Context-based Visual Feedback Recognition

by

Louis-Philippe Morency

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[February 2007]
~~October 2006~~

Author [illegible]
Department of Electrical Engineering and Computer Science
October 30th, 2006

Certified by [illegible]
Trevor Darrell
Associate Professor
Thesis Supervisor

Accepted by [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Context-based Visual Feedback Recognition

by

Louis-Philippe Morency

Submitted to the Department of Electrical Engineering and Computer Science
on October 30th, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

During face-to-face conversation, people use visual feedback (e.g., head and eye gesture) to communicate relevant information and to synchronize rhythm between participants. When recognizing visual feedback, people often rely on more than their visual perception. For instance, knowledge about the current topic and from previous utterances help guide the recognition of nonverbal cues. The goal of this thesis is to augment computer interfaces with the ability to perceive visual feedback gestures and to enable the exploitation of contextual information from the current interaction state to improve visual feedback recognition.

We introduce the concept of *visual feedback anticipation* where contextual knowledge from an interactive system (e.g. last spoken utterance from the robot or system events from the GUI interface) is analyzed online to anticipate visual feedback from a human participant and improve visual feedback recognition. Our multi-modal framework for context-based visual feedback recognition was successfully tested on conversational and non-embodied interfaces for head and eye gesture recognition.

We also introduce *Frame-based Hidden-state Conditional Random Field* model, a new discriminative model for visual gesture recognition which can model the substructure of a gesture sequence, learn the dynamics between gesture labels, and can be directly applied to label unsegmented sequences. The FHCRF model outperforms previous approaches (i.e. HMM, SVM and CRF) for visual gesture recognition and can efficiently learn relevant contextual information necessary for visual feedback anticipation.

A real-time visual feedback recognition library for interactive interfaces (called Watson) was developed to recognize head gaze, head gestures, and eye gaze using the images from a monocular or stereo camera and the context information from the interactive system. Watson was downloaded by more then 70 researchers around the world and was successfully used by MERL, USC, NTT, MIT Media Lab and many other research groups.

Thesis Supervisor: Trevor Darrell
Title: Associate Professor

Acknowledgments

First and foremost, I would like to thank my research advisor, Trevor Darrell, who with his infinite wisdom has always been there to guide me through my research. You instilled in me the motivation to accomplish great research and I thank you for that.

I would like to thank the members of my thesis committee, Candy Sidner, Michael Collins and Stephanie Seneff, for their constructive comments and helpful discussions.

As I learned over the years, great research often starts with great collaboration. One of my most successful collaborations was with two of the greatest researchers: Candy Sidner and Christopher Lee. I have learned so much while collaborating with you and I am looking forward to continuing our collaboration after my PhD.

Over the past 6 years at MIT I have worked in three different offices with more than 12 different officemates and I would to thank all of them for countless interesting conversations. I would like to especially thank my two long-time "officemates" and friends: C. Mario Christoudias and Neal Checka. You have always been there for me and I appreciate it greatly. I'm really glad to be your friend. I would like to thank my accomplice Ali Rahimi. I will remember the many nights we spent together writing papers and listening to Joe Dassin and Goldorak.

I would like to thank Victor Dura Vila who was my roommate during my first three years at MIT and became one of my best friends and confident. Thank you for your support and great advice. I will always remember that one night in Tang Hall where we used your white board to solve this multiple-choice problem.

My last three years in Boston would not have been the same without my current roommate Rodney Daughtrey. I want to thank you for all the late night discussions after Tuesday night hockey and your constant support during the past three years. Dude!

I would like to thank Marc Parizeau, my undergraduate teacher from Laval University, Quebec city, who first introduced me to research and shared with me his passion for computer vision.

I would like to thank my uncle Jean-Francois for introducing me to C++ pro-

gramming and computer science in general. I will always remember all the summer days spent at Orleans Island in your company. Thank you for sharing your wisdom and always pushing me to reach my dreams.

Merci chere maman pour tout ce que tu as fait pour moi. Durant plus de 20 annnees (et encore aujourd'hui), tu m'as donne ton amour et ta sagesse. Tu m'as appris a partager et a respecter les autres. Je ne pourrai te le dire assez souvent: Merci maman. Je t'aime.

Merci cher papa de m'avoir encourage a toujours aller plus loin. C'est grace a toi si j'ai fait ma demande a MIT. Durant ma derniere annee a Quebec, j'ai decouvert ce que la complicite pere-fils signifiait. Merci pour tout ce que tu as fait pour moi. Je t'aime papa.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, research in computer science has brought about new technologies to produce multimedia animations, and sense multiple input modalities from spoken words to detecting faces from images. These technologies open the way to new types of human-computer interfaces, enabling modern interactive systems to communicate more naturally with human participants. Figure 1-1 shows two examples of modern interactive interfaces: Mel, a robotic penguin which hosts a participant during a demo of a new product [84] and MACK, an on-screen character which offers directions to visitors of a new building [69]. By incorporating human-like display capabilities into these interactive systems, we can make them more efficient and attractive, but doing so increases the expectations from human participants who interact with these systems. A key component missing from most of these moderns systems is the recognition of visual feedback.

During face-to-face conversation, people use visual feedback to communicate relevant information and to synchronize rhythm between participants. When people interact naturally with each other, it is common to see indications of acknowledgment, agreement, or disinterest given with a simple head gesture. For example, when finished with a turn and willing to give up the floor, users tend to look at their conversational partner. Conversely, when they wish to hold the floor, even as they pause their speech, they often look away. This often appears as a "non-deictic" or *gaze-averting* eye gesture while they pause their speech to momentarily consider a

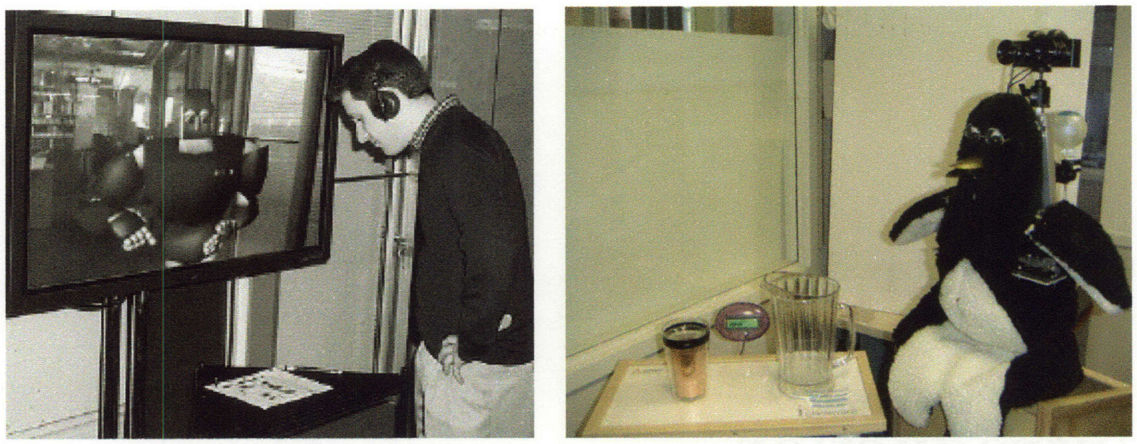


Figure 1-1: Two examples of modern interactive systems. MACK (left) was designed to study face-to-face grounding [69]. Mel (right), an interactive robot, can present the iGlassware demo (table and copper cup on its right) or talk about its own dialogue and sensorimotor abilities.

response [35, 86].

Recognizing visual feedback such as this from the user can dramatically improve the interaction with the system. When a conversational system is speaking, head nods can be detected and used by the system to know that the listener is engaged in the conversation. This may be more natural interface compared to the original version where the robot had to ask a question to get the same feedback. By recognizing eye gaze aversion gestures, the system can understand delays resulting from the user thinking about their answer when asked a question. It is disruptive for a user to have a system not realize the user is still thinking about a response and interrupt prematurely, or to wait inappropriately thinking the user is still going to speak when the user has in fact finished their turn. The main focus of this thesis is visual feedback recognition for interactive systems.

1.1 Challenges

Recognition of visual feedback for interactive systems brings many challenges:

Ambiguous Gestures Different types of visual feedback can often look alike. For example, a person's head motion during a quick glance to the left looks similar to a short head shake. A person's head motion when looking at their keyboard before typing looks similar to a head nod. We need a recognition framework which is able to differentiate these ambiguous gestures.

Subtle Motion Natural gestures are subtle and the range of motion involved is often really small. For example, when someone head nods to ground (which is intended as an acknowledgement) a sentence or part of a sentence, the head motion will usually be really small and often only one nod is performed.

