The Use of Contextual Clues in Reducing False Positives in an Efficient Vision-Based Head Gesture Recognition System
AUTHOR:	Brian M. Blonski
DATE SUBMITTED:	May 2010
COMMITTEE CHAIR:	Franz J. Kurfess, Ph.D.
COMMITTEE MEMBER:	Clark Turner, Ph.D.
COMMITTEE MEMBER:	David Janzen, Ph.D.

Abstract

The Use of Contextual Clues in Reducing False Positives in an Efficient Vision-Based Head Gesture Recognition System

Brian M. Blonski

This thesis explores the use of head gesture recognition as an intuitive interface for computer interaction. This research presents a novel vision-based head gesture recognition system which utilizes contextual clues to reduce false positives. The system is used as a computer interface for answering dialog boxes. This work seeks to validate similar research, but focuses on using more efficient techniques using everyday hardware. A survey of image processing techniques for recognizing and tracking facial features is presented along with a comparison of several methods for tracking and identifying gestures over time. The design explains an efficient reusable head gesture recognition system using efficient lightweight algorithms to minimize resource utilization. The research conducted consists of a comparison between the base gesture recognition system and an optimized system that uses contextual clues to reduce false positives. The results confirm that simple contextual clues can lead to a significant reduction of false positives. The head gesture recognition system achieves an overall accuracy of 96% using contextual clues and significantly reduces false positives. In addition, the results from a usability study are presented showing that head gesture recognition is considered an intuitive interface and desirable above conventional input for answering dialog boxes. By providing the detailed design and architecture of a head gesture recognition system using efficient techniques and simple hardware, this thesis demonstrates the feasibility of implementing head gesture recognition as an intuitive form of interaction using preexisting infrastructure, and also provides

evidence that such a system is desirable.

Acknowledgements

I'd like to thank my advisor Franz Kurfess for his assistance and guidance. Without his support, I would have had much more trouble finding and narrowing down my thesis topic. I would also like to thank Lynne Slivovsky for getting me interested in the field of computer vision back when I was considering senior projects.

I would like to thank David Janzen and Clark Turner for their guidance through my graduate course work. They have both made my graduate work much more interesting and taught me a great deal. David Janzen has taught me a lot about software development and Clark Turner has always made hard work seem fun.

I would like to thank my family and friends for their support and encouragement: my parents, who have always encouraged me to do my best but never pressure me even when my thesis took longer than promised; my older brother Casey's teasing to help me put down the video game controller when I should be working, and my girlfriend Mae Richie who helped me stay sane after staying up until 3:00am regularly every night trying to finish in time. Finally, I'd like to thank all the people who helped me gather results for my thesis by sitting through my annoying tests.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview of Head Gesture Recognition

Head gesture recognition is the process of analyzing head movements over time to determine when intentional gestures are made. Advanced image processing techniques can detect and track facial features, and trained statistical models can be used to detect common gestures from such movements. This combination of tracking and analysis provides an additional interface through gesture recognition.

Rick Kjeldsen explores the applications of head gesture interfaces in *Head gestures for computer control*[12]. He identifies four categories: pointing, continuous control, spatial selection, and symbolic selection. Most head gesture implementations focus on symbolic selection which includes nodding and shaking as means for confirmation and rejection. This thesis will focus solely on symbolic selection.

Gesture Recognition techniques face many challenges due to accuracy and efficiency constraints. The system must be efficient enough to perform in real-time, and robust enough to handle a wide variety of environments and users. Despite

a number of promising attempts, head gestures have yet to gain traction outside specialized domains. However, more and more integrated devices include web cameras and suffer from reduce interfaces. Head gestures may offer a convenient supplemental form of input. Gesture based input has also been gaining attention in video games. Microsoft's upcoming Project Natal may introduce vision-based gesture recognition to a much wider audience.

In this thesis, I explore previous gesture recognition systems as well as develop my own gesture recognition system. The defining feature of my attempt is the focus on efficiency and the use of contextual clues to reduce false positives. I study the effects of using contextual clues by comparing performance before and after adding contextual clues to the system.

1.2 Motivation

Users interact with an ever increasing number of devices on a daily basis. As users face more and more diverse devices, they require simpler, more intuitive, and more intelligent forms of interacting with their devices. Simplified and natural interfaces reduce the learning curve a user needs to operate their devices effectively which could greatly boost productivity and user satisfaction. Head gestures may provide such an interface in some situations. The usability studies conducted in *Head gesture recognition in intelligent interfaces: the role of context in improving recognition*[18] by Morency and Darrell shows that users found it both more natural and more efficient to use head gestures over the mouse or keyboard in the domain of answering dialog boxes. This work serves as a large inspiration for this thesis. I seek to validate these results with my own study.

There are many specialized applications where head gestures could prove use-

ful. Devices where traditional interfaces are limited such as kiosks or smartphones could benefit from a head gesture user interface. A user could control their device without using a mouse or keyboard, or even without touching the device at all.

Intelligent interfaces could also utilize head gestures. By monitoring movements from the user, the system can gather information about the user and how they interact with the system. An intelligent interface could determine aspects of the user such as user attention, focus, frustration, comprehension, and more. Understanding user gestures, intentional and unintentional, could greatly advance the field of human-computer interaction. Interfaces presented in *Gaze-based infotainment agents*[25] and *Gazemarks: gaze-based visual placeholders to ease attention switching*[11] use gaze to determine user focus and adjust to create a more personalized user experience.

Head gestures provide a great opportunity for interaction because they are a natural form of communication to most users. We can adapt head gestures to provide a simple and intuitive interface for users for certain tasks [20]. Due to the ubiquity and low cost of web cameras, vision based gesture recognition presents a viable new form of interaction. Taking advantage of established infrastructure, we can to provide a natural and intuitive interface to many devices through the use of head gestures.

Chapter 2

Problem Statement

2.1 Objective

In this thesis, I attempt to verify that contextual clues can be used to significantly reduce the number of false positives in a head gesture recognition system. My system uses more efficient techniques and different image processing algorithms then previously attempted. I compare the accuracy and efficiency of my head gesture recognition system with and without the contextual clues optimization. In addition, this thesis also serves as a survey of gesture recognition techniques. I summarize some of the most common image processing algorithms and gesture recognition techniques. I also examine usability of head gesture systems for answering dialog boxes during specific tasks.

During development, I aspired to create an accurate and efficient head gesture recognition system without the use of any special hardware beyond a simple web camera. Some special types of cameras can assist image recognition, but do not take advantage of the already widely established web camera base. I also designed

the system with low end machines in mind. Smaller devices such as netbooks and smartphones have become quite popular, but they lack the power of a traditional desktop. A head gesture recognition system that can run on low end machines could target such devices and bring head gestures where they may be needed most. In this thesis, I use a low end netbook for my evaluations.

Many techniques for recognizing head gestures have been developed. Choosing the appropriate image processing techniques is key to success. This thesis identifies and compares several techniques for recognizing head gestures. I perform a survey of current vision based head gesture recognition techniques including their strengths and weaknesses. I identify the most accurate and efficient head gesture recognition techniques, including those that use special hardware. I also compare different techniques for analyzing and recognizing gestures.

For this thesis, accuracy is defined as the percentage of true positives amount out of all true positives, false positives, and missed gestures. Although I include an analysis of gesture miss rate, I mainly focus on the effects of contextual clues on false positives. I define the following terms in this thesis:

- A true positive is when the system correctly detects an intentional head gesture.
- A false positive is when the system detects a gesture without the user's intention, regardless if the gesture performed correctly or not.
- A missed gesture is an intentional gesture not detected by the system.
- Efficiency refers to CPU load and frames per second of the system.

Chapter 3 contains a survey of gesture recognition systems using accuracy and efficiency as metrics. Additionally I use special hardware, such as stereo camera

or IR camera, as an additional metric. I chose these criteria on the assumption that, to be adopted by users, head gesture recognition must be accurate, have low system overhead, and require no special or expensive hardware beyond a simple web camera.

2.2 Problems and Challenges

Object detection and image processing are long standing challenges that computer scientists have worked long to overcome. While many advanced image processing techniques exists for object detection, a gesture recognition system requires real-time performance which poses an even greater challenge. Accuracy must balance with efficiency to achieve a robust real-time system. Other common challenges include false positives in facial recognition, false positives in gesture recognition, and misses of gestures that are too small and quick[8]. Tracking facial features at extreme angles poses another challenge. Many implementations depend on the user's eyes as tracking points and may lose these points at large angles during a head shake.

My main goal is to address the challenge of false positives. Hidetoshi Nonaka refers to this as the "Midas Touch" problem in *Communication Interface with Eye-Gaze and Head Gesture Using Successive DP Matching and Fuzzy Inference*[24]. The Midas Touch problem refers to the famous parable of King Midas, whose touch turned anything to gold. While desirable some of the time, it quickly becomes a detriment if constantly active, as Midas discovered when his food, drink, and even daughter turned to gold at his touch. The Midas Touch problem applies to head gesture recognition because a user may make unintentional gestures with their head that could be mistaken for a command. Since

users often make unintentional gestures with their head during everyday interaction, false positives are frequent[19, 18, 20]. Reducing false positives is therefore crucial to developing a usable gesture recognition system.

Environmental variations pose another challenge to image processing and object recognition techniques. The diverse range of background environments possible complicate object recognition attempts, and changes in lighting can cause some systems to lose tracking even if detection initially succeeds. Variations in user pose and position further complicate things, especially if tracking movements such as rotation over time. Choosing appropriate object recognition and tracking techniques is critical to a successful gesture tracking system. Some techniques deal with different environments or changing lighting better than others. Updating the tracking template can help alleviate this problem, but also introduces the risk of degrading over time if the template is not updated carefully[3].

Although there are a great number of challenges to developing a head gesture recognition system, computer scientists have spent decades devising clever techniques to overcome these challenges. In Chapter 3, I review some of these techniques and their applications. In Chapter 4, I describe the techniques I chose for my system, the general design of my system, and the optimization I made to use contextual clues to reduce false positives. In Chapter 5, I describe the actual system architecture, as well as the processes and libraries used to develop it. In Chapter 6, I evaluate my system's performance and the effects of the contextual optimization on false positives. I also briefly touch on usability of the system. Finally, in Chapter 7, I summarize the work done in this thesis and identify areas for future work.

Chapter 3

Related Work

3.1 Overview

There are many different implementations of vision based head gesture recognition software. In this thesis, I survey many different image processing techniques used to detect and track faces, as well as methods for analyzing movement and detecting gestures.

Some implementations involves special hardware such as the IBM PupilCam[4] or a stereo vision camera[20]. This special hardware assists with detecting and tracking the face which simplifies the image processing required. The PupilCam uses stereo IR cameras to generate alternating colors within a users pupils. This makes detecting and tracking the pupils relatively simple. While implementations like these can be fairly robust, accurate, and efficient, they have the drawback of requiring additional hardware that may not be immediately available to many users. Users would have to purchase such hardware before they were able to use the head gesture interface. This may be appropriate for some applications,

but it is unlikely most users will want to buy additional and possibly expensive equipment.

Implementing head gestures with a simple web camera leverages previously installed infrastructure and provides a low cost option for the user. Many small integrated devices such as smartphones often include cameras. There have been many successful implementations that do not require special hardware[2, 8, 22, 14]. For implementations that use common web cameras, it is more important to have a strong feature recognition component because the hardware cannot assist in feature detection.

Using contextual clues in gesture recognition is not a completely novel idea. Louis-Philippe Morency and Trevor Darrell implemented contextual clues in a head gesture recognition system[18, 19, 20]. In their work, contextual clues are used to predict when head gestures are expected to improve accuracy. A small delay is used to insure users have time to read the dialog box and do not accidently dismiss it too quickly. Morency and Darrell use a complex and computationally expensive Support Vector Machine for their contextual based gesture recognition. My system differs by using a much less computationally expensive Finite State Machine and well chosen image processing techniques with contextual clues to achieve a similar effect. The Morency and Darrell also performed studies to find likely time intervals for gestures and performed a usability study on using head gestures to answer dialog boxes. The implementation in *Contextual Recognition of Head Gestures*[19] shows an increase in about 15% in gesture recognition accuracy using contextual clues compared to the system without contextual clues. This thesis attempts to validate those results using a finite state machine and different image recognition techniques. Morency and Darrells' implementation also relies on a stereo vision based approach and therefore violates one of the requirements

I outline for this thesis.

Another implementation to use contextual clues is described in *Infotainment Devices Control by Eye Gaze and Gesture Recognition Fusion*[21] uses eye gaze as a contextual clue for controlling an mp3 player with head gestures. The system uses an embedded device with a CCD camera. This provides an example where head gestures may be useful.

Often multiple techniques are combined to achieve greater accuracy than any one technique can achieve alone. The implementation presented by Crowley and Berard in *Multi-modal tracking of faces for video communications*[3] used multiple vision processes and attempted to select the optimal process by using confidence factors. This system ran at 20 frames per second on a 150 MHz CPU.