Natural Visual Feedback Which visual gestures should we be looking for? While a human participant may potentially offer a large range of visual feedback while interacting with the system, which visual feedback, if correctly recognized by the interactive system, will improve the system performance and which visual gestures are naturally performed by the user?

User Independent and Automatic Initialization Recognition of the visual feedback from a new user should start automatically even though this new user never interacted with the system before and was not in the system's training data. Also, our algorithms should be able to handle different facial appearances, including people with glasses and beards, and different ways to gesture (short head nod versus slower head nod).

Diversity of System Architectures Each interactive system has its own architecture with a different representation of internal knowledge and its own protocol for exchange of information. An approach for visual feedback recognition should be easy to integrate with pre-existing systems.

1.2 Contributions

The contributions of this thesis intersect the fields of human-computer interaction (HCI), computer vision, machine learning and multi-modal interfaces.

Visual Feedback Anticipation is a new concept to improve visual feedback recognition where contextual knowledge from the interactive system is analyzed online to anticipate visual feedback from the human participant. We developed a multi-modal framework for context-based visual recognition was successfully tested on conversational and non-embodied interfaces for head gesture and eye gestures recognition and has shown a statistically significant improvement in recognition performance.

Frame-based Hidden-state Conditional Random Fields is a new discriminative model for visual gesture recognition which can model the sub-structure of a gesture sequence, can learn the dynamics between gesture labels and be directly applied to label unsegmented sequences. Our FHCRF model outperforms previous approaches (i.e. HMM and CRF) for visual gesture recognition and can efficiently learn relevant contextual information necessary for visual feedback anticipation.

Visual Feedback User Study In the context of our user studies with virtual and physically embodied agents, we investigate four kinds of visual feedback that are naturally performed by human participants when interacting with interactive interfaces and which, if automatically recognized, can improve the user experience: head gaze, head gestures (head nods and head shakes), eye gaze and eye gestures (gaze aversions).

Adaptive View-based Appearance Model is a new user independent approach for head pose estimation which merges differential tracking with view-based tracking. AVAMs can track large head motion for long periods of time with bounded drift. We experimentally observed an RMS error within the accuracy limit of an attached inertial sensor.

Watson is a real-time visual feedback recognition library for interactive interfaces that can recognize head gaze, head gestures, eye gaze and eye gestures using the images of a monocular or stereo camera. Watson has been downloaded by more then 70 researchers around the world and was successfully used by MERL, USC, NTT, Media Lab and many other research groups.

The following five subsections discuss with more details each contribution.

1.2.1 Visual Feedback Anticipation

To recognize visual feedback efficiently, humans often use contextual knowledge from previous and current events to anticipate when feedback is most likely to occur. For example, at the end of a sentence during an explanation, the speaker will often look at the listener and anticipate some visual feedback like a head nod gesture from the listener. Even if the listener does not perform a *perfect* head nod, the speaker will most likely accept the gesture as a head nod and continue with the explanation. Similarly, if a speaker refers to an object during his/her explanation and looks at the listener, even a short glance will be accepted as glance-to-the-object gesture (see Figure 1-2).

In this thesis we introduce[1] the concept of visual feedback anticipation for human-computer interfaces and present a context-based recognition framework for analyzing online contextual knowledge from the interactive system and anticipate visual feedback from the human participant. As shown in Figure 1-2, the contextual information from the interactive system (i.e. the robot) is analyzed to predict visual feedback and then incorporated with the results of the vision-only recognizer to make the final decision. This new approach makes it possible for interactive systems to simulate the natural anticipation that humans use when recognizing feedback.

Our experimental results show that incorporating contextual information inside the recognition process improves performance. For example, ambiguous gestures like a

[1]First published at AISB in April 2005 [72].

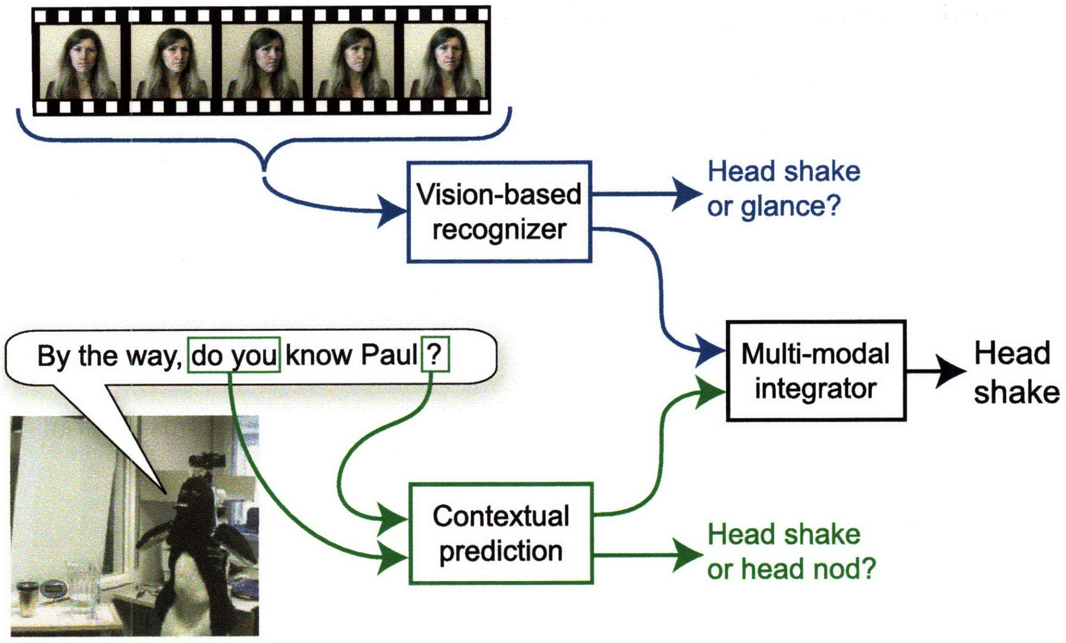


Figure 1-2: Contextual recognition of head gestures during face-to-face interaction with a conversational robot. In this scenario, contextual information from the robot's spoken utterance helps to disambiguate the listener's visual gesture.

head glance and head shake can be differentiated using the contextual information (see Figure 1-2). With a conversational robot or virtual agent, lexical, prosodic, timing, and gesture features are used to predict a user's visual feedback during conversational dialogue. In non-conversational interfaces, context features based on user-interface system events (i.e. mouse motion or a key pressed on the keyboard) can improve detection of head gestures for dialog box confirmation or document browsing.

1.2.2 FHCRF: Visual Gesture Recognition

Visual feedback tends to have a distinctive internal sub-structure as well as regular dynamics between individual gestures. When we say that a gesture has internal sub-structure we mean that it is composed of a set of basic motions that are combined in an orderly fashion. For example, consider a head-nod gesture which consists of moving the head up, moving the head down and moving the head back to its starting position. Further, the transitions between gestures are not uniformly likely: it is clear that a head-nod to head-shake transition is less likely than a transition between a head-nod and another gesture (unless we have a very indecisive participant).

In this thesis we introduce a new visual gesture recognition algorithm, which can capture both sub-gesture patterns and dynamics between gestures. Our Frame-based Hidden Conditional Random Field (FHCRF) model is a discriminative, latent variable, on-line approach for gesture recognition. Instead of trying to model each gesture independently as previous approaches would (i.e. Hidden Markov Models), our FHCRF model focuses on what best differentiates all the visual gestures. As we show in our results, our approach can more accurately recognize subtle gestures such as head nods or eye gaze aversion.

Our approach also offers several advantages over previous discriminative models: in contrast to Conditional Random Fields (CRFs) [56] it incorporates hidden state variables which model the sub-structure of a gesture sequence, and in contrast to previous hidden-state conditional models [79] it can learn the dynamics between gesture labels and be directly applied to label unsegmented sequences. We show that, by assuming a deterministic relationship between class labels and hidden states, our model can be efficiently trained.

1.2.3 User Studies with Interactive Interfaces

While a large range of visual feedback is performed by humans when interacting face-to-face with each other, it is still an open question as to what visual feedback is performed naturally by human participants when interacting with an interactive system. Another important question is which visual feedback has the potential to improve current interactive systems when recognized.

With this in mind, we performed a total of six user studies with three different types of interactive systems: virtual embodied interfaces, physical embodied interfaces and non-embodied interfaces. Virtual embodied interfaces, also referred to as Embodied Conversational Agent (ECA) [16] or Intelligent Virtual Agent (IVA) [3], are autonomous, graphically embodied agents in an interactive, 2D or 3D virtual environment which are able to interact intelligently with the environment, other embodied agents, and especially with human users. Physical embodied interfaces, or robots, are physically embodied agents able to interact with human user in the real world. Non-embodied interfaces are interactive systems without any type of embodied character, which include traditional graphical user interfaces (GUIs) and information kiosks (without an on-screen character).