I identify two main components in head gesture recognition system: Feature Detection and Gesture Recognition. In feature detection, visual techniques are used to identify areas or features of interest. Tracking occurs by comparing the position of these features between frames. By observing change in location between frames, gestures can be identified by using a Gesture Recognition technique.

In the following sections, I survey many of the techniques available for feature detection and motion tracking.

3.2 Feature Detection

Feature detection is a complex problem that many are still trying to solve. This thesis focuses specifically on facial feature detection. While a lot of progress has been made, feature detection still falls far short of human perception. For

example, images taken in outdoor environments still pose quite a challenge due to changes in lighting and pose[35]. Variations in skin-color, expression, and facial appearance, as well as the possible occurrence of obscuring objects such as glasses or a mustache make facial detection quite difficult[27].

There are numerous techniques available for detecting faces and features in images. Faces have a number of unique characteristics that help identify them, but they also can largely vary between users and even between facial expressions of the same user. In the domain of gesture recognition, we can make some reasonable assumptions that will narrow the scope of the feature detection required. Inferences about distance, position, and pose of the user's face can greatly simplify the techniques needed. The methods for feature detection can vary between each implementation, and often multiple techniques are used[3, 9, 14]. The following are several methods used in facial detection.

3.2.1 Background Subtraction

The method of background subtraction relies on the fact that the user's face will likely be moving in the image while most of the background is static. It works by subtracting the similar areas between two frames, leaving only the change in the image [3, 7, 10]. This often outlines the user's head as it detects subtle changes between the user and the background as the user's head drifts. Unfortunately this technique is very susceptible to lighting changes and non-static backgrounds.

3.2.2 Blink Detection

A subset of background subtraction used in the implementation presented by Shinjiro Kawato and Nobuji Tetsutani in *Detection and Tracking of Eyes for*

Gaze-camera Control[10] is blink detection. Blink detection works on a very simple principle. It counts on the fact that the user must blink periodically to help it detect eye locations. The sudden change between frames is easily detectable using background subtraction and can be used to identify the location of the user's eyes. The eyes can then be tracked between subsequent frames with a different technique.

3.2.3 Color-of-Skin

Another common technique used in facial detection is to apply some a color-of-skin filter to find the areas that are likely to be the user's face. This technique is computationally fast and reliable, but not very precise[3]. It assumes that the face is the most prominent skin area, and that there are no skin colored objects in the background. It is a widespread technique, and is often combined with other techniques to improve precision[3, 4, 7, 14]. Like background subtraction, this technique can be sensitive to lighting changes. There are also differences between user skin colors as well which requires calibration of skin template. In the implementation presented by Crowley and Berard, the system updates the color template periodically by detecting known skin areas using blink detection[3]. This automatically adapts to the user and lighting changes.

The use of skin color filtering is common in many implementations [7, 9, 10, 14]. This technique is often used to limit the area for a more computationally intensive algorithm to search. The feature can then be tracked between images without applying the skin-color filter.

3.2.4 Sum-of-Squared Difference (SSD)

Sum-of-squared differences (SSD) is a mathematical method for determining similar parts of an image[3, 23]. SSD works by taking a feature template and comparing it with sections of the image. The section that is found to be most similar to the template is then assumed to be the feature being tracked[23]. This method is very effective in tracking features once they have been identified.

Limiting the area of comparison can cut down on processing of the algorithm. For example, this algorithm can be run on portions of the image that are known to be the same skin color as the feature template by using a skin color filter first. An more efficient implementation would be to limit the area to within a close region of the last known location of the feature [3].

3.2.5 Pupil Detection

This technique identifies the location of the user's eyes and tracks their position. Eyes are a fairly constant feature across users and are ideal for tracking. The implementation presented in by Davis in *A Perceptual User Interface for Recognizing Head Gesture Acknowledgements*[4] uses the IBM PupilCam to locate the user's pupils. The IBM PupilCam uses unique characteristics of the eye to facilitate location. It uses stereo-vision infrared (IR) light to produce an effect similar to red eye. The two cameras and IR LEDs are multiplexed to produce alternating bright and dark effects in the pupils that can then easily be identified and tracked between frames. Another implementation uses a similar IR camera to the same effect[8]. This technique is accurate and efficient, but requires special hardware to implement.

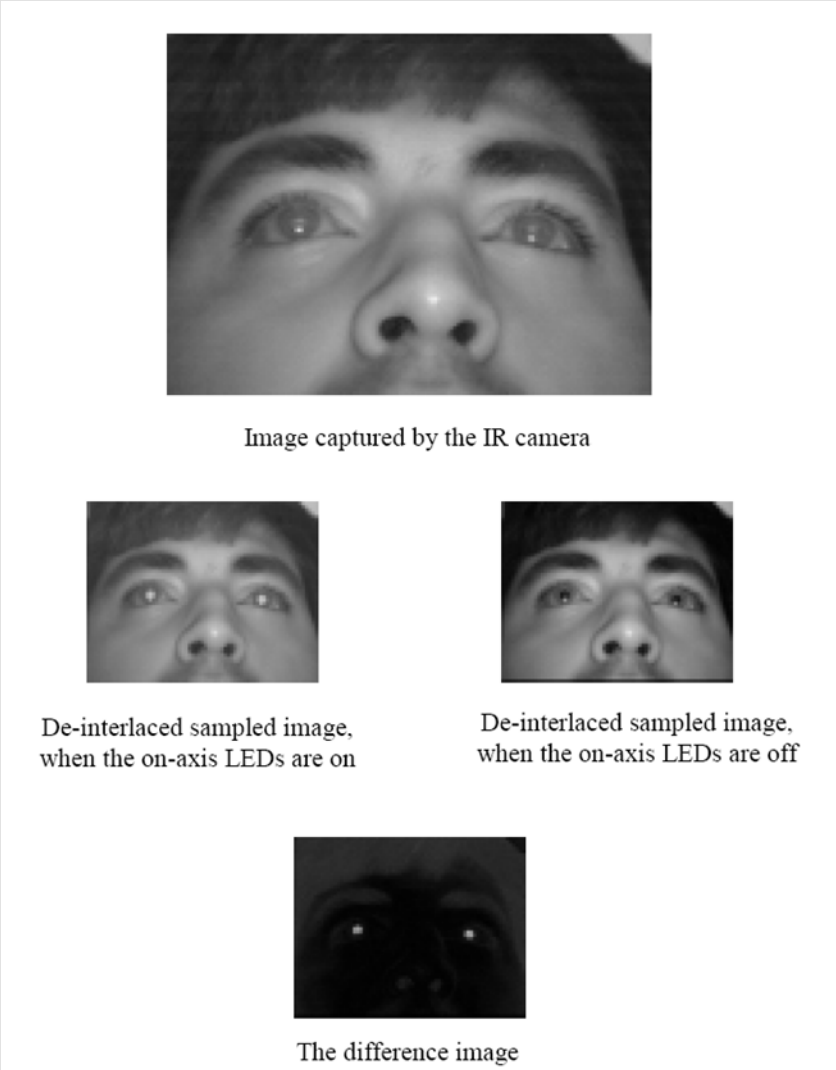

Figure 3.1: Pupil Detection Using an IR Camera [8]

There are other techniques that can track the locations of the eyes without special hardware. A combination of blink detection and SSD tracking, presented earlier, can also effectively be used for pupil detection and tracking. The eyes can also be detected through more complex methods such as using heuristic rules based on the geometrical characteristics of the face as described by Choi and Rhee in *Head gesture recognition using HMMs*[2].

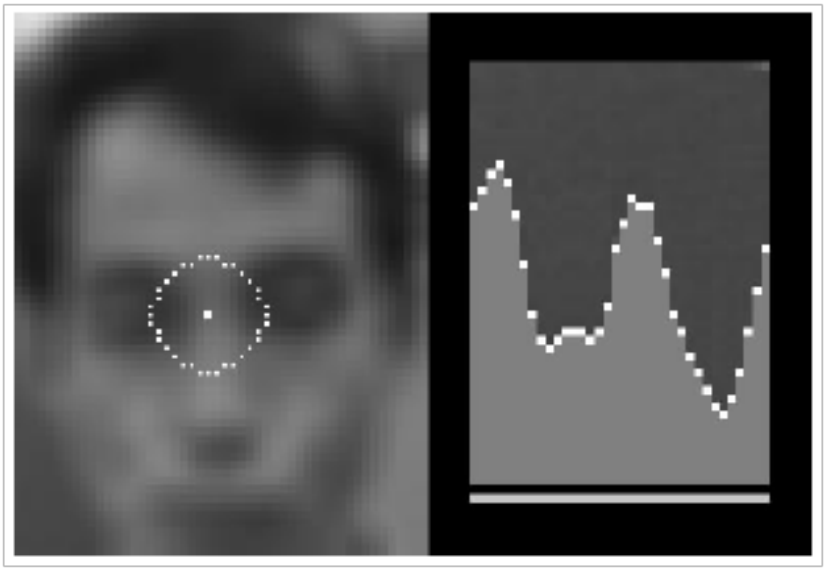

Figure 3.2: Circle-Filter with Pixel Values Around Circle Diameter [9]

3.2.6 Circle-Filter

The circle-filter is a simple yet effective technique that uses the unique characteristics of the face to determine the "between-the-eyes" point. This technique is used in the implementation presented in *Real-Time Detection of Nodding and Head-Shaking by Directly Detecting and Tracking the "Between-Eyes"*[9]. It works on the assumption that the forehead and bridge of the nose are relative bright spots compared to the eyes. This corresponds to two cycles of alternating bright and dark spots on a circle centered between the eyes. Using a circle-frequency filter, local maximum are identified. The true between-the-eyes point is then determined from these points using simple feature criteria[9]. Once the true between-the-eyes point is detected, it is used as a template and tracked. This implementation runs at 13 frames per second on a 175 MHz CPU.

3.2.7 Haar Classifier Cascasdes

Haar Classifier Cascades efficiently detect Haar-like features using cascade classifiers. This method was first presented by Viola and Jones in *Rapid Object Detection using a Boosted Cascade of Simple Features*[30] for the rapid detection of any object using AdaBoost classifier cascades. A Haar Classifier detects Haar-like features rather than pixels. Haar-like features detect change in contrast between adjacent rectangular groups of pixels. Haar Classifier Cascades boost efficiency by eliminating large areas of an image from processing by using only a few Haar features. Scaling is also very efficient using Haar Classifier Cascades. The accuracy of a Haar Cascade Classifier can be adjusted by changing the number of stages in the cascade. Decreasing the number of cascades increases the positive hit rate, but also the false positive rate[31].

The Haar Classifier Cascade method can perform accurate real time feature detection, but has some weaknesses. Accuracy of a Haar Classifier Cascade largely depends on how well the classifier is trained. False positives are a large problem due to naturally occurring face like shapes in the environment. Also, it can still be computationally expensive to continually detect facial features between each frame, requiring continual reprocessing of large areas that are irrelevant. Most importantly, Haar Classifiers have poor point tracking abilities. While it can identify the location of the face fairly easily, it is not very robust in tracking small movements such as rotation where the general location of the object remains stationary. The Haar Classifier identifies a general area where the face may be located.