In our user studies, we investigate four kinds of visual feedback that have the potential to improve the experience of human participants when interacting one-on-one with an interactive interface: head gaze, head gestures, eye gaze and eye gestures. Head gaze can be used to determine to whom the human participant is speaking and can complement eye gaze information when estimating the user's focus of attention. Head gestures such as head nods and head shakes offer key conversational cues for information grounding and agreement/disagreement communication. Eye gaze is an important cue during turn-taking and can help to determine the focus of attention. Recognizing eye gestures like eye gaze aversions can help the interactive system to know when a user is thinking about his/her answer versus waiting for more information.

1.2.4 AVAM: Robust Online Head Gaze Tracking

One of our goal is to develop a user independent head gaze tracker that can be initialized automatically, can track over a long period of time and should be robust to different environments (lighting, moving background, etc.). Since head motion is used as an input by the FHCRF model for head gesture recognition, our head pose tracker has to be sensitive enough to distinguish natural and subtle gestures.

In this thesis, we introduce a new model for online head gaze tracking: Adaptive View-based Appearance Model (AVAM). When the head pose trajectory crosses itself, our tracker has bounded drift and can track an object undergoing large motion for long periods of time. Our tracker registers each incoming frame against the views of the appearance model using a two-frame registration algorithm. Using a linear Gaussian filter, we simultaneously estimate the pose of the head and adjust the view-based model as pose-changes are recovered from the registration algorithm. The adaptive view-based model is populated online with views of the head as it undergoes different orientations in pose space, allowing us to capture non-Lambertian effects. We tested our approach on a real-time rigid object tracking task and observed an RMS error within the accuracy limit of an attached inertial sensor.

1.2.5 Watson: Real-time Library

We developed a real-time library for context-based recognition of head gaze, head gestures, eye gaze and eye gestures based on the algorithms described in this thesis. This library offers a simplified interface for other researchers who want to include visual feedback recognition in their interactive system. This visual module was successfully used in many research projects including:

1. **Leonardo**, a robot developed at the Media lab, can quickly learn new skills and tasks from natural human instruction and a few demonstrations [102];

2. **Mel**, a robotic penguin developed at MERL, hosting a participant during a demo of a new product [89, 92];

3. **MACK** was originally developed to give direction around the Media Lab and now its cousin **NU-MACK** can help with direction around the Northwestern University campus [69];

4. **Rapport** was a user study performed at the USC Institute for Creative Technologies to improve the engagement between a human speaker and virtual human listener [36].

1.3 Road map

In chapter 2, we present our user studies with virtual embodied agents, robots and non-embodied interfaces designed to observe natural and useful visual feedback. We also present some related user studies where part of our approach for visual feedback recognition was used.

In chapter 3, we first present our novel model, AVAM, for online head gaze estimation with a comparative evaluation with an inertial sensor. We then present our approach for eye gaze estimation based on view-based eigenspaces. We conclude Chapter 3 with our new algorithm, FHCRF, for gesture recognition, and show its performance on vision-based head gestures and eye gestures recognition.

In chapter 4, we present our new concept for context-based visual feedback recognition and describe our multi-modal framework for analyzing online contextual information and anticipating head gestures and eye gestures. We present experimental results with conversational interfaces and non-embodied interfaces.

In chapter 5, we summarize our contributions and discuss future work, and in Appendix A, we provide the the user guide of Watson, our context-based visual feedback recognition library.

Chapter 2

Visual Feedback for Interactive Interfaces

During face-to-face conversation, people use visual feedback to communicate relevant information and to synchronize communicative rhythm between participants. While a large literature exists in psychology describing and analyzing visual feedback during human-to-human interactions, there are still a lot of unanswered questions about natural visual feedback for interactive interfaces.

We are interested in natural visual feedback, meaning visual feedback that human participants perform automatically (without being instructed to perform these gestures) when interacting with an interactive system. We also want to find out what kind of visual feedback, if recognized properly by the system, would make the interactive interface more useful and efficient.

In this chapter, we present five user studies designed to (1) analyze the visual feedback naturally performed by human participants when interacting with an interactive system and (2) find out how recognizing this visual feedback can improve the performance of interactive interfaces. While visual feedback can be expressed by almost any part of the human body, this thesis focuses on facial feedback, since our main interest is face-to-face interaction with embodied and non embodied interfaces. In the context of our user studies, we discuss forms of visual feedback (head gaze, eye gaze, head gestures and eye gestures) that were naturally performed by our participants. We then show how interactive interfaces can be improved by recognizing and utilizing these visual gestures.

Since interactive systems come in many different forms, we designed our user studies to explore two different axes: embodiment and conversational capabilities. For embodiment, we group the interactive interfaces into three categories: virtual embodied interfaces, physical embodied interfaces and non-embodied interfaces. Virtual embodied interfaces are autonomous, graphically embodied agents in an interactive, 2D or 3D virtual environment. Physical embodied interfaces are best represented as robots or robotic characters that interact with a human user in the real world. Non-embodied interfaces are interactive systems without any type of embodied character such as a conventional operating system or an information kiosk.

The conversational capabilities of interactive systems can range from simple trigger-based interfaces to elaborated conversational interfaces. Trigger-based interfaces can be seen as simple responsive interfaces where the system performs a pre-determined set of actions when triggered by the input modality (or modalities). Conversational interfaces usually incorporate a dialogue manager that will decide the next set of actions based on the current inputs, a history of previous actions performed by the participants (including the interface itself) and the goals of the participants. Some interfaces like the *conversational tooltips* described later in this chapter are an intermediate between a trigger-based approach and a conversational interface since they use a trigger to start the interaction (i.e. you looking at a picture) and then start a short conversation if you accept the help from the agent.

The following section reviews work related to the use of visual feedback with interactive interfaces. We focus on two interactive systems that are particularly relevant to our research: MACK, an interactive on-screen character and Mel, an interactive robot. In Section 2.2, we present three user studies that we performed with virtual embodied agents: Look-to-talk, Conversational Tooltips and Gaze Aversion. In Section 2.3, we present a collaborative user study made with researchers at MERL about head nodding for interaction with a robot. In Section 2.4, we present a user study with two non-embodied gesture-based interfaces: dialog box answering/acknowledgement and document browsing. Finally, in Section 2.5, we summarize the results of the user studies and present four kinds of natural visual feedback useful for interactive interfaces.

2.1 Related Work

Several systems have exploited head-pose cues or eye gaze cues in interactive and conversational systems. Stiefelhagen has developed several successful systems for tracking face pose in meeting rooms and has shown that face pose is very useful for predicting turn-taking [98]. Takemae *et al.* also examined face pose in conversation and showed that, if tracked accurately, face pose is useful for creating a video summary of a meeting [101]. Siracusa *et al.* developed a kiosk front end that uses head pose tracking to interpret who was talking to whom in a conversational setting [93]. The position and orientation of the head can be used to estimate head gaze, which is a good estimate of a person's attention. When compared with eye gaze, head gaze can be more accurate when dealing with low resolution images and can be estimated over a larger range than eye gaze [65].

Several authors have proposed face tracking for pointer or scrolling control and have reported successful user studies [105, 52]. In contrast to eye gaze [118], users seem to be able to maintain fine motor control of head gaze at or below the level needed to make fine pointing gestures[1]. However, many systems required users to manually initialize or reset tracking. These systems supported a direct manipulation style of interaction and did not recognize distinct gestures.

A considerable body of work has been carried out regarding eye gaze and eye motion patterns for human-computer interaction. Velichkovsky suggested the use of eye motion to replace the mouse as a pointing device [108]. Qvarfordt and Zhai used eye-gaze patterns to sense user interest with a map-based interactive system [80]. Li and Selker developed the InVision system which responded to a user's eye fixation patterns in a kitchen environment [60].

[1]Involuntary microsaccades are known to limit the accuracy of eye-gaze based tracking [47].

In a study of eye gaze patterns in multi-party (more than two people) conversations, Vertegaal *et al.* [109] showed that people are much more likely to look at the people they are talking to than any other people in the room. Also, in another study, Maglio *et al.* [61] found that users in a room with multiple devices almost always look at the devices before talking to them. Stiefelhagen *et al.* [99] showed that the focus of attention can be predicted from the head position 74% of the time during a meeting scenario.

Breazeal's work [14] on infantoid robots explored how the robot gazed at a person and responded to the person's gaze and prosodic contours in what might be called pre-conversational interactions. Davis and Vaks modeled head nods and head shakes using a timed finite state machine and suggested an application with an on-screen embodied agent [28].

There has been substantial research in hand/body gesture in for human-computer interaction. Lenman *et al.* explored the use of pie- and marking menus in hand gesture-based interaction[59]. Cohen *et al.* studied the issues involved in controlling computer applications via hand gestures composed of both static and dynamic symbols[24].

There has been considerable work on gestures with virtual embodied interfaces, also known as embodied conversational agents (ECAs). Bickmore and Cassell developed an ECA that exhibited many gestural capabilities to accompany its spoken conversation and could interpret spoken utterances from human users [8]. Sidner *et al.* have investigated how people interact with a humanoid robot [91]. They found that more than half their participants naturally nodded at the robot's conversational contributions even though the robot could not interpret head nods. Nakano *et al.* analyzed eye gaze and head nods in computer–human conversation and found that their subjects were aware of the lack of conversational feedback from the ECA [69]. They incorporated their results in an ECA that updated its dialogue state. Numerous other ECAs (e.g. [106, 15]) are exploring aspects of gestural behavior in human-ECA interactions. Physically embodied ECAs—for example, ARMAR II [31, 32] and Leo [13]–have also begun to incorporate the ability to perform articulated body tracking and recognize human gestures.

2.1.1 MACK: Face-to-face grounding

MACK (Media lab Autonomous Conversational Kiosk) is an embodied conversational agent (ECA) that relies on both verbal and nonverbal signals to establish common ground in computer–human interactions [69]. Using a map placed in front of the kiosk and an overhead projector, MACK can give directions to different research projects of the MIT Media Lab. Figure 2-1 shows a user interacting with MACK.

The MACK system tokenizes input signals into utterance units (UU) [78] corresponding to single intonational phrases. After each UU, the dialogue manager decides the next action based on the log of verbal and nonverbal events. The dialogue manager's main challenge is to determine if the agent's last UU is grounded (the information was understood by the listener) or is still ungrounded (a sign of miscommunication).

As described in [69], a grounding model has been developed based on the verbal and nonverbal signals happening during human–human interactions. The two main nonverbal patterns observed in the grounding model are gaze and head nods. Nonverbal patterns are used by MACK to decide whether to proceed to the next UU or elaborate on the current one. Positive evidence of grounding is recognized by MACK if the user looks at the map or nods his or her head. In this case, the agent goes ahead with the next UU 70% of the time. Negative evidence of grounding is recognized if the user looks continuously at the agent. In this case, MACK will elaborate on the current UU 73% of the time. These percentages are based on the analysis of human–human interactions. In the final version of MACK, Watson, our real-time visual feedback recognition library, was used to estimate the gaze of the user and detect head nods.

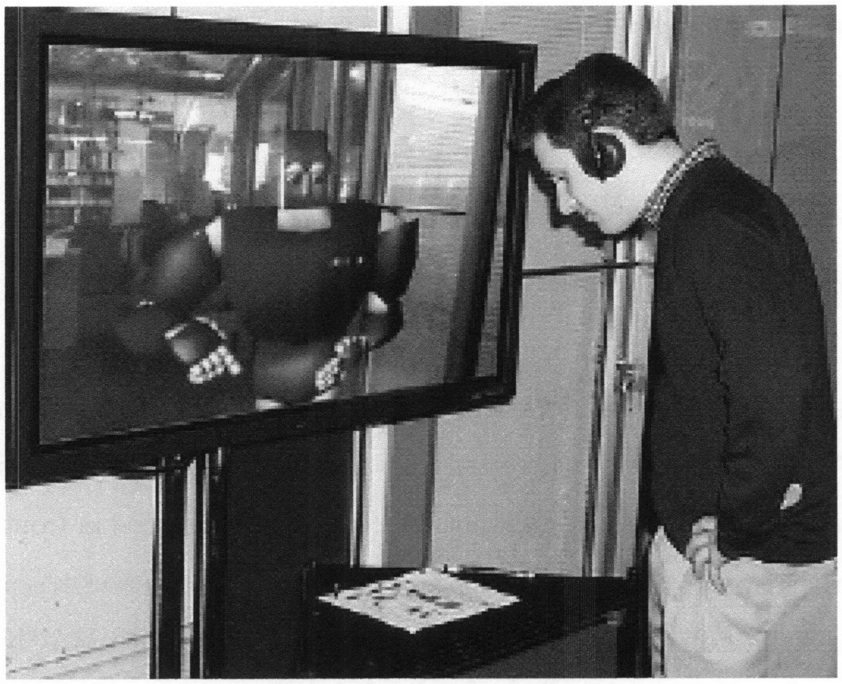


Figure 2-1: MACK was designed to study face-to-face grounding [69]. Directions are given by the avatar using a common map placed on the table which is highlighted using an over-head projector. The head pose tracker is used to determine if the subject is looking at the common map.

2.1.2 Mel: Human–Robot Engagement

Mel is a robot developed at Mitsubishi Electric Research Labs (MERL) that mimics human conversational gaze behavior in collaborative conversation [89]. One important goal of this project is to study engagement during conversation. The robot performs a demonstration of an invention created at MERL in collaboration with the user (see Figure 2-2).

Mel's conversation model, based on COLLAGEN [84], determines the next move on the agenda using a predefined set of engagement rules, originally based on human–human interaction [90]. The conversation model also assesses engagement information about the human conversational partner from a Sensor Fusion Module, which keeps track of verbal (speech recognition) and nonverbal cues (multiview face detection[110]).

A recent experiment using the Mel system suggested that users respond to changes in head direction and gaze by changing their own gaze or head direction[89]. Another interesting observation is that people tend to nod their heads at the robot during

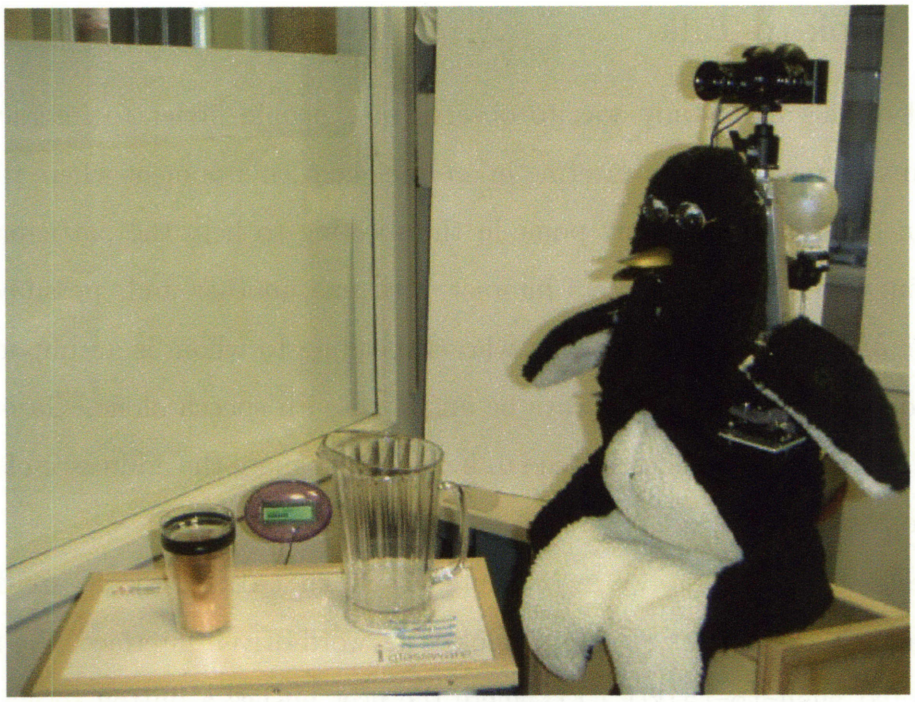


Figure 2-2: Mel, an interactive robot, can present the iGlassware demo (table and copper cup on its right) or talk about its own dialogue and sensorimotor abilities.

explanation. These kinds of positive responses from the listener could be used to improve the engagement between a human and robot.

2.2 Virtual Embodied Interfaces

In this section, we explore how visual feedback can be used when interacting with an on-screen character. For this purpose we present three user studies that range from a trigger-based interface to simple conversational interfaces. The "look-to-talk" experiment, described in Section 2.2.1, explores the use of head gaze as a trigger for activating an automatic speech recognizer inside a meeting environment. The "conversational tooltips" experiment, described in Section 2.2.2, explores an intermediate interface between trigger-based and conversational using head gaze and head gestures. The "gaze aversion" experiment, described in Section 2.2.3 looks at recognition of eye gestures for a scripted conversational interface.