Sub-features could be tracked using Haar Classifier Cascades to increase point tracking, but this would require additional processing and increase the chance

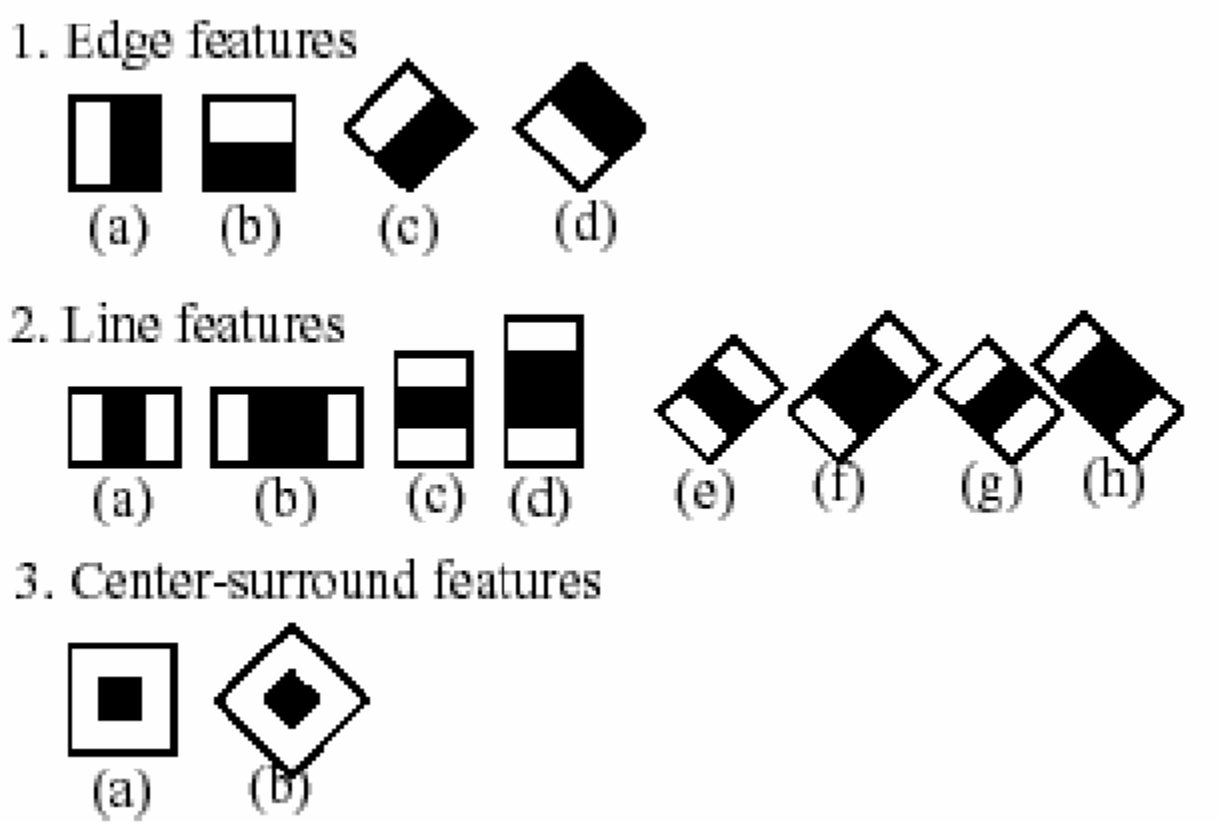

Figure 3.3: Common Haar Features [32]

for false positives. A reasonable implementation of head tracking would use Haar Classifiers to identify the face area and sub-feature classifiers to identify and track features such as the user's eyes[32].

3.2.8 Lucas-Kanade

For point tracking, I use the Lucas-Kanade algorithm, first presented by Bruce D. Lucas and Takeo Kanade in *An Iterative Image Registration Technique with an Application to Stereo Vision* [15] and later refined in *Detection and Tracking of Point Features* [29]. This approach works by minimizing the SSD difference between sequential images. This algorithm is very efficient and can track point movement as well as scaling, rotation, and shearing. Most traditional methods are very computationally expensive, but the Lucas-Kanade tracking algorithm greatly increases efficiency by reducing the number of potential matches between images. The Lucas-Kanade algorithm also has applications for stereo vision[15].

The biggest factor for determining accuracy of the Lucas-Kanade tracking algorithm is the points chosen for tracking. By choosing key areas on the face for tracking (including points around the eyes, nose, and mouth), the Lucas-Kanade algorithm can track features with subpixel accuracy. The Lucas-Kanade algorithm's ability to track rotation and shearing help track the facial features as they rotate and/or are lost during a head gesture such as a shake or nod. The Lucas-Kanade tracking algorithm's ability to follow scaling is also helpful in maintaining tracking although less important for the purposes of this research.

3.3 Gesture Recognition

After the face is identified, it can be tracked between frames. The resulting motions can generate gestures. There are several different methods for identifying gestures that offer varying amounts of accuracy and speed. The two most common techniques are Finite-State Machines (FSM) and Hidden Markov Models (HMM) [2, 4, 8, 17, 22, 14]. Other methods include neural networks [2, 6, 8, 20, 24] and Support Vector Machines (SVM) [18, 19, 20, 28].

3.3.1 Finite State Machines (FSM)

Finite State Machines (FSM) is a model composed of a number of finite states and transitions between those states. FSMs model head gestures as an ordered set of states in spatio-temporal space[17]. FSMs are generally less computationally expensive than more advanced methods such as Hidden Markov Models and Neural Networks. The FSM uses the continuous stream of feature trajectories to determine if the FSM should change state or stay in the current one. When the FSM reaches its final state, the gesture is recognized.

FSM are simple and easy to implement and can offer reasonable performance. The implementation in *A Perceptual User Interface for Recognizing Head Gesture Acknowledgements*[4] uses a timed FSM to recognize nods and shakes. No conclusive results are offered, but no false positives were generated in the example used. This system ran at 30Hz on a 1Ghz Pentium III computer. However this implementation used additional special hardware such as the IBM PupilCam and a Maxtor Meteor II frame grabber.

3.3.2 Hidden Markov Models (HMM)

Hidden Markov Models (HMM) are statistical models that assume that the outcome of any state only depends on its current state and not any previous state. The parameters for a HMM are unknown and must be trained from observable data. HMMs are computationally more demanding than FSMs and require a large amount of training data [17]. HMM can offer high accuracy, but must be trained for proper performance.

Two separate implementation of HMMs, presented in *Head gesture recognition using HMMs*[2] uses two and then three HMMs in gesture detection. One HMM is used for the x-axis and another is used for the y-axis. This system assumes that head nodding includes a lot of y-axis movement, and shaking requires a lot of x-axis movement. Each HMM monitors its respective axis for a movement and presents a confidence rating of that gesture being performed. The second implementation adds a third HMM to help identify neutral gestures which are gestures with a lot of x **and** y axis movement. However, the addition of this third HMM resulted in lower gesture recognition accuracy.

3.3.3 Support Vector Machines and Neural Networks

Two other techniques worth mentioning are Support Vector Machines (SVM) and Neural Networks. Both are complex and computationally expensive techniques that must be trained similarly to HMMs. The SVM approach for head gesture recognition has outperformed previous techniques based on HMMs[18]. Neural Networks have also been used in several implementations[32, 31, 5, 34, 33].

Neural Networks have been used for both feature detection[26] and gesture

recognition[28]. Neural networks are much more complex than FSMs and even HMMs and do not appear to offer any significant advantages.

In the implementation presented by Morency et al. uses contextual clues to improve the accuracy of a SVM head gesture recognition system[18]. Morency et al. reported that they chose a SVM because of its superiority to HMMs in accuracy. Since the system achieved a recognition performance of 85.3% unassisted by context (lower than some of the implementations using FMSs and HMMs), it is likely the accuracy of the method being used is largely dependent on the particular implementation. Using contextual clues, the system was able to improve performance to 91%, netting about 6% increase in accuracy. This thesis explores if similar improvements can be achieved when using contextual clues with a FSM.

3.3.4 Summary of Related Work

FSMs offer comparable performance to HMMs, but are quicker and more efficient. They do not have to be trained, but must be programmed with statistically proven models[5]. McGlaun et al. compared the accuracy of the FSM and HMM approaches in *Robust Video-Based Recognition of Dynamic Head Gestures in Various Domains-Comparing a Rule-Based and a Stochastic Approach*[16]. This comparison shows HMMs approximately 4% more accurate with the FSMs netting 93.7% accuracy compared to 97.3% for HMMs. This suggests that a FSMs can offer comparable performance while being computationally less demanding. It should be noted that HMMs can be limited by the characteristics of their training set, while the statistically predictive FSMs could possibly lead to more accurate gesture recognition [17]. However, HMMs can more easily be extended to include additional gestures [16] where a FSM must be specifically

programmed for each feature.

The following tables summarize the results presented in the above survey. Table 3.1 summarizes accuracy data including separate data for head nods and head shakes where reported. Table 3.2 summarizes efficiency data if present. Special hardware is reported with a simple *yes* or *no* metric. *Yes* indicates some addition hardware requirement beyond a simple web camera, such as a stereo or IR camera. *No* indicates that the implementation did not use any special hardware besides standard low resolution camera.

Efficiency is measured by frames per second (fps) of the image processing as well as the processing power of the CPU measured in megahertz. This metric does not take into consideration actual CPU load or differences between processor architectures, operating systems, or any other differences. However, one can still infer some sense of efficiency from the this metric.

Missing from these metrics is a breakdown of true positive and false positive rates. These results were disappointingly omitted from most of the surveyed work. A dash indicates an unreported or inappropriate metric. Tables 3.1 and 3.2 shows a high-level comparison of the results. Please see the individual studies for more detailed results.

[1]These results include accuracy for an additional type of "neutral gestures" which were much lower than the accuracy of nods and shakes. Hence the much lower total accuracy.

Implementation	Accuracy(%)			Special
	Shakes	Nods	Average	Hardware
Eye Location with FSM[2]	96.7	93.3	87.8[1]	No
Eye Location with 2 HMMs[2]	96.7	96.7	88.9[1]	No
Eye Location with 3 HMMs[2]	86.6	93.3	85.6[1]	No
IR Camera with HMM[8]	75.0	81.08	78.46	Yes
Circle Filter with FSM[9]	-	-	86	No
Color Filter FSM[16]	-	-	93.7	No
Color Filter HMM[16]	-	-	97.3	No
SVM with Stereo Vision and Context[18]	-	91	91	Yes
SVM with Stereo Vision and Context II[19]	100	90	95	Yes
SVM with Stereo Vision and Context III[20]	98	93	95.5	Yes
Color Filter HMM[22]	-	-	87	No
Kanade-Lucas-Tomasi FSM[5]	96.4	95	95.7	No
Multi-view Model HMM[14]	85.2	90.6	88.1	No

Table 3.1: Summary of Accuracy of Related Work

Implementation	Efficiency(%)		Special
	CPU(MHz)	FPS	Hardware
Eye Location with FSM[2]	200	-	No
Eye Location with 2 HMMs[2]	200	-	No
Eye Location with 3 HMMs[2]	200	-	No
Multimodal Tracking[14]	150	20	No
PupilCam with FSM[4]	1000	30	Yes
IR Camera with FSM[8]	933	30	Yes
Circle Filter with FSM[9]	175	13	No
Stereo Vision[10]	866	30	Yes
SVM with Stereo Vision and Context[18]	-	18	Yes
SVM with Stereo Vision and Context II[19]	-	18	Yes
SVM with Stereo Vision and Context III[20]	-	18	Yes
Color Filter and HMM[22]	-	20	No
Multi-view Model HMM[14]	1300	-	No

Table 3.2: Summary of Efficiency of Related Work

Chapter 4

Design

4.1 Assumptions

For the design of this project, I make several assumptions that help simplify the design of the system and maintain a high accuracy. These include assumptions about the user's position, orientation, distance from camera, lighting, etc. The following is a list of the major assumptions made for this project.

- The user will be positioned in front of the camera and remain as such throughout use of the system.
- The user will oriented towards the camera during the entire use of the system, and will not turn their head more than 90°.
- The environment lighting will consist of sufficient daylight to illuminate the user.
- The environment lighting will not dramatically change abruptly, such as turning all the lights on or off, although slight changes are expected.

- The user's face will not be totally obscured or leave the image during use of the system.
- The user will not have any significantly obscured or missing facial features. Glasses and facial hair do not significantly obscure facial features.
- The user will have a light skin tone.
- The user will be within five feet of the camera, but no closer than one foot.
- There will only be one user in the frame at any one time.

I believe many of these assumptions are valid for the intended use of the system. Normal use would consist of a single user sitting in front of the system oriented towards it in a well lit room during use. Situations with more than one user would require modification. The assumptions about maintaining constant unobscured view of the user throughout the use of the system is not practical for a real world system, but are made here to simplify the system so that automatic recalibration of the facial position and error handling of the system can be ignored as they are not the focus of this thesis. Since the system is able to automatically acquire users at initialization, reacquisition should require little effort. Assumptions about the user are made solely due to a lack of diversity in the pool of test subjects.

The above assumptions were key in the design of the system. Using the assumptions about the general location and orientation of the user, we can make intelligent guesses about the most likely position of the face. This helps determine which feature recognition techniques to use. The assumptions about the user's environment help determine the best point tracking algorithm to use.

4.2 Design Overview

The following chapter presents the design of the core head gesture recognition library used to gather results. This library is used later in two test prototypes described in Chapter 5. The design consists of four main components: Feature Detection, Feature Tracking, Motion Tracking, and Gesture Recognition. The Feature Detector is used to determine the initial position of the face. Once the face has been identified, the Feature Tracker tracks various points on the face between frames. The Motion Tracker analyzes the motion of points between two frames and determines the overall movement of the head. The Gesture Recognizer analyzes head motion over time and determines if the user has performed a gesture.

For Feature Detection, this design uses the Haar Cascade Classifier algorithm, described in Section 3.2.7, due to it's efficiency and accuracy. I also considered Blink Detection (see section 3.2.1) and Circle-Filters (see section 3.2.6), but I determined that these methods were not reliable enough. Eye Detection required users to blink, which may not occur in a timely manner. Answering dialog boxes require quick detection since a user is likely to answer the dialog box within a few seconds of it being displayed. Circle-Filters are an intriguing method, but are less sophisticated than Haar Classifier Cascades and suffer from similar drawbacks[9].

For Feature Tracking, I use the Lucas-Kanade algorithm, described in section 3.2.8 to determine points within the face region to track. Experiments with the Color-of-Skin Filter showed that it was not accurate enough to track small head gestures. Although I developed an implementation of a Color-of-Skin Filter, it would often loose accuracy over time as the color template was updated, and background objects would often confuse it.