2.2.1 Look-to-Talk

The goal of this user study was to observe how people prefer to interact with a simple on-screen agent when meeting in an intelligent environment where the virtual agent can be addressed at any point in the meeting to help the participants[1]. In this setting where multiple users interact with one another and, possibly, with a multitude of virtual agents, knowing who is speaking to whom is an important and difficult question that cannot always be answered with speech alone. Gaze tracking has been identified as an effective cue to help disambiguate the addressee of a spoken utterance [99].

To test our hypothesis, we implemented `look-to-talk`(LTT), a gaze-driven interface, and `talk-to-talk` (TTT), a spoken keyword-driven interface. We have also implemented `push-to-talk`(PTT), where the user pushes a button to activate the speech recognizer. We present and discuss a user evaluation of our prototype system as well as a Wizard-of-Oz setup.

Experimental Study

We set up the experiment to simulate a collaboration activity among two subjects and a software agent in an the Intelligent Room [23] (from here on referred to as the I-Room). The first subject (subject A) sits facing the front wall displays, and a second "helper" subject (subject B) sits across from subject A. The task is displayed on the wall facing subject A. The camera is on the table in front of subject A, and Sam, an animated character representing the software agent, is displayed on the side wall (see Figure 2-3). Subject A wears a wireless microphone and communicates with Sam via IBM ViaVoice. Subject B discusses the task with subject A and acts as a collaborator. The I-Room does not detect subject B's words and head-pose.

Sam is the I-Room's "emotive" user interface agent. Sam consists of simple shapes forming a face, which animate to continually reflect the I-Room's state (see Figure 2-3). During this experiment, Sam reads quiz questions through a text-to-speech

[1]The results in Section 2.2.1 were obtained in collaboration with Alice Oh, Harold Fox, Max Van Kleek, Aaron Adler and Krzysztof Gajos. It was originally published at CHI 2002 [72].

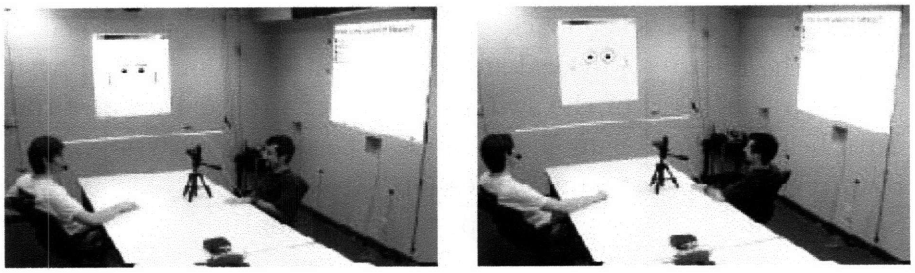


Figure 2-3: Look-To-Talk in non-listening (left) and listening (right) mode.

synthesizer and was constrained to two facial expressions: non-listening and listening.

To compare the usability of `look-to-talk` with the other modes, we ran two experiments in the I-Room. We ran the first experiment with a real vision- and speech-based system, and the second experiment with a Wizard-of-Oz setup where gaze tracking and automatic speech recognition were simulated by an experimenter behind the scenes. Each subject was asked to use all three modes to activate the speech recognizer and then to evaluate each mode.

For a natural `look-to-talk` interface, we needed a fast and reliable computer vision system to accurately track the users gaze. This need has limited gaze-based interfaces from being widely implemented in intelligent environments. However, fast and reliable gaze trackers using state-of-the-art vision technologies are now becoming available and are being used to estimate the focus of attention. In our prototype system, we estimate gaze with a 3-D gradient-based head-pose tracker [83] (an early of version of our head gaze tracker described in Section 3.1) that uses shape and intensity of the moving object. The tracker provides a good non-intrusive approximation of the user s gaze.

There were 13 subjects, 6 for the first experiment and 7 for the Wizard-of-Oz experiment. All of them were students in computer science, some of whom had prior experience with `talk-to-talk` in the I-Room. After the experiment, the subjects rated each of the three modes on a scale of one to five on three dimensions: ease of use, naturalness, and future use. We also asked the subjects to tell us which mode they liked best and why.

Each subject was asked three sets of five trivia questions, each set using a different

Mode	Activate	Feedback	Deactivate	Feedback
PTT	Switch the microphone to "on"	Physical status of the switch	Switch the microphone to "mute"	Physical status of the switch
LTT	Turn head toward Sam	Sam shows listening expression	Turn head away from Sam	Sam shows normal expression
TTT	Say computer	Special beep	Automatic (after 5 sec)	None

Table 2.1: How to activate and deactivate the speech interface using three modes: `push-to-talk` (PTT), `look-to-talk` (LTT), and `talk-to-talk` (TTT).

mode of interaction in counterbalanced order. In the Wizard-of-Oz experiment, we ran a fourth set in which all three modes were available, and the subjects were told to use any one of them for each question. Table 2.1 illustrates how users activate and deactivate the speech recognizer using the three modes and what feedback the system provides for each mode.

Results and Discussion

As we first reported in [72], for the first experiment, there was no significant difference (using analysis of variance at D=0.05) between the three modes for any of the surveyed dimensions. However, most users preferred `talk-to-talk` to the other two. They reported that `talk-to-talk` seemed more accurate than `look-to-talk` and more convenient than `push-to-talk`.

For the Wizard-of-Oz experiment, there was a significant difference in the naturalness rating between `push-to-talk` and the other two (p=0.01). This shows that, with better perception technologies, both `look-to-talk` and `talk-to-talk` will be better choices for natural human-computer interaction. Between `look-to-talk` and `talk-to-talk`, there was no significant difference on any of the dimensions. However, five out of the seven subjects reported that they liked `talk-to-talk` best compared to two subjects who preferred `look-to-talk`. One reason for preferring `talk-to-talk` to `look-to-talk` was that there seemed to be a shorter latency in `talk-to-talk` than `look-to-talk`. Also, a few subjects remarked that Sam seemed disconnected from the task, and thus it felt awkward to look at Sam.

Despite the subjects' survey answers, for the fourth set, 19 out of 30 questions were answered using `look-to-talk`, compared with 9 using `talk-to-talk` (we have this data for five out of the seven subjects; the other two chose a mode before beginning the fourth set to use for the entire set, and they each picked `look-to-talk` and `talk-to-talk`). When asked why he chose to use `look-to-talk` even though he liked `talk-to-talk` better, one subject answered " I just turned my head to answer and noticed that the Room was already in listening mode." This confirms the findings in [61] that users naturally look at agents before talking to them.

This user study about the `look-to-talk` concept shows that gaze is an important cue in multi-party conversation since human participants naturally turn their head toward the person, avatar or device they want to talk to. For an interactive system, to recognize this kind of behavior can help distinguish if the speaker is addressing the system or not.

2.2.2 Conversational Tooltips

A tooltip is a graphical user interface element that is used in conjunction with a mouse cursor. As a user hovers the cursor over an item without clicking it, a small box appears with a short description or name of the item being hovered over. In current user interfaces, tooltips are used to give some extra information to the user without interrupting the application. In this case, the cursor position typically represents an approximation of the user attention.

Visual tooltips are an extension of the concept of mouse-based tooltips where the user's attention is estimated from the head-gaze estimate. There are many applications for visual tooltips. Most museum exhibitions now have an audio guide to help visitors understand the different parts of the exhibition. These audio guides use proxy sensors to determine the location of the visitor or need input on a keypad to start the prerecorded information. Visual tooltips are a more intuitive alternative.

We define visual tooltips as a three-step process: deictic gesture, tooltip, and answer. During the first step, the system analyzes the user's gaze to determine if a specific object or region is under observation. Then the system informs the user

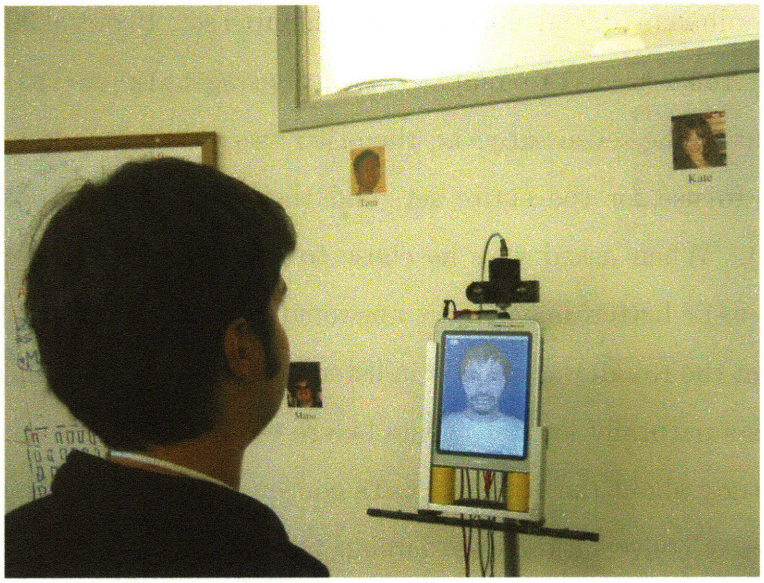


Figure 2-4: Multimodal kiosk built to experiment with *Conversational tooltip.* A stereo camera is mounted on top of the avatar to track the head position and recognize head gestures. When the subject looks at a picture, the avatar offers to give more information about the picture. The subject can accept, decline or ignore the offer for extra information.

about this object or region and offers to give more information. During the final step, if the user answers positively, the system gives more information about the object.