The motion tracker uses simple statistical analysis for determining the motion of the head. The possible motions include Left, Right, Up, Down, and Center. Since the Lucas-Kanade algorithm is so sensitive, the motion must exceed a specified threshold to eliminate false positives from small movements.

The Gesture Recognizer consists of a simple FSM. This FSM consists of eleven states: Left1, Left2, Right1, Right2, Up1, Up2, Down1, Down2, Nod, Shake, and None. The FSM starts in the None position. When a motion is made, the FSM moves to the first level of the corresponding state in the FSM. A motion in the opposite direction move the FSM to the second state of that direction. A motion back in the original direction moves the FSM to the corresponding gesture state. The FSM then moves back to the None state. Any incorrect gesture during this process moves the FSM to the None state as well. A full diagram of the FSM is depicted in Figure 4.1.

I developed two programs for testing. These test programs trigger dialog boxes at random time intervals that must be answered with head gestures. One test program uses only the base gesture recognition system to answer dialog boxes. The second adds contextual clues to trigger gesture detection only when a dialog box is active. This consists of simply initializing the tracker right before the dialog box is displayed and stopping tracking after a gesture is detected. An in-depth analysis of optimal contextual clues is beyond the scope of this paper. For this thesis, I limit the contextual clues optimization to only detect whether or not a dialog box is active.

After the primary study and evaluation, I implemented a short optimization to the gesture FSM to expand the types of head and nod gestures it recognizes. Additionally, a few minor tweaks were made to several parameters to increase accuracy.

4.3 Feature Recognition

I chose to use a collaboration of Haar Classifier Cascades and the Lucas-Kanade algorithm to perform my feature detection and tracking. The Haar Classifier Cascade detects facial features well, but is not accurate when it comes to tracking rotation and other subtle head movements. My early experiments determined that Haar Classifiers Cascades did not provide the accuracy necessary for tracking subtle head movements. I attempted to incorporate skin color filtering, using the region detected by the Haar Classifier Cascade as a template. This, too, failed to provide the accuracy needed for head gesture detection. However, after experimenting with the Lucas-Kanade algorithm, I found it proficiently and accurately tracked a series of small regions on the face.

The Lucas-Kanade method lacks a method for determining which regions to track initially. I combine the Haar Classifier Cascade and Lucas-Kanade algorithms to achieve my feature tracking. The Haar Classifier determines the face region, after which the Lucas-Kanade algorithm precisely tracks a number of points set in that region.

4.4 Motion Tracking

Motion tracking is implemented using statistical analysis of the distance between tracking points on the x and y axises in each subsequent frame. Head momentum between two frames can be measured as up, down, left, right, or center. Extra effort is not made to determine diagonal movement, instead relying on the greater of x or y axis movement to determine direction. Motion over time changes state in the FSM. When the FSM reaches a full gesture state, an event

is triggered and the finite state machine is reset.

4.5 Gesture Recognition

Gesture recognition consists of two gestures: nods and shakes. A nod is defined as a series of three consecutive motions in an up-down-up or down-up-down configuration. These motions must be consecutive, but allow center or "null" motions between motions. A motion of left or right voids the gesture. Similarly, a shake gesture is defined as a series of three movements in a left-right-left or right-left-right configuration with up or down motions voiding the gesture. A gesture must be contained within forty frames to be considered valid. Gestures longer than forty frames are reset to help eliminate false positives generated by slow movements over long periods of time.

This thesis limits gesture detection to these two gestures. It is possible to add more gestures to the recognition algorithm, but findings in *Design Issues for Vision-based Computer Interaction Systems* [13] and *Head Gestures for Computer Control* [12] suggest that, while gestures such as nodding and shaking one's head is intuitive for some actions, gestures such as tilting one's head is uncomfortable and should be limited if used at all. Research suggests that nodding and shaking one's head is an intuitive interface for answering dialog boxes, but does not provide an intuitive interface for document browsing[18, 20].

Detection of the head gestures use a simple FSM which chosen for its simplicity and efficiency. This FSM has eleven states and detects nods and shakes. Please see Figure 4.1 on 30 for the detailed design of the finite state machine.

Figure 4.2 shows a modified FSM used in the secondary study described in

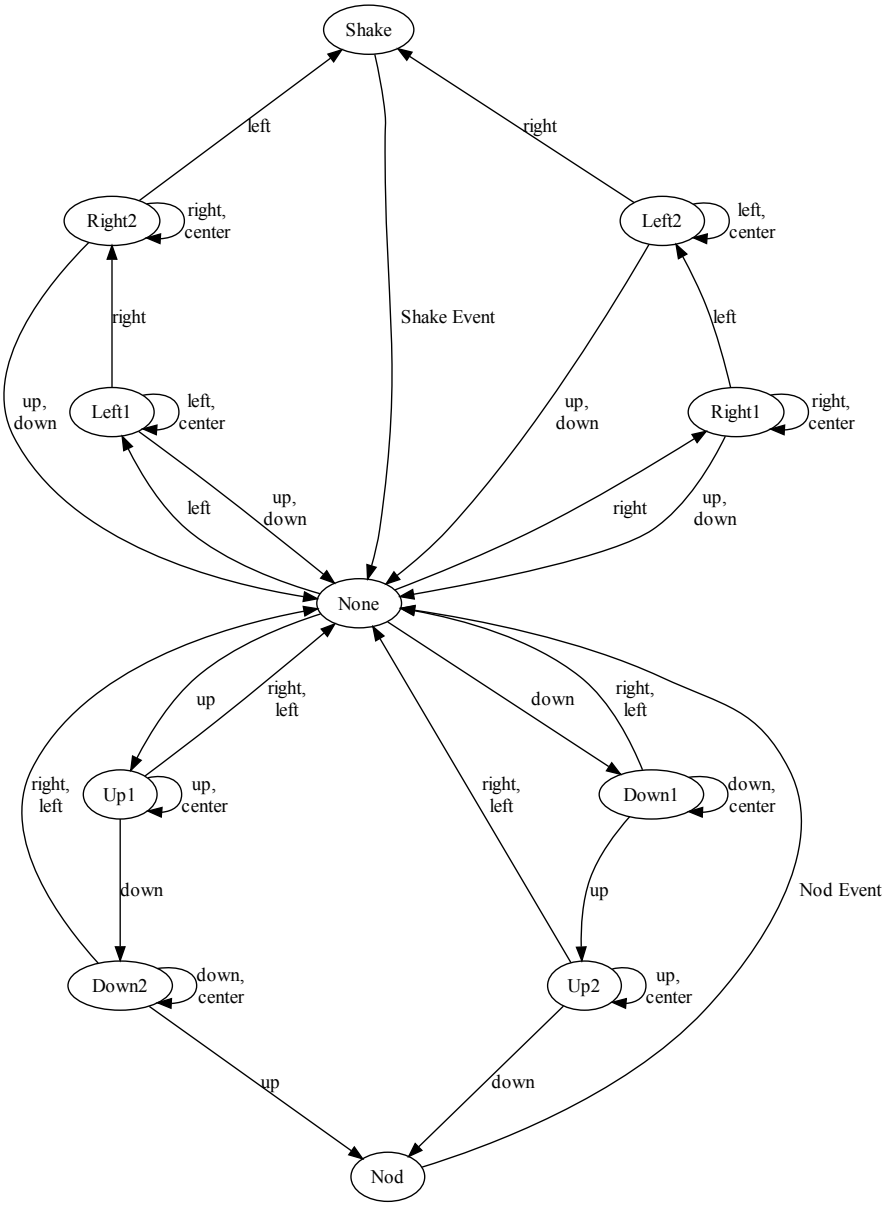

Figure 4.1: Head Gesture FSM

Chapter 6. This FSM is very similar to the first one, but has additional paths to the final nod and shake gesture states. This allows for single nod and shake gestures such as down-up-center or left-right-center.

4.6 Prototype Design

Two prototypes using this library were developed: NaiveGestures and ContextGestures. Both prototypes are simple test programs that randomly generate dialog boxes and dismiss those dialog boxes with head gestures. NaiveGestures uses the core head gesture library to constantly track head gestures and trigger events when they are detected. ContextGestures only initializes head tracking when a dialog box is triggered and stops tracking after it is dismissed. In this way, ContextGestures seeks to reduce false positives by only monitoring head gestures when they are relevant, i.e. when a dialog box is on screen. The following sections describe the design of the head gesture recognition library and the two test programs.

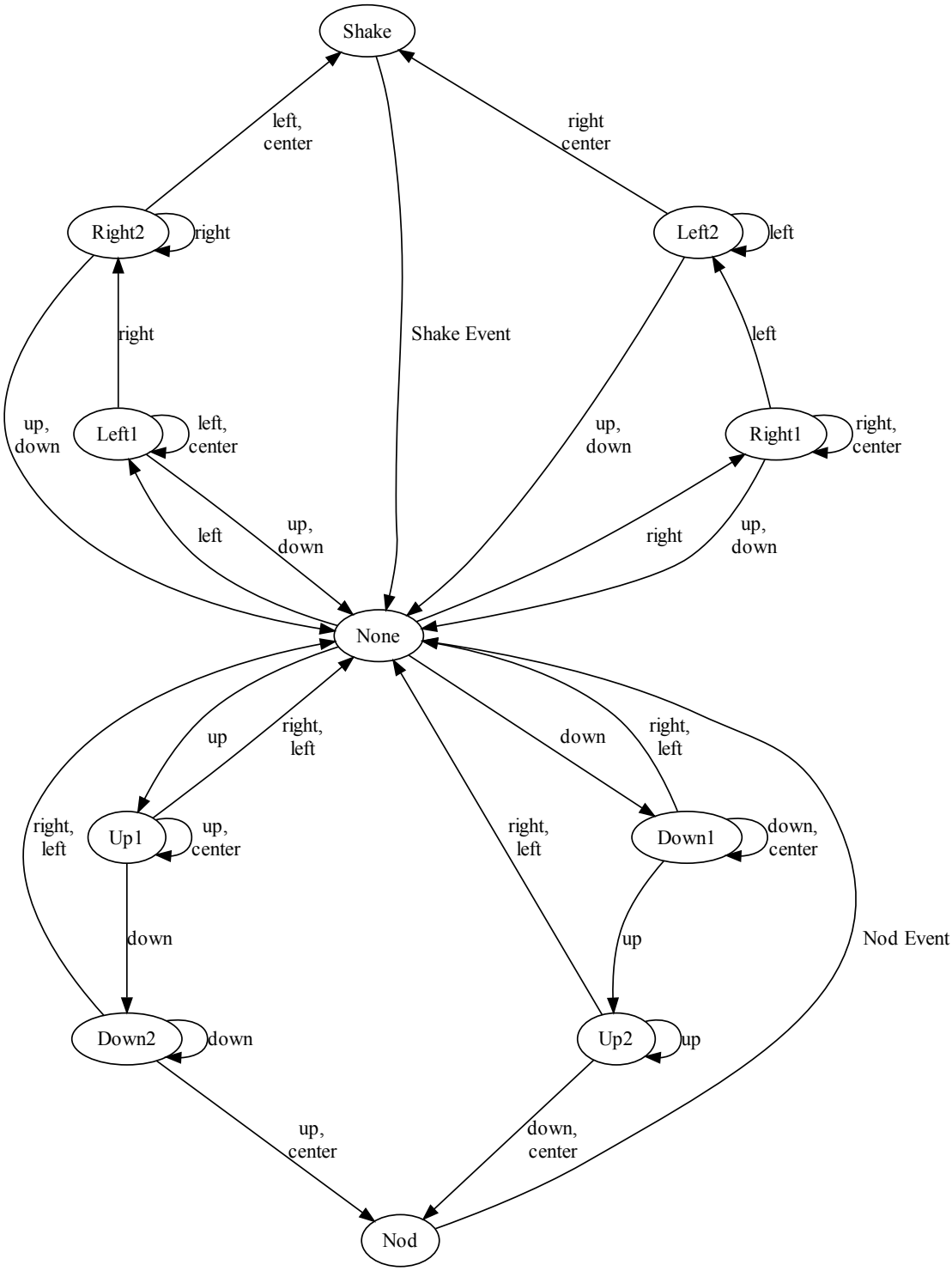

Figure 4.2: FSM with Expanded Head Gestures

Chapter 5

Implementation

5.1 Implementation Overview

The prototype project is implemented in C and C++ using the OpenCV library. Development took place on Microsoft Windows platform using Microsoft Visual Studio 2008. The system uses a simple web camera with a 320x240 resolution for image capturing.