To work properly, the system that offers visual tooltips needs to know where the user attention is and if the user wants more information. A natural way to estimate the user's focus is to look at the user's head orientation. If a user is interested in a specific object, he or she will usually move his or her head in the direction of that object [98]. Another interesting observation is that people often nod or shake their head when answering a question. To test this hypothesis, we designed a multimodal experiment that accepts speech as well as vision input from the user. The following section describes the experimental setup and our analysis of the results.

Experimental Study

We designed this experiment with three tasks in mind: exploring the idea of visual tooltips, observing the relationship between head gestures and speech, and testing our head-tracking system. We built a multimodal kiosk that could provide information about some graduate students in our research group (see Figure 2-4). The kiosk consisted of a Tablet PC surrounded by pictures of the group members. A stereo camera [30] and a microphone array were attached to the Tablet PC.

The central software part of our kiosk consists of a simple event-based dialogue manager. The dialogue manager receives input from the Watson tracking library (described in Appendix A) and the speech recognition tools [71]. Based on these inputs, the dialogue manager decides the next action to perform and produces output via the text-to-speech routines [2] and the avatar [41].

When the user approaches the kiosk, the head tracker starts sending pose information and head nod detection results to the dialogue manager. The avatar then recites a short greeting message that informs the user of the pictures surrounding the kiosk and asks the user to say a name or look at a specific picture for more information. After the welcome message, the kiosk switches to listening mode (the passive interface) and waits for one of two events: the user saying the name of one of the members or the user looking at one of the pictures for more than n milliseconds. When the vocal command is used, the kiosk automatically gives more information about the targeted member. If the user looks at a picture, the kiosk provides a short description and offers to give more information. In this case, the user can answer using voice (yes, no) or a gesture (head nods and head shakes). If the answer is positive, the kiosk describes the picture, otherwise the kiosk returns to listening mode.

For our user study, we asked 10 people (between 24 and 30 years old) to interact with the kiosk. Their goal was to collect information about each member. They were informed about both ways to interact: voice (name tags and yes/no) and gesture (head gaze and head nods). There were no constraints on the way the user should interact with the kiosk.

Results and Discussion

10 people participated in our user study. The average duration of each interaction was approximately 3 minutes. At the end of each interaction, the participant was asked some subjective questions about the kiosk and the different types of interaction

(voice and gesture).

A log of the events from each interaction allowed us to perform a quantitative evaluation of the type of interaction preferred. The avatar gave a total of 48 explanations during the 10 interactions. Of these 48 explanations, 16 were initiated with voice commands and 32 were initiated with conversational tooltips (the user looked at a picture). During the interactions, the avatar offered 61 tooltips, of which 32 were accepted, 6 refused and 23 ignored. Of the 32 accepted tooltips, 16 were accepted with a head nod and 16 with a verbal response. Our results suggest that head gesture and pose can be useful cues when interacting with a kiosk.

The comments recorded after each interaction show a general appreciation of the conversational tooltips. Eight of the ten participants said they prefer the tooltips compared to the voice commands. One of the participants who preferred the voice commands suggested an on-demand tooltip version where the user asked for more information and the head gaze is used to determine the current object observed. Two participants suggested that the kiosk should merge the information coming from the audio (the yes/no answer) with the video (the head nods and head shakes).

This user study about "conversational tooltips" shows how head gaze can be used to estimate the user's focus of attention and presents an application where a conversation is initiated by the avatar when it perceives that the user is focusing on a specific target and may want more information.

2.2.3 Gaze Aversion

Eye gaze plays an important role in face-to-face interactions. Kendon proposed that eye gaze in two-person conversation offers different functions: monitor visual feedback, express emotion and information, regulate the flow of the conversation (turn-taking), and improve concentration by restricting visual input [51]. Many of these functions have been studied for creating more realistic ECAs [107, 109, 34], but they have tended to explore only gaze directed towards individual conversational partners or objects.

We define three types of distinctive eye motion patterns or "eye gestures": eye contact, deictic gestures, and non-deictic gestures. Eye contact implies one participant looking at the other participant. During typical interactions, the listener usually maintains fairly long gazes at the speaker while the speaker tends to look at the listener as he or she is about to finish the utterance [51, 70]. Deictic gestures are eye gestures with a specific reference which can be a person not currently involved in the discussion, or an object. Griffin and Bock showed in their user studies that speakers look at an object approximately 900ms before referencing it vocally [37]. Non-deictic gestures are eye movements to empty or uninformative regions of space. This gesture is also referred to as a *gaze-averting* gesture [35] and the eye movement of a thinker [86]. Researchers have shown that people will make gaze-averting gestures to retrieve information from memory [85] or while listening to a story [95]. Gaze aversion during conversation has been shown to be a function of cognitive load [35].

These studies of human-to-human interaction give us insight regarding the kind of gestures that could be useful for ECAs. Humans do seem to make similar gestures when interacting with an animated agent. Colburn *et al.* looked at eye contact with ECAs and found a correlation between the time people spend looking at an avatar versus the time they spend looking at another human during conversation [25].

We have observed that eye motions that attend to a specific person or object tend to involve direct saccades, while gaze aversion gestures tend to include more of a "wandering" eye motion. Looking at still images may be inconclusive in terms of deciding whether it is a gaze aversion gesture or a deictic eye movement, while looking at the dynamics of motion tends to be more discriminative (Figure 2-5). We therefore investigate the use of eye motion trajectory features to estimate gaze aversion gestures.

To our knowledge, no work has been done to study gaze aversion by human participants when interacting with ECAs. Recognizing such eye gestures would be useful for an ECA. A gaze aversion gesture while a person is thinking may indicate the person is not finished with their conversational turn. If the ECA senses the aversion gesture, it can correctly wait for mutual gaze to be re-established before taking its turn.

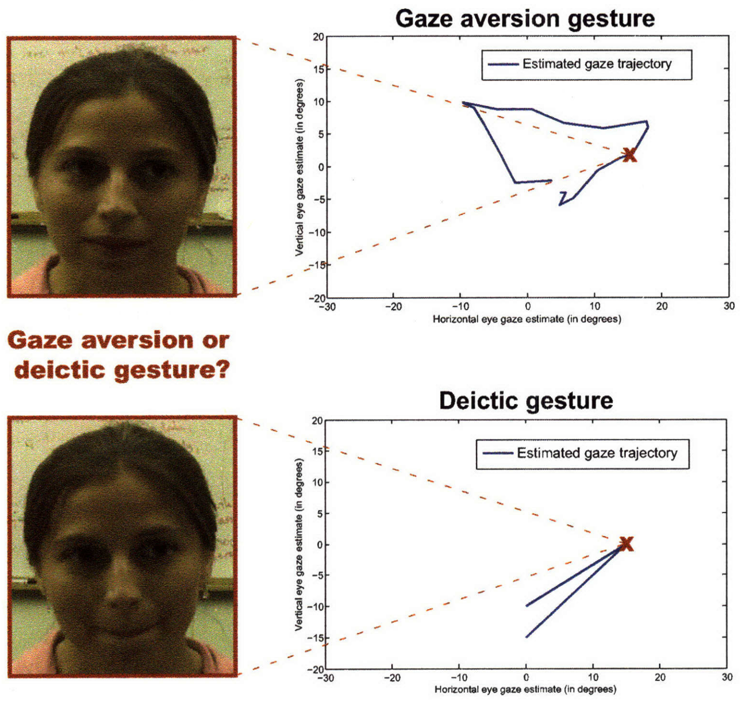


Figure 2-5: Comparison of a typical gaze aversion gesture (top) with a "deictic" eye movement (bottom). Each eye gesture is indistinguishable from a single image (see left images). However, the eye motion patterns of each gesture are clearly different (see right plots).

Experimental Study

Our user study was designed with two tasks in mind: (1) to observe the kind of eye gestures people make when interacting with an ECA, and (2) to evaluate how well we can recognize these gestures. The first task is the main topic of this section while the results of the second task are described later in Section 3.3 where we present our algorithm for gesture recognition.