The Haar Classifier Cascade and Lucas-Kanade algorithms are used in conjunction for feature detection and point tracking. OpenCV handled most of the low level image processing and simplified the implementation of these algorithms. The Haar Classifier Cascade provides the primary feature detection method. The Haar Classifier Cascade does not have the consistency to reliably identify the same features smoothly between frames even while moving or through lighting changes. The Haar Classifier Cascade provides an excellent way of identifying the most likely area containing a face, but experimentation showed that the Haar Classifier Cascade often cannot smoothly track head movement between frames

with the accuracy necessary for head gesture recognition. The Haar Classifier also tends to lose tracking at extreme angles such as the peak of a head nod or shake. Additionally, processing the entire image for the location of the face every frame using a Haar Classifier Cascade is costly resource wise and is not very efficient. Large unimportant areas of the image must be repeatedly processed despite having already determined the location of the face previously. To address the consistency and efficiency concerns, the Lucas-Kanade algorithm is used as a point tracker.

A Color-of-Skin filter was also considered for tracking purposes. While the Color-of-Skin Filter is a very simple and efficient algorithm, it is not very accurate nor robust. Additionally, it alone cannot identify facial features and must be calibrated first. To accomplish this, the Haar Cascade Classifier is used to identify a region of skin. This region is then passed to the skin filter for calibration. This allows easier tracking of the skin region. However, there are many inaccuracies to the algorithm. Changes in the environment and especially lighting can completely destroy the accuracy of the skin filter. This detection method is also not very good at detecting specific points on the face and will often include elements such as arms or skin colored elements in the background. To address changes in the environment, the skin filter can be continually self calibrating, passing the newly detected region back into its calibration to account for gradual changes in lighting. However, since the detected area usually contains some elements of the background, noise is often included in these calibrations leading to less and less accurate calibrations until accuracy is greatly diminished if not lost completely. Due to the skin tracker's inadequacies, it was abandoned in favor of the Lucas-Kanade method for head tracking.

The Lucas-Kanade algorithm is an accurate and robust feature tracking al-

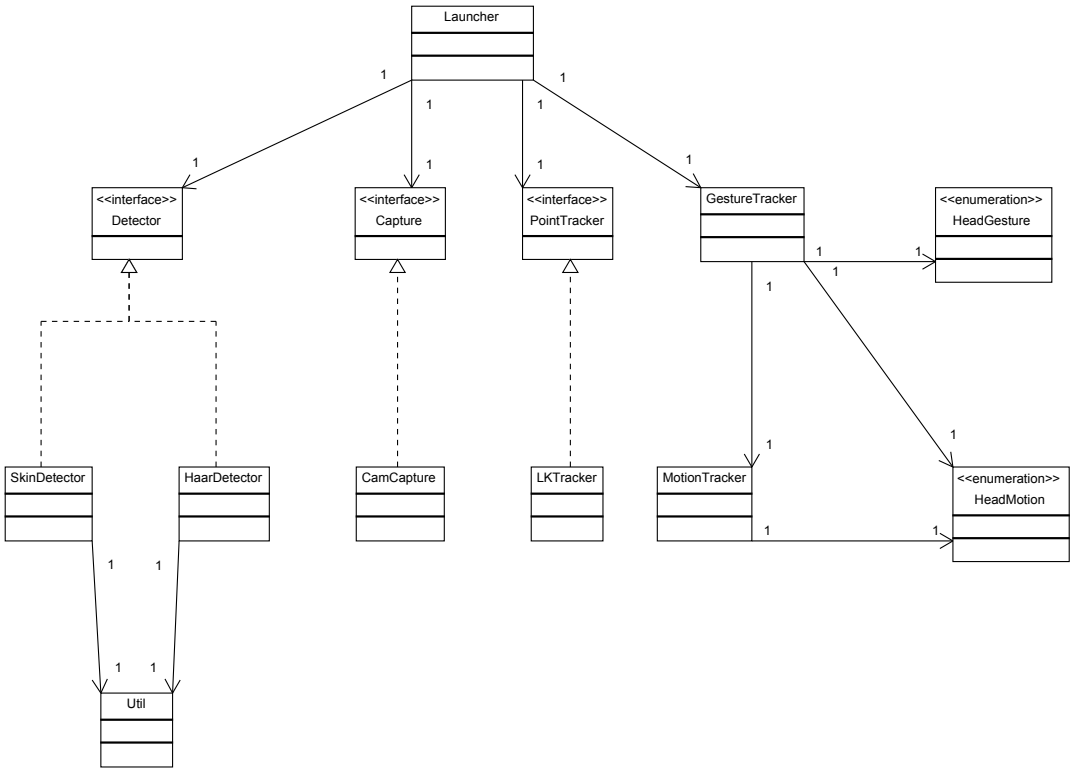

Figure 5.1: Diagram of Class Dependencies.

gorithm. While unable to identify facial features itself, it can accurately track a region to subpixel accuracy once the feature has been identified. This method proves to be very robust and can even track features that are temporarily lost or obscured as long as the region isn't covered too much.

Initially the Haar Tracker identifies the general location of the face for tracking. This region is passed to a method the determines a number of points on the face to track. These points are then passed to the LK Tracker for tracking.

5.2 Software Process

Some software practices from the Extreme Programming (XP) agile software process were adopted for use in this project. Many of the practices were not applicable due to the fact that no team management was necessary since I was the sole developer and client. Regrettably, unit testing was neglected due to the highly visual nature of the project. Testing consisted of manual system tests using debugging console or video output while interacting with the system. During development, I used Users Stories to guide several release iterations. Releases aimed at satisfying the most important immediate requirements.

The first release was a proof of concept side-by-side comparison of the Haar Classifier Cascade and Color-of-Skin Filter algorithms. These algorithms operated separately, and the face region had to be manually selected with the mouse to begin tracking with the Color-of-Skin Filter. No motion detection was performed. The second release added the LK Tracker in addition to motion tracking capabilities. The third release included gesture recognition and also dropped the Color-of-Skin Filter. This was the release used for the majority of testing. The final release for this project included some tweaks to various tracking parameters and a modification to the FSM to recognize several variations of the nod and shake head gestures.

The project used Subversion (SVN) for revision control. Care was taken for proper SVN repository management, including proper commit logging and tagging of releases. The project consists of four major releases with more than one hundred commits to the SVN repository and over 1200 lines of code.

I used an academic license of Visual Studio 2008 Professional for development. The software was developed for the Microsoft Windows platform and tested on

Windows XP and Windows 7 x64. However, care was taken to make the software cross-platform, and the head gesture recognition library should be portable to the Linux platform with relatively minor effort.

5.3 OpenCV

I use OpenCV for much of the image processing in the head gesture recognition library. OpenCV is a computer vision library originally developed by Intel. It is free for both academic and commercial use under the open source BSD license. The library is cross-platform and focuses mainly on real-time image processing[1].

The OpenCV library provides implementations of the Viola-Jones detector[30] (referred in this paper as the Haar Classifier Detector) as the Lucas-Kanade algorithm[1]. OpenCV also includes several sample trained cascades for the Haar Classifier. The focus of this project is not on training Haar Classifiers so a sample classifier was used.

Using the OpenCV library allowed for rapid development of a head gesture recognition system. The implementations in the OpenCV library are robust and efficient. The work presented here to focus on implementing and evaluating a functional head gesture recognition system that incorporates contextual clues.

5.4 System Architecture

Figures 5.5 and 5.6 show detailed class diagrams of the head gesture recognition library. The system has four main components. The Feature Detector, the Feature Tracker, the Motion Tracker, and the Gesture Tracker. These compo-

nents are realized by the HaarDetector, LKTracker, MotionTracker, and GestureTracker classes respectively. Other components in the system include the CamCapture, Launcher, SkinDetector, and Utils classes. Additionally, the system uses an event based system for generating and receiving gestures which include the GestureEvent, NodReceiver, and ShakeReceiver classes. The head gesture tracking functionality is contained within a static library. The test prototypes link against this library and register event handlers for the gesture events generated by the core library. The following sections describe the architecture in detail.

5.4.1 CamCapture

The CamCapture class realizes the Capture interface and is the component used to capture video from a camera. The Capture interface defines a single method `getFrame()` which returns an image. The CamCapture class returns a frame from the first attached camera device detected. It also contains logic for calculating and displaying the time and frames per second, recording video, and handling situations where no camera is attached to the system.

5.4.2 HaarDetector

The HaarDetector class implements the Detector interface and provides the primary method for detecting the facial features. The Detector class provides two main methods: `detect(image)` and `select(rectangle)`. The detect method returns a rectangular region where the face is likely to be located. The select method is used to pass the Detector a region of interest, and is mostly unused with the HaarDetector.

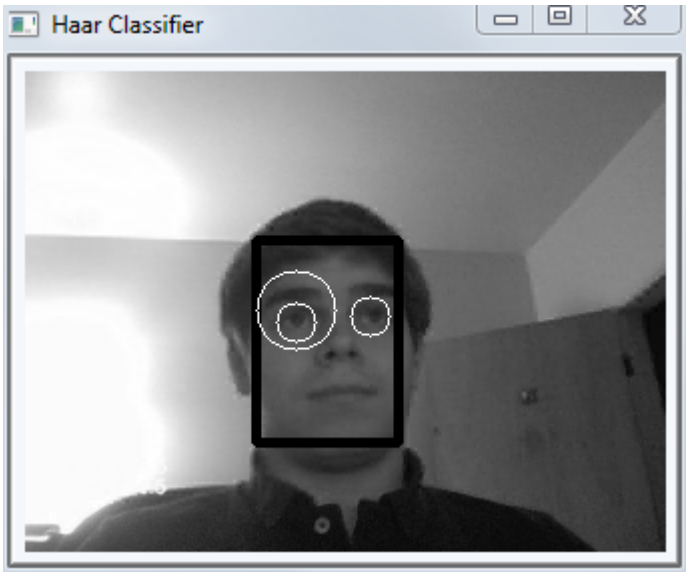

Figure 5.2: Haar Feature Detection

The HaarDetector loads a sample cascade from an external XML file at initialization. The detect method detects the rectangular region containing the face if one is detected. Multiple faces can be detected, but only the first is used. If no face is detected, it returns null. The HaarDetector also has the ability to detect sub-features such as the eyes. This proved unnecessary in the final prototype system. Figure 5.4.2 shows an example feature detection with eye sub-feature detection turned on.

5.4.3 SkinDetector

The SkinDetector class is an alternate implementation of the Detector interface that uses selected skin colors to detect facial features. The select method would pass in a region of interest to use as a color histogram for skin detection.

Early prototypes used the SkinDetector in conjunction with the HaarDetector to try and improve accuracy. The region of interest generated from the HaarDe-

tector would be passed to the more efficient SkinDetector for tracking. The SkinDetector proved too inaccurate and unreliable for tracking head gestures and was eventually excluded completely from the final prototype. The possibility still remains to use the SkinDetector to attempt to improve robustness of the current system.

5.4.4 LKTracker

The LKTracker class implements the PointTracker interface. The primary functions the LKTracker are the `detect(image)`, `select(rectangle)`,`getPoints()`, and `getNumPoints()` methods. The detect method does nothing until a region of interest is selected with the select method. Once a region is selected, the LKTracker is able to determine some tracking points within the selected area. Subsequent calls to detect will return an array of points corresponding to the new position of those points in the given image. The getPoints method returns a two dimensional array. This array contains an array of the positions of the tracking points in the previous frame as well as an array of the corresponding positions of the tracking points in the current frame. Figure 5.3 shows an example of points tracked by the LKTracker.

The LKTracker is very efficient, but lacks the ability to detect features on its own. Although the technique is fairly robust dealing with rotation and temporary obstruction or loss of a tracking point, a more drastic loss of a tracking point, such as the face being obscured by another object such as an arm, may result in loosing one or more tracking points. Ideally, a secondary system would provide a fall back mechanism for reestablishing tracking points if they are lost. For the purpose of this research, such a system was not necessary.

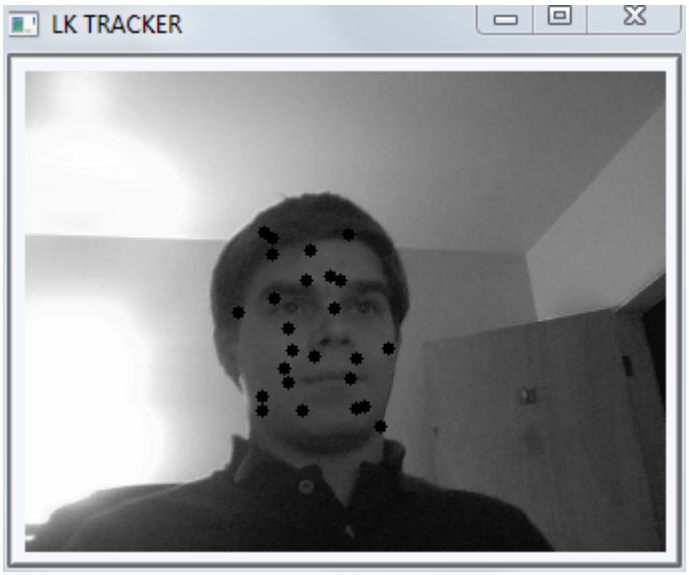

Figure 5.3: Lucas-Kanade Algorithm Tracking Points

5.4.5 MotionTracker

The MotionTracker class contains the `detect()` and `getMotion()` methods. The detect method uses takes a two dimensional array generated from the LK-Tracker as an argument. It compare every element between the two arrays to determine change in position. It determines the direction of motion by taking the greater of the x or y-axis. Possible motions include CENTER, RIGHT, LEFT, UP, and DOWN.