For the purpose of this user study, we built a multimodal kiosk with an interactive avatar that can perform a survey of 100 questions (see Figure 2-6). Sample questions asked during the user study include:

1. Are you a student?

2. Is your age an even number?

3. Do you live on campus?

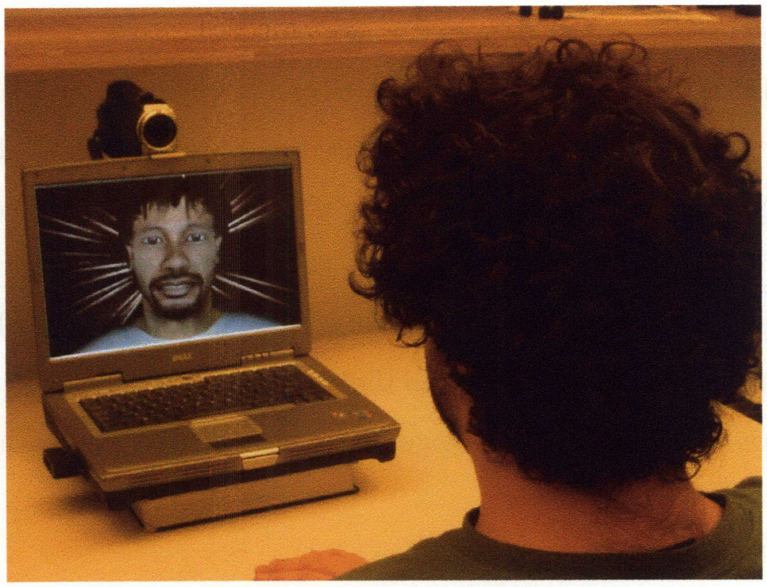


Figure 2-6: Multimodal interactive kiosk used during our user study.

4. Do you like Coke better than Pepsi?

5. Is one of the official languages in Canada Spanish?

6. Does Canada have a president?

7. Is fifteen minus five equal to nine?

8. Is five a prime number?

Our user study was composed of 6 participants: 2 men and 4 women, aged between 25-35 years old. Each interaction lasted approximately 10-12 minutes. At the beginning of the experiment, participants were informed that they would interact with an avatar who would ask them 100 questions. Participants were asked to answer every question with a positive answer (by saying "yes" and/or head nodding) or a negative answer (by saying "no" and/or head shaking) and were not aware that their eye gestures were being monitored[1].

[1]The main purpose of this user study was to observe the eye gaze patterns made by human participants. For this reason, we are not discussing in this thesis the performance of our head gestures recognizer for this user study but more details can be found in a related paper about co-adaptation [21].

The kiosk consisted of a 15.4" screen and a monocular camera with an integrated microphone placed on top of the screen. Participants sat in a chair placed 1.3 meters in front of the screen. The screen was elevated so that the eyes of the avatar were approximately at the same height as the eyes of the participant. The central software component of our kiosk consisted of a simple event-based dialogue manager that produced output using the AT&T text-to-speech engine [2] and the Haptek virtual avatar [41]. The experimenter, sitting to the right of each participant, used a remote keyboard to trigger the dialogue manager after each answer from each participant.

Results and Discussion

Since eye gaze gestures can be subtle and sometimes hard to differentiate even for a human, we asked three people to annotate the aversion gestures in the video sequences corresponding to each subject. The following definition was given to each coder for gaze aversion gestures: eye movements to empty or uninformative regions of space, reflecting "look-away" or "thinking".

Even though the user study did not include any explicit deictic reference to the environment around the user, human participants naturally made deictic eye gestures during the interactions. Most of these deictic gestures were targeted to the video camera or the experimenter sitting on the right side of the participant. Coders were instructed to label these deictic eye gestures as "non-gaze-aversion".

During the process of labeling and segmentation of the six video sequences from our user study, the three coders labeled 114, 72 and 125 gaze aversion gestures. The variation between coders is not that surprising since gaze gestures are subtle. We decided to define our ground truth as the intersection of all three coders, for a total of 72 gaze aversion gestures.

The average length of gaze aversion gestures was 1.3 seconds. Since all verbal answers from the users were a short "yes" or "no" response, waiting an extra two seconds for an answer may be too long for the embodied agent. Without any visual feedback recognition, the dialogue manager could potentially identify silence as a sign for misunderstanding and would repeat the question or ask the user if he/she understood the question.

On average, our six participants made gaze aversion gestures twelve times per interaction with a standard deviation of 6.8. Since 100 questions were asked during each interaction, on average 12% of the time, people made a gaze aversion gesture that was labeled by all three coders. In our experiment, most gaze aversions were gestures where the participant was thinking about their answer. Since our dialogue manager was relatively simple, few gaze aversion gestures had the purpose of keeping the avatar from interrupting the human participant (i.e. holding the floor). We anticipate that with a more complex dialogue manager and a better speech recognizer, human participants would express an even greater amount of gaze aversion gestures.

Our results suggest that people do make gaze aversion gestures while interacting with an ECA and that it would be useful for an avatar to recognize these patterns. A gaze aversion gesture while a person is thinking may indicate the person is not finished with their conversational turn. If the embodied agent senses the aversion gesture, it can correctly wait for mutual gaze to be re-established before taking its turn.

2.3 Physical Embodied Interfaces

2.3.1 Mel: Effects of Head Nod Recognition

Following the observation that human participants were head nodding naturally during the user study described in Section 2.1.2, this section presents a user study addressing the effects of head nod recognition with human-robot interaction[1].

Experimental Study

Participants held one of two conversations with the robot, one to demonstrate either its own abilities or to demonstrate collaboratively the IGlassware equipment. During these conversations people nodded at the robot, either because it was their means

[1]This work was done in collaboration with Candace Sidner, Christopher Lee and Clifton Forlines. This section excerpts from the paper originally published at HRI 2006 [92].

for taking a turn after the robot spoke (along with phrases such as "ok" or "yes" or "uh-huh"), or because they were answering in the affirmative a yes/no question and accompanied their linguistic "yes" or "ok" with a nod. Participants also shook their heads to answer negatively to yes/no questions, but we did not study this behavior because too few instances of "no" answers and head shakes occurred in our data. Sometimes a nod was the only response from the participant.

The robot was equipped with a stereo camera and head gaze was estimated using the AVAM approach described in Section 2.2.3 while head nod detection was performed using the discriminative approach described in [67]. The robot used the conversational system and architecture described in [84, 90]. Head nods were reported from the sensorimotor subsystem to the vision system in the same way that other gestures (for example, looking at an object) were measured.

A total of 49 participants interacted with the robot. None had interacted with our robot before. Most had never interacted with any robot. One participant had an abbreviated conversation because the robot mis-recognized the user's intention and finished the conversation without a complete demo. However, the participant's conversation was long enough to include in the study. Thirty-five participants held the IGlassware demo with the robot and fourteen participants held the self demo where Mel explains its own capabilities.

The participants were divided into two groups, called the MelNodsBack group and the MelOnlyRecognizesNods group. The MelNodsBack group had fifteen participants who were told that the robot understood some nods during conversation; they participated in a conversation in which the robot nodded back to the person every time it recognized a head nod. It should be noted that nodding back in this way is not something that people generally do in conversation. People nod to give feedback on what the other one has said, but having done so, their conversational partners only rarely nod in response. When they do, they are generally indicating some kind of mutual agreement. Nonetheless, by nodding back the robot gives feedback to the user on their behavior. Due to mis-recognition of nods, this protocol meant that the robot sometimes nodded when the person did not nod.

	IGlassware	Self	Total
MelNodsBack	9	6	15
MelOnlyRecognizesNods	6	8	14
NoMelNods	20	0	20

Table 2.2: Breakdown of participants in groups and demos.

The MelOnlyRecognizesNods group had fourteen subjects who held conversations without knowledge that the robot could understand head nods, although the nod recognition algorithms were operational during the conversation. We hypothesized that participants might be affected by the robot's nodding ability because 1) when participants nodded and spoke, the robot took another turn whereas without a response (verbal and nod), the robot waited a full second before choosing to go on and similarly, 2) when participants responded only with a nod, the robot took another turn without waiting further. Again mis-recognition of nods occurred, although the participants received no gestural feedback about it. The breakdown into groups and demo types is illustrated in Table 2.2.

These participants are in contrast to a base condition called the NoMelNods group, with 20 subjects who interacted with the robot in a conversation in which the robot did not understand nods, and the participants were given no indication that it could do so (see Section 2.1.2). This group, collected in 2003, held only the IGlassware equipment conversation with the robot.