To eliminate false positives from small movements, the distance must exceed a specified threshold. If not, the motion is counted as a CENTER motion. The threshold was determined empirically during development on a single user and should not be considered optimized.

The overall direction of motion was determined by taking the sum distance of each tracking point that moved in that corresponding direction. Each CENTER movement added the value of the threshold as the value of each point. The motion

with the greatest value is then set as the overall head motion.

5.4.6 GestureTracker

The GestureTracker class contains the FSM representation of the current gesture. This class contains an instance of the MotionTracker class. The `track()` method passes the array of tracking point locations to the MotionTracker, along with the number of expected points. It returns the current state of the FSM. A detailed diagram of the FSM used is depicted in Figure 3.3.1.

5.4.7 Launcher

The launcher class controls the flow of execution between components. It is the entry point to the head gesture library. It also exposes the Gesture Events. These events can be hooked to external gesture receivers to perform different functions. The basic flow of execution is depicted in Figure 5.4. The component responsible for each process is labeled in parenthesis.

The flow of execution starts when the `run()` method is called. The first step after initialization is to get a frame from the camera. After a frame has been queried, the Launcher checks if a face region has been detected. If not, the Launcher class queries another frame and repeats the process until a face has been detected. If a face is detected, its location is stored and passed to the LKTracker. The LKTracker will then determine the location of the face in all subsequent calls, updating the face position while it does so. The position of the face is then passed to the MotionTracker, where the general motion between two frames is detected. Finally, the motion moves to the GestureTracker, which updates its FSM to determine if a nod or shake gesture has been generated. The

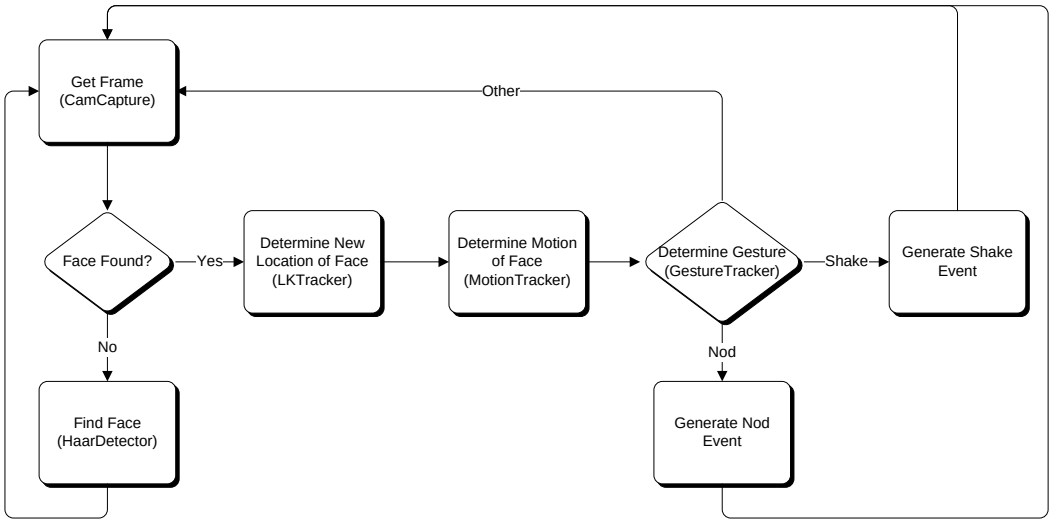

Figure 5.4: Flow of Execution of the Launcher Class

Launcher class checks the gesture returned by the GestureTracker and triggers a Nod or Shake GestureEvent if the corresponding gesture is detected. This process repeats until it is terminated.

5.4.8 Test Prototypes

The two test prototypes, NaiveGestures and ContextGestures, are relatively simple. They mainly consist of a simple wrapper for the Launcher class found in the head gesture recognition library. In addition, some logic for randomly generating dialog boxes is added, including threading to run the dialog boxes and the gesture recognition concurrently.

The main difference between NaiveGestures and ContextGestures is that ContextGestures contained an extra thread to control starting and stopping of the Launcher class based off status of the dialog box thread. This is what serves as the contextual clue optimization for this research. Besides this difference, NaiveG-

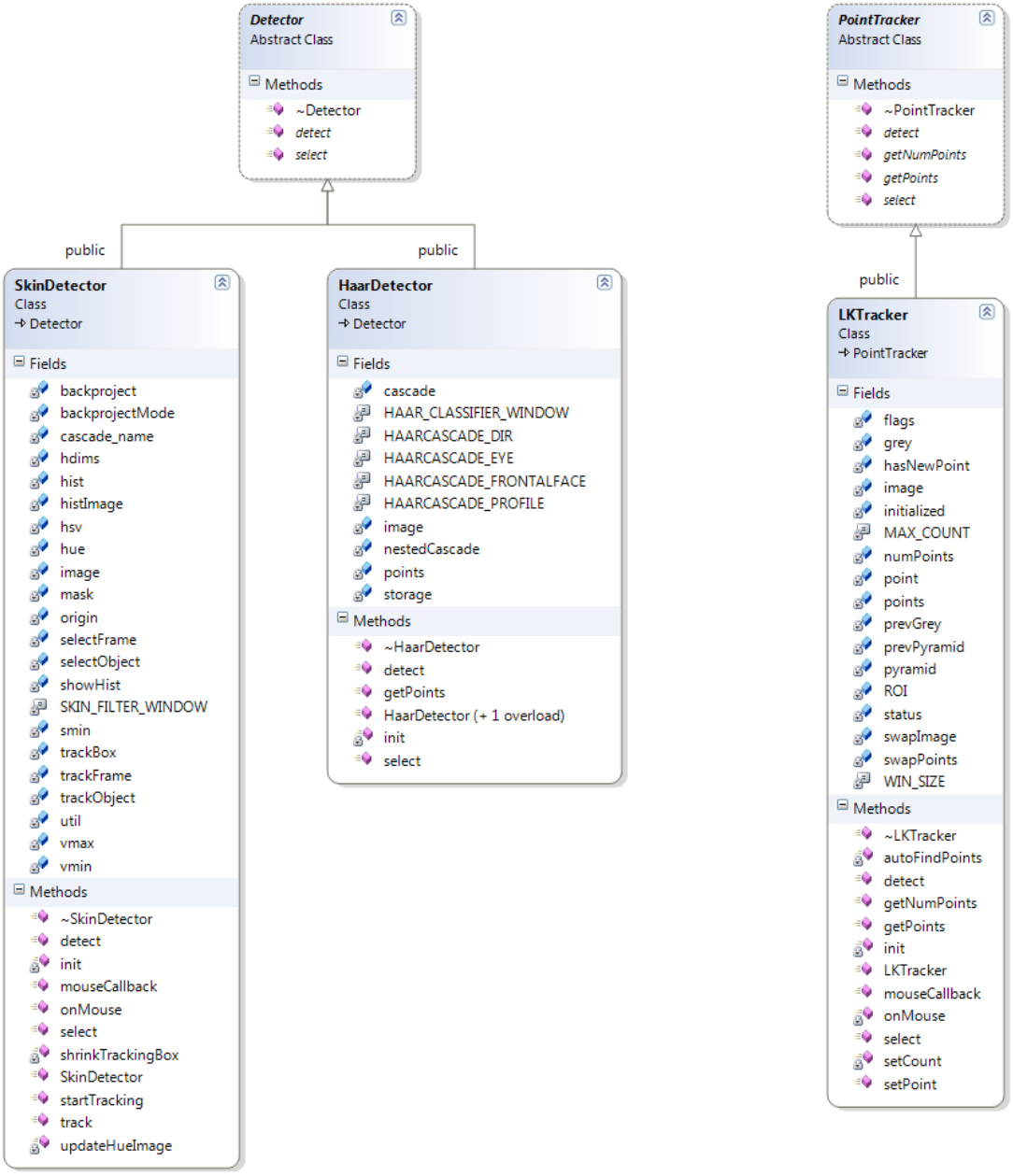

Figure 5.5: This Diagram Shows a Detailed Class Diagram of the Gesture Recognition Library.

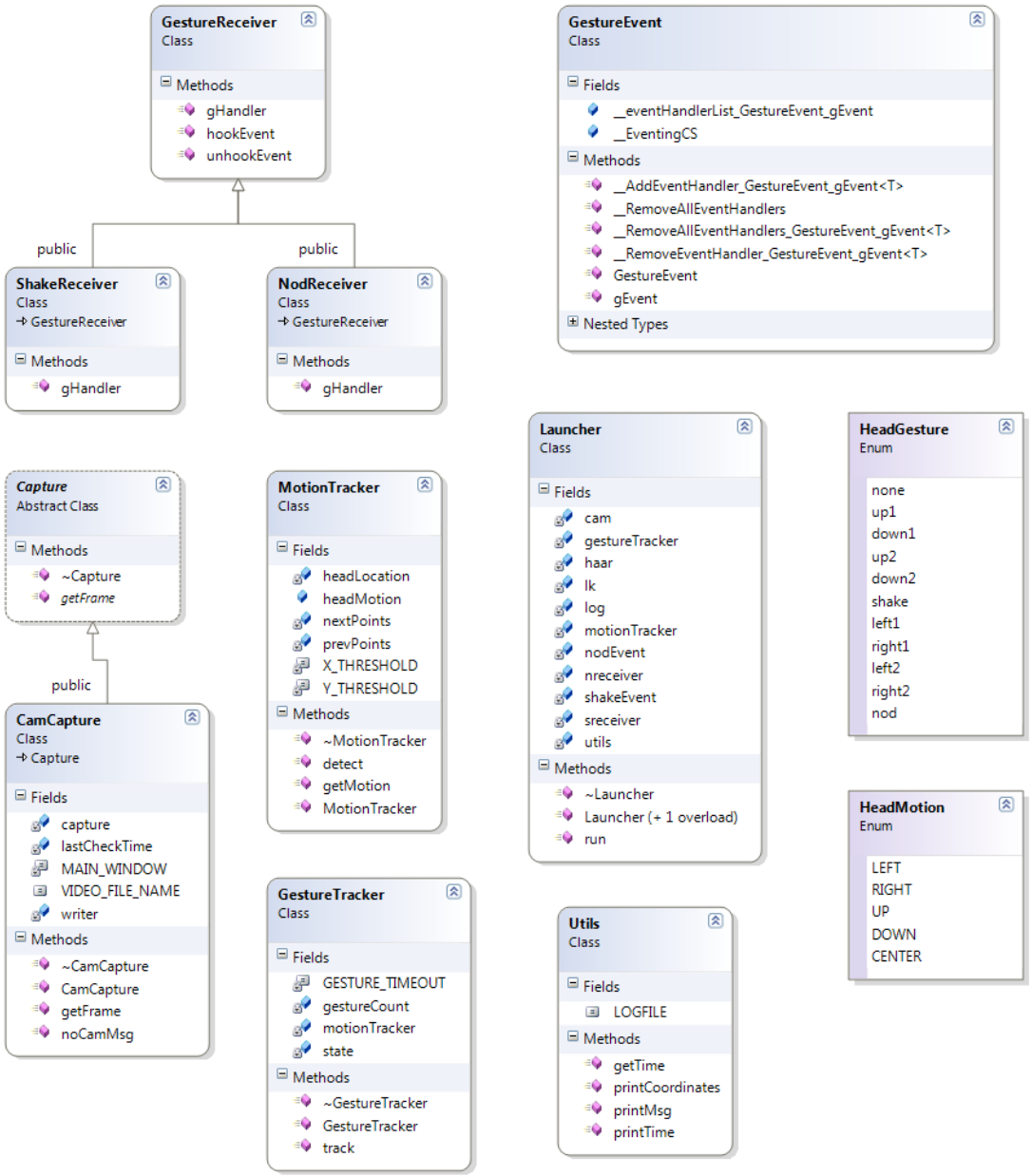

Figure 5.6: This Diagram Shows a Detailed Class Diagram of the Gesture Recognition Library.

estures and ContextGestures shared most of their code and used the same techniques for identifying head gestures. This is what makes the comparison between them significant because the contextual clues optimization should be the only relevant difference. There is no apparent difference between the two programs as the user is running either of them.

Chapter 6

Evaluation

6.1 Experimental Setup

After the implementation was completed, I conducted a several user studies with the finished system. The studies consisted of a pilot study, a primary study, and a secondary study using a modified prototype to address some issues noticed in the primary study. Additionally each user was asked to fill out a short usability survey on the test prototypes. The goal of the study was to compare accuracy of the two test prototypes NaiveGestures and ContextGestures. Results were logged and manually recorded throughout each user trial in addition to data programatically collected by the test prototypes themselves.

Details about the experimental subjects, environment, and procedure are given in sections 6.1.1, 6.1.2, 6.1.3 respectively. The results of the study is presented in section 6.2 and the results of the usability study are presented in section 6.5.

6.1.1 Experiment Subjects

The user test pool consists of randomly pulled students from CPE 581 - Computer Support for Knowledge Management, taught by Professor Franz Kurfess. The students were all male and had significant technical background. Users varied in facial features including facial hair and some test subjects wore glasses. All test subjects had relatively light skin.

Because of the low diversity in the experimental subjects, it is difficult to draw conclusions to a wider general user base. It is possible the significant technical background common to all test users improved their ability to adapt to the system. Alternatively, it's possible that the high amount of training using conventional forms of computer input actually made the user less able to learn a new interface that would be familiar with subjects without as much technical background. The results of the usability study are particularly questionable since it pulls for a very specific user group and does not accurately represent most users.

For the secondary study, I attempted to address this by selecting more diverse users including some outside the computer science department. Despite this, the user pool is still relatively narrow, but does include more variation in technical background and typing proficiency.

6.1.2 Experiment Environment

The experiments took place in a controlled environment in room 14-238B on the California Polytechnic State University campus in San Luis Obispo. The test machine consisted of an Asus EEE 1000h netbook with an Hercules Optiplex Glass Webcam. The netbook was chosen as the preferred platform for its lim-

ited input methods and reduced performance over most desktop machines. This helped simulate the target environment where a user may benefit from a head gesture recognition system. A mouse was not available to the test subjects during the experiment, although they were allowed to use the track pad and keyboard to navigate if desired. The netbook has a 1.6 GHz Intel Atom processor. The webcam has a resolution of 320x240 in color.

For the secondary study, a wireless mouse and keyboard were added to prevent the user form directly interacting with the netbook. This is to address a problem with users shaking the camera which is explained in section 6.3. The experiments were also performed at a different location.

6.1.3 Procedure

The experiment is divided into two sections: a short pilot study, and the actual study. The pilot study includes several unrecorded trials outside of the official environment, and two trials officially recorded in the official environment. Adjustments were made during this period to adapt to the environment and establish a consistent experimental procedure. The second study consists of seven different subjects and constitutes the official test results.

Each trial was personally supervised. Each user received a brief introduction to the system and a short tutorial on how to use it at the start of each trial. The user would perform two tasks: a reading task and a writing task. At the start of each task, the user would start one of the test prototypes. The system would present the user dialog boxes which the user would respond to with head gestures. Figure 6.1 shows an example of one such dialog box.

The procedure for the secondary study was slightly different in that the user

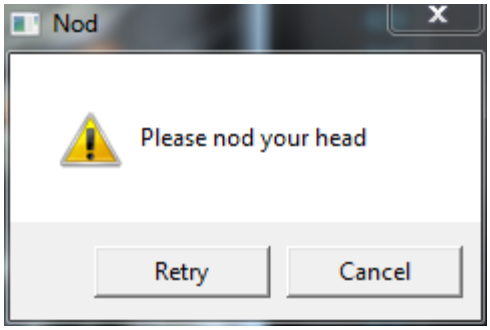

Figure 6.1: This dialog box prompts a user to nod their head.

was not instructed on which head gestures to use, and instead used whichever head gestures were most natural for them. This eliminated any training needed to use the system.

The system would collect data on ten head gestures before notifying the user and then terminating. At this point, the user would start the next prototype and continue with the current task. One both prototypes had collected data, the user would move on to the next task.

To avoid bias, NaiveGestures and ContextGestures were given the codenames *Yellow* and *Purple* respectively. Colors were chosen because they are order agnostic, and do not imply one is better than the other. Additionally, the order that the test subjects ran NaiveGestures and ContextGestures was randomized to account for any training effects. The order alternated between subjects and tasks. For example, the first subject would run *Yellow* followed by *Purple* for the first task, then *Purple* followed by *Yellow* for the next task. The next user would then run *Purple* followed by *Yellow* for the first task, then *Yellow* followed by *Purple*. This attempts to negate any training bias from the user learning the system during the test.

The reading task consisted of reading a short one-and-a-half page PDF. First

the subject would start one of the test prototypes. After a notification that the system was running, the student would then open the PDF and begin reading. They were allowed to navigate the document with the keyboard or touchpad. Once the test program had recorded enough data, it would notify the user it was finished, and the user would be instructed to start the second program and continue reading. If the user finished the document before the test program had finished recording data, they were asked to start reading from the beginning of the document again.

Once data for the reading task was collected by each program, the users were instructed to move onto the writing task. This task consisted of opening Word Pad and copying some sample text off a piece of paper laid to the right of the keyboard. Like the first task, the user was asked to start one of the test programs before the beginning the task. When the program notified the user that it had finished collecting data, the user was instructed to stop their typing, start the next prototype, and resume typing.

Both tasks were designed to simulate situations where false positives would be common. The reading task simulates a everyday situation of reading and browsing a document. The writing task purposely forces the user to move their head by making them read an external document. The user must move their head to look at the document, then back at the screen, and in some cases at the keyboard.

Each test was manually observed and recorded. When the system failed to recognize a head gesture, the user was instructed to state "Miss", which was manually recorded along with the gesture attempted. The user would then try again until the gesture was recognized, or in rare cases, the program restarted. If a dialog box was dismissed without the user intentionally performing a gesture,

they were instructed to state "False Positive" and the false positive was recorded. If the user performed the wrong gesture, they were instructed to say "Error". Comparing this data to the data logged by the program, the number of hits, misses, and false positives was calculated. Errors were manually verified and subtracted from the false positive rate.

6.2 Results

The results of the study are presented in Tables 6.1, 6.2, and 6.3. The results are divided by program, then task, then gesture type. A *Hit* is defined as any gesture that was successfully detected by the test system. A *Miss* is defined as each correct gesture the subject performed that was not detected by the test system. A *False Positive* is defined as a gesture that was detected by the system without the user's intention. These include gestures between dialog boxes and incorrect gestures detected during a dialog box that was not an error. An *Error* is defined as an intentional but incorrect gesture given by the user. Errors are used to distinguish between genuine False Positives and simple user error. Accuracy is defined as the percentage of Hits out of the sum of all Hits, Misses, and False Positives. Errors are not included in this calculation. Table 6.4 shows the percentage of false positives out of all gestures for the primary study. Table 6.5 shows the rate of misses out of expected gestures (not including false positives).

6.3 Discussion

As one can see from the results that the ContextGestures prototype performed better than the NaiveGestures prototype consistently through all studies. The

	# of Gestures	Hit	Miss	False Positive	Error	Accuracy
Naive Read Nod	19	11	5	3	0	58%
Naive Read Shake	13	9	2	2	2	69%
Naive Read All	32	20	7	5	2	63%
Naive Write Nod	142	12	3	127	7	8%
Naive Write Shake	47	8	2	37	1	17%
Naive Write All	189	20	5	164	8	11%
Context Read Nod	17	10	5	2	0	59%
Context Read Shake	16	10	5	1	0	63%
Context Read All	33	20	10	3	0	61%
Context Write Nod	12	11	1	0	1	92%
Context Write Shake	11	9	2	0	0	82%
Context Write All	23	20	3	0	0	87%

Table 6.1: Pilot Study Results

	# of Gestures	Hit	Miss	False Positive	Error	Accuracy
Naive Read Nod	40	28	5	7	9	70%
Naive Read Shake	58	39	4	15	0	67%
Naive Read All	98	67	9	22	9	68%
Naive Write Nod	276	36	6	234	4	13%
Naive Write Shake	55	31	2	55	2	56%
Naive Write All	331	67	8	289	6	20%
Context Read Nod	28	27	1	0	4	96%
Context Read Shake	39	38	1	0	0	97%
Context Read All	67	65	2	0	4	97%
Context Write Nod	35	34	1	0	0	97%
Context Write Shake	37	34	3	0	6	92%
Context Write All	72	68	4	0	6	94%

Table 6.2: Primary Study Results

	# of Gestures	Hit	Miss	False Positive	Error	Accuracy
Naive Read Nod	65	17	0	48	0	26%
Naive Read Shake	79	23	0	56	0	29%
Naive Read All	144	40	0	104	0	28%
Naive Write Nod	31	15	1	15	1	48%
Naive Write Shake	167	25	1	141	1	15%
Naive Write All	198	40	2	156	2	20%
Context Read Nod	28	22	3	3	0	79%
Context Read Shake	19	18	0	1	0	95%
Context Read All	47	40	3	4	0	85%
Context Write Nod	19	18	0	1	0	95%
Context Write Shake	22	22	0	0	1	100%
Context Read All	41	40	0	1	1	98%

Table 6.3: Secondary Study Results

Study	Naive Read	Context Read	Naive Write	Context Write
Pilot Study	16%	9%	87%	0%
Primary Study	22%	0%	84%	0%
Secondary Study	68%	8%	77%	2%

Table 6.4: False Positive Rate

Study	Naive Read	Context Read	Naive Write	Context Write
Pilot Study	35%	50%	25%	15%
Primary Study	13%	3%	12%	6%
Secondary Study	0%	8%	5%	0%

Table 6.5: Miss Rate

results in Table 6.4 highlight the effect of contextual clues in reducing false positives. The low number of false positives with ContextGestures clearly indicates that the contextual clues optimization significantly reduces false positives even when faced with large false positive rates.

The high number of false positives in the NaiveGestures writing task is worth examining. Upon review of the video recordings of the results taken by the gesture recognition software, I was able to determine that in a few of the test trials, the user would type so furiously on the keyboard as to shake the entire test unit. Since the camera was mounted on the top of the netbook, this would cause the video to shake up and down, simulating a nod gestures. Indeed, the vast majority of false positives were nod gestures. Most of these false positives were generated by a few select users, with over one hundred of the total false positives being generated by a single user. Because of this, the results may be flawed and should not be considered a fair measure of system accuracy. However, it is important to note that ContextGestures was subjected to the same conditions and still did not generate any false positives. Table 6.4 show the false positive rate of NaiveGestures ans ContextGestures.

In the secondary study, I eliminated the problem with the shaking camera by providing a external keyboard to type on. However, the false positive rate is high not only in the writing task for NaiveGestures, but also the reading task. This was caused by the expanded FSM recognizing more types of gestures. More head movements were recognized as single nod or shake head gestures. Additionally, the user pool was more diverse with some users outside of the Computer Science department and a few that were not fluent typists. This caused higher false positives since the non-fluent typists would move their head more looking between the keyboard, screen, and document. During the writing task, the majority of

false positives were shakes which is expected since the user is looking back and forth from the paper to the screen.

The results for the miss rate are more ambiguous. In some cases the miss rate increased, but mostly it decreased for ContextGestures. This was likely due to the fact ContextGestures would recalibrate between each dialog box event where NaiveGestures would only calibrate once at the beginning of a specific task. Tracking likely degraded during extended use, causing a higher miss rate later on. Ultimately, since both prototypes use the same head gesture recognition algorithms, them to have similar accuracy.

Examining the primary study, we can see that contextual clues boost accuracy by about 30% for the reading task and 75% for the writing task. This suggests that adding contextual clues to a head gesture system can significantly reduce false positives. The exact improvement most likely depends on the head gesture system and the nature of the contextual clues, but one can still expect a significant improvement by using contextual clues.

The secondary study uses a modified prototype designed to recognize more gestures and have a lower miss rate at the cost of generating more false positives. We can see the results for ContextGestures are very similar to the primary study even though we reduced our miss rate and made false positives more likely. In addition, users had less technical background and more likely to generate false positives since they had to look at the keyboard more often. This suggests we can optimize our techniques to have lower miss rates but higher false positive rates and use contextual clues to compensate.

6.4 Possible Sources of Errors

The pilot study showed that different subjects used different head gestures. Some greatly exaggerated their gestures or gestured very slowly. This cause high inaccuracies since the system was designed to pick up small and quick gestures, and the slow gestures would time out. To compensate for this, the expected head gestures were demonstrated to establish a common gesture baseline.

Shakes and Nods are somewhat different. Some users reported that continuing a shake gesture was easier than a nod gesture. If the system did not respond to the shake, the user would naturally keep gesturing until it responded. For nods, however, most users found it most natural to nod only once. The effects of this can be seen in the results section where shakes have a lower miss rate than nods.

In some cases, they system would not respond immediately even though the gesture was recognized. This was noticed during evaluation when occasionally a user would report a miss, then shortly after the gesture would be recognized without the user performing another gesture. This delay appeared to be random, possibly based on background processes in the system, and was about a second in length. I attempted to correct for this when it was obviously apparent that the gesture was recognized, but in some cases, the user would report a miss and quickly perform a second gesture. It is possible some of these misses are actually hits that were not immediately responsive in the system.

6.5 Usability Study

In addition to the experimental results gathered, test subjects were asked to fill out a short usability study on the head gesture recognition system. This

study consisted of several statements with agree or disagree answers. The study included the following questions plus a short section for comments:

- I found using head gestures for answering dialog boxes intuitive.
- I prefer using head gestures for answering dialog boxes when browsing documents.
- I prefer using head gestures for answering dialog boxes when typing.
- I would use head gestures to answer dialog boxes if they were available on a system.

Out of the fourteen subjects questioned, **93%** found using head gestures for answering dialog boxes intuitive. **71%** prefered using head gestures for answering dialog boxes when browsing documents, and **64%** prefered using had gestures for answering dialog boxes when typing. **79%** would use head gestures to answer dialog boxes if they were available on a system. This strongly supports that using head gestures for answering dialog boxes is intuitive and even prefereable over convetional input. It is still fairly unknown which other applications this is true for, if any.

I found the lower preference for using head gestures during the typing task surprising. I had expected that users would prefer using head gestures during typing since they did not have to move their hands from the keyboard. I expected less users to prefer using head gestures for the reading task than the writing task because the user was allowed to use the trackpad for the reading task and would not have to move their hands from the keyboard like during the writing task. Comments and observation from the usability study showed that some users actually preferred to sit back and relax during the reading task and found

head gestures useful for answering dialog boxes without having to sit forward to interact with the device. This suggests that head gestures may be useful for devices where users a generally in a relaxed position such as in front of a television.

User comments and observation suggest that the users preferred using head gestures during the writing task less since they had to look back at the screen to read the dialog box. Most users were fluent typists and could preform the writing task without looking at the screen. They found having to look back at the screen annoying compared to the reading task since it would pull their attention away from the document they were copying and they would often lose their place. Since the user did not preform the same task with conventional input, the user's opinion may be biased in this case. I hypothesize that user preference for head gestures would be higher in situations where the user types while focusing on the screen, such as when writing original material. Several user comments support this hypothesis, stating that being able to answer dialog boxes without having to move their hands from the keyboard was very convenient.

6.6 Performance

The system ran constantly at 20-30 frames per second on a 1.6 GHz Intel Atom processor using a cheap webcam with a 320x240 resolution. CPU load was approximately 3% at idle without running either prototype. The CPU load increased to 20-30% while running NaiveGestures, and 5% at idle with ContextGestures with ramp-ups to 40-60% during gesture recogntion while displaying a dialog box. Figures 6.2 and 6.3 show the CPU load from windows performance monitor for NaiveGestures and ContextGestures respectively.

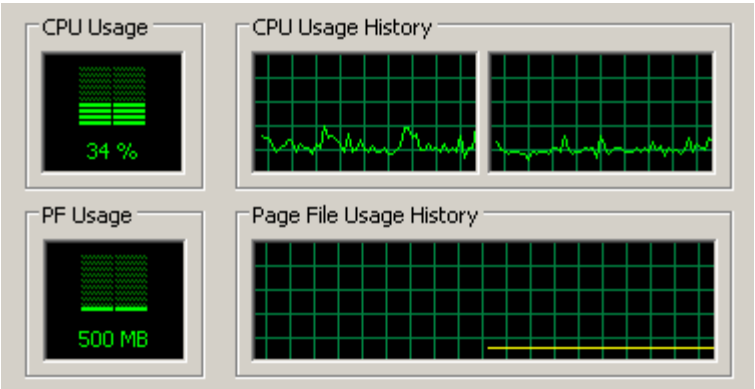

Figure 6.2: NaiveGestures CPU Load

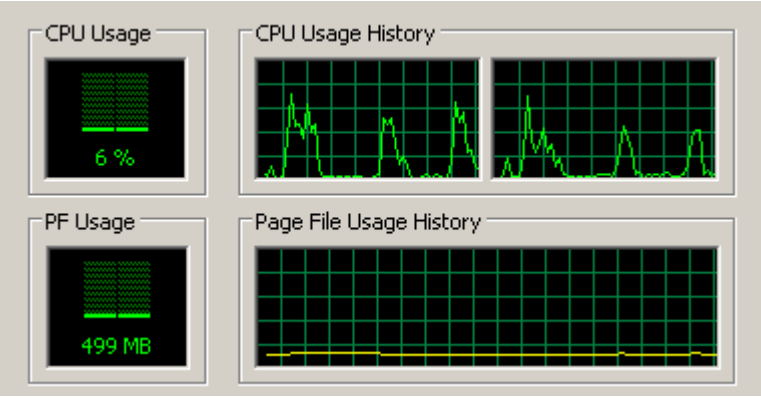

Figure 6.3: ContextGestures CPU Load

This suggests that the test prototypes are capable of running on even slower hardware. NaiveGesures increases CPU load by about 20% contantly. However, ContextGestures only increases CPU load by aproximately 1-2% while idle (which would be the majority of operation). However, there appears to be some overhead required for reinitalization of the head gesture system which results in almost twice as much CPU load when recognizing a gesture. This suggests there might be a small delay on low performance devices while the CPU load dramatically increases to initialize the head gestures.

Chapter 7

Conclusion

7.1 Summary

The research presented in this thesis validates that using contextual clues significantly reduces the number of false positives in a vision-based head gesture recognition system. My research suggests that the solution presented satisfies the requirements for efficiency and accuracy without using special hardware. The work presented is original from other work in that it uses a FSM to improve efficiency over the work presented by Morency et al. The improvement I saw was much greater than Morency et al., but my implementation was much more susceptible to false positives without the contextual clues. This suggests that contextual clues may make up the difference when using techniques more prone to false positives. I am able to achieve comparable accuracy to Morency using a more efficient but less accurate gesture recognition method. My results clearly show that contextual clues dramatically reduce false positives and helps address the Midas Touch problem.

A major contribution of this thesis is a reusable head gesture recognition library. ContextGestures ran in real time with an accuracy in the high nineties and false positives from zero to eight percent. This suggests that such a system may be ready for everyday use. The CPU load was also low during idle periods so head gesture recognition will not consume many system resources. With more work, this system may become stable enough for general use.

The results of the usability study suggests that head gestures make an intuitive and desirable interface for answering dialog boxes. Such an interface may be useful in system where the user is generally in a relaxed position or has limited tradition interface such as a television or hand held device. The results show that such an interface may be desirable even when traditional interfaces are available.

7.2 Future Work

The results in this thesis are promising, but much remains to be explored in the realm of head gesture recognition. The implementation presented in this thesis performed admirably without any significant optimization or training. With a concentrated effort, accuracy could be improved by determining optimal values for parameters such as gestures timeout, minimum gesture magnitude, etc. Most of these values were chosen arbitrarily without much scientific and little empirical support. Research in head gestures could produce an optimal set of parameters to improve accuracy to everyday usable levels.

The evaluation results presented in this research are somewhat weak. The tested user group is small and not very diverse. While accuracy was high in a controlled environment with controlled tasks, it remains to be seen how the system would perform in everyday environments. More rigorous testing is required

to provide a concrete accuracy rating.

The system could also be expanded to provide input to yes or no questions. Since the gesture recognition system was built as a library using an event based architecture, it can be used in any system to trigger events with nods and shakes. These could be used as input to yes/no questions for dialog boxes, or even other unexplored forms of input.

Probably the most important work yet to be done is a serious study of practical applications of head gesture recognition technology. While many users find head gestures natural for interacting with dialog boxes, it is unknown what other applications head gesture recognition could be useful for. Additionally, determining a good set of contextual clues to use would also provides a significant body of research yet to be explored.

Some possible applications for continuous head gestures include changing perspective in 3D virtual environments such as in a video game or with 3D modeling software. Head gestures can be used to move the head of a 3D avatar, or head tracking can be used to change perspective to simulate depth in a 3D environment.

Other significant future work includes implementing head gesture recognition on a hand held device such as a smartphone. These platforms are significantly different from a traditional PC and provides a whole new set of challenges for accurate gesture recognition.

Bibliography

[1] Bradski and Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. OReilly Media, Inc., 2008.

[2] H. I. Choi and P. K. Rhee. Head gesture recognition using HMMs. *Expert Systems with Applications*, 17(3):213–221, Oct. 1999.

[3] J. L. Crowley and F. Berard. Multi-modal tracking of faces for video communications. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pages 640—645, 1997.

[4] J. W. Davis. A perceptual user interface for recognizing head gesture acknowledgements. *In ACM Workshop on Perceptual User Interfaces*, pages 15—16, 2001.

[5] L. Dong, Y. Jin, L. Tao, and G. Xu. Recognition of Multi-Pose head gestures in human conversations. In *ICIG '07: Proceedings of the Fourth International Conference on Image and Graphics*, pages 650–654, 2007.

[6] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108(1-2):52–73, 2007.

[7] T. Funahashi, T. Fujiwara, and H. Koshimizu. Face and eye tracking for

gaze analysis. In *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, pages 1337–1341, 2007.

[8] A. Kapoor and R. W. Picard. A real-time head nod and shake detector. In *in Proceedings from the Workshop on Perspective User Interfaces*, pages 1–5, Orlando, Florida, 2001. ACM.

[9] S. Kawato and J. Ohya. Real-Time detection of nodding and Head-Shaking by directly detecting and tracking the "Between-Eyes". In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*, page 40. IEEE Computer Society, 2000.

[10] S. Kawato and N. Tetsutani. Detection and tracking of eyes for gaze-camera control. *Image and Vision Computing*, 22(12):1031–1038, Oct. 2004.

[11] D. Kern, P. Marshall, and A. Schmidt. Gazemarks: gaze-based visual placeholders to ease attention switching. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 2093–2102, New York, NY, USA, 2010. ACM.

[12] R. Kjeldsen. Head gestures for computer control. In *RATFG-RTS '01: Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems (RATFG-RTS'01)*, pages 61–67. IEEE Computer Society, 2001.

[13] R. Kjeldsen and J. Hartman. Design issues for vision-based computer interaction systems. In *In Proc. of the Workshop on Perceptual User Interfaces. 2001*, pages 1–8, Orlando, Florida, 2001. ACM.

[14] P. Lu, M. Zhang, X. Zhu, and Y. Wang. Head nod and shake recognition based on multi-view model and hidden markov model. In *CGIV '05: Pro-*

ceedings of the International Conference on Computer Graphics, Imaging and Visualization, pages 61–64, Washington, DC, USA, 2005. IEEE Computer Society.

[15] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130, 1981.

[16] G. McGlaun, F. Althoff, M. Lang, and G. Rigoll. Robust Video-Based recognition of dynamic head gestures in various Domains-Comparing a Rule-Based and a stochastic approach. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 180–197, 2004.

[17] S. Mitra and T. Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007.

[18] L. Morency and T. Darrell. Head gesture recognition in intelligent interfaces: the role of context in improving recognition. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 32–38, Sydney, Australia, 2006. ACM.

[19] L. Morency, C. Sidner, C. Lee, and T. Darrell. Contextual recognition of head gestures. In *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, pages 18–24, Torento, Italy, 2005. ACM.

[20] L. Morency, C. Sidner, C. Lee, and T. Darrell. Head gestures for perceptual interfaces: The role of context in improving recognition. *Artificial Intelligence*, 171(8-9):568–585, June 2007.

[21] T. Nawaz, M. Mian, and H. Habib. Infotainment devices control by eye gaze and gesture recognition fusion. *Consumer Electronics, IEEE Transactions on*, 54(2):277 –282, may 2008.

[22] P. C. Ng and L. D. Silva. Head gestures recognition. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 266–269, 2001.

[23] K. Nickels and S. Hutchinson. Estimating uncertainty in SSD-based feature tracking. *Image and Vision Computing*, 20(1):47–58, 2002.

[24] H. Nonaka. Communication interface with Eye-Gaze and head gesture using successive DP matching and fuzzy inference. *Journal of Intelligent Information Systems*, 21(2):105–112, 2003.

[25] H. Prendinger, T. Eichner, E. André, and M. Ishizuka. Gaze-based infotainment agents. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pages 87–90, New York, NY, USA, 2007. ACM.

[26] H. A. Rowley, S. Baluja, and T. Kanade. Neural Network-Based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[27] K. Sung and T. Poggio. Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):39–51, 1998.

[28] J. Tang and R. Nakatsu. A head gesture recognition algorithm. In *Advances in Multimodal Interfaces ICMI 2000*, pages 72–80, 2000.

[29] C. Tomasi and T. Kanade. Detection and tracking of point features. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 9:137—154, 1991.

[30] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, pages 511–518, 2001.

[31] P. Viola and M. Jones. Robust real-time object detection. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 2001.

[32] P. I. Wilson and J. Fernandez. Facial feature detection using haar classifiers. *J. Comput. Small Coll.*, 21(4):127–133, 2006.

[33] Y. Wu, T. S. Huang, and N. Mathews. Vision-based gesture recognition: A review. *Lecture Notes in Computer Science*, 1739:103—115, 1999.

[34] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, 2006.

[35] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458, 2003.