Protocol for the study: The study was a between-subjects design. Each participant was randomly pre-assigned into one of the two nodding conditions (that is, no subjects had conversations in both nodding conditions). Video cameras were turned on after the participant arrived. The participant was introduced to the robot (as Mel) and told the stated purpose of the interaction (i.e. to have a conversation with Mel). Participants were told that they would be asked a series of questions at the completion of the interaction. Participants were also told what responses the robot could easily understand (that is, "yes", "no", "okay", "hello", "good bye", their first names, and "please repeat"), and in the case of the MelNodsBack condition, they were told that the robot could understand some of their nods, though probably not all. They were

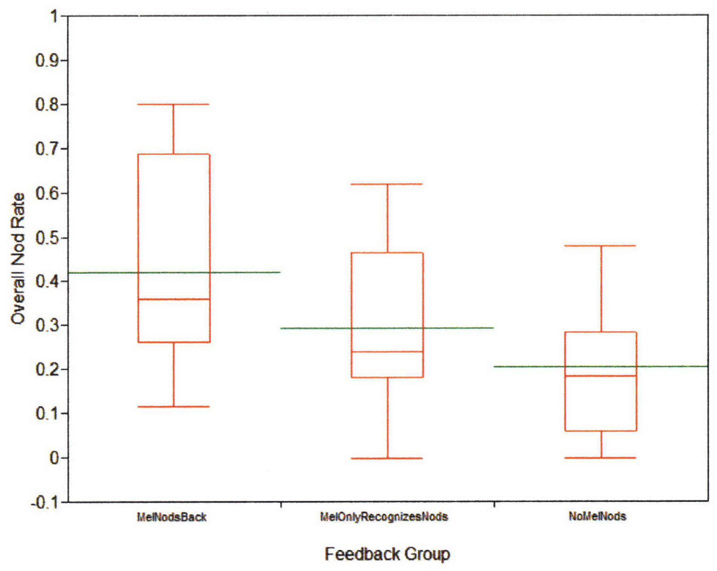


Figure 2-7: Overall Nod Rates by Feedback Group. Subjects nodded significantly more in the MelNodsBack feedback group than in the NoMelNods group. The mean Overall Nod Rates are depicted in this figure with the wide lines.

not told that the robot would nod back at them when they nodded. Participants in the 2003 study had been told the same material as the MelOnlyRecognizesNods participants.

When the robot was turned on, the participant was instructed to approach Mel. The interaction began, and the experimenter left the room. Interactions lasted between 3 to 5 minutes. After the demo, participants called in the experimenter and were given a short questionnaire, which was not relevant to the nodding study.

Results and Discussion

The study used a between-subjects design with Feedback Group as our independent variable, and Overall Nod Rate, Nod with Speech Rate, and Nod Only Rate as our three dependent variables. In total, 49 people participated in the study, fifteen in the MelNodsBack group, fourteen in the MelOnlyRecognizesNods group. An additional twenty participants served in the NoMelNods group.

A one-way ANOVA indicates that there is a significant difference among the three

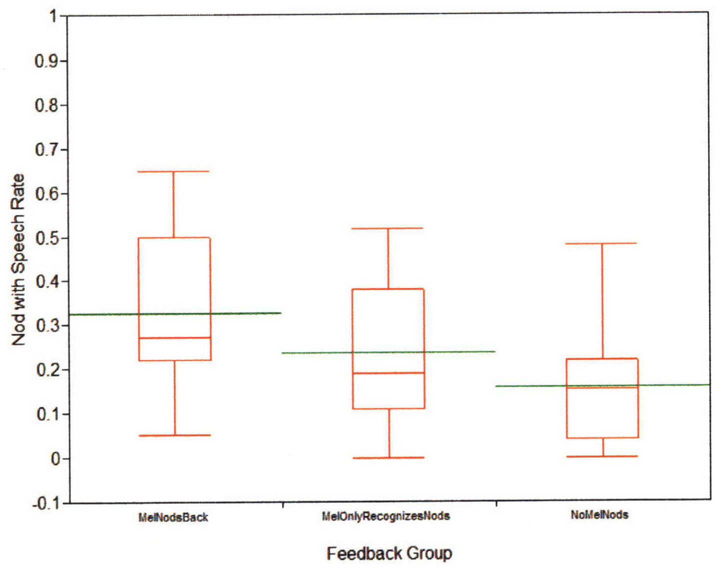


Figure 2-8: Nod with Speech Rates by Feedback Group. Again, subjects nodded with speech significantly more frequently in the MelNodsBack feedback group than in the NoMelNods group.

feedback groups in terms of Overall Nod Rate ($F_{2,46} = 5.52$, $p < 0.01$). The mean Overall Nod Rates were 42.3%, 29.4%, and 20.8% for MelNodsBack, MelOnlyRecognizesNods, and NoMelNods groups respectively. A post-hoc LSD pairwise comparison between all possible pairs shows a significant difference between the MelNodsBack and the NoMelNods groups (p=0.002). No other pairings were significantly different. The mean Overall Nod Rates for the three feedback groups are shown in Figure 2-7.

A one-way ANOVA indicates that there is also a significant difference among the three feedback groups in terms of Nod with Speech Rate ($F_{2,46} = 4.60$, $p = 0.02$). The mean Nod with Speech Rates were 32.6%, 23.5%, and 15.8% for the MelNodsBack, MelOnlyRecognizesNods, and NoMelNods groups respectively. Again, a LSD post-hoc pairwise comparison between all possible pairs of feedback groups shows a significant difference between the MelNodsBack and NoMelNods groups (p=0.004). Again, no other pairs were found to be significantly different. The mean Nod with Speech Rates for the three feedback groups are shown in Figure 2-8.

Finally, a one-way ANOVA found no significant differences among the three feed-

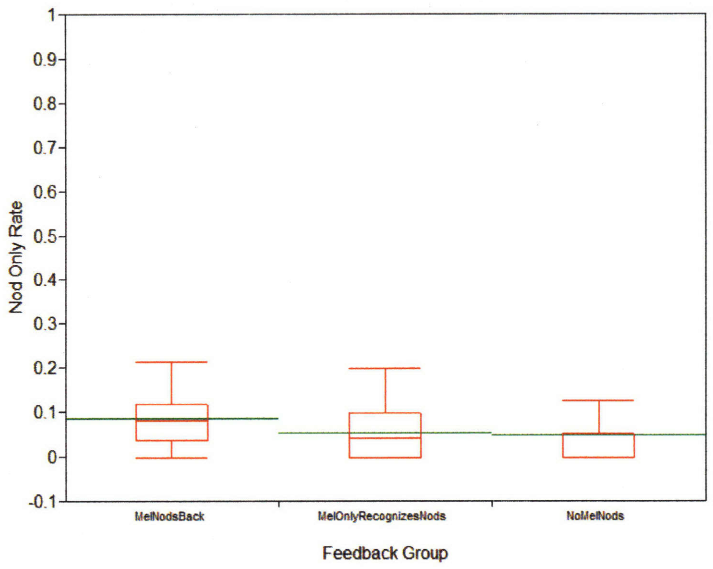


Figure 2-9: Nod Only Rates by Feedback Group. There were no significant differences among the three feedback groups in terms of Nod Only Rates.

back conditions in terms of Nod Only Rate ($F_{2,46} = 1.08$, $p = 0.35$). The mean Nod Only Rates were much more similar to one another than the other nod measurements, with means of 8.6%, 5.6% and 5.0% for the MelNodsBack, MelOnlyRecognizesNods, and NoMelNods groups respectively. The mean Nod Only Rates for the three feedback groups are shown in Figure 2-9.

These results above indicate that under a variety of conditions people will nod at a robot as a conversationally appropriate behavior. Furthermore, these results show that even subjects who get no feedback about nodding do not hesitate to nod in a conversation with the robot. We conclude that conversation alone is an important feedback effect for producing human nods, regardless of the robot's ability to interpret it.

The two statistically significant effects for nods overall and nods with speech that were found between the NoMelNods group and the MelNodsBack group indicate that providing information to participants about the robot's ability to recognize nods and giving them feedback about it makes a difference in the rate at which they produce nods. This result demonstrates that adding perceptual abilities to a humanoid robot that the human is aware of and gets feedback about provides a way to affect the outcome of the human and robot's interaction.

2.4 Non-embodied Interfaces

In this section, we explore how visual feedback can be used to improve interactive interfaces with no on-screen character. The most common examples of this type of interface are the graphical interfaces used by modern operating systems and personal computers. Instead of using an on-screen character, these interfaces rely on visual objects (widgets) like dialogue boxes to communicate a warning or ask a question. While the medium may be different, it would be interesting to find out if visual feedback can also be used for this type of interface. The following section presents our user study with gesture-based interactions.

2.4.1 Gesture-based Interactions

Head nods and head shakes are natural gestures commonly used during face-to-face interaction. Inspired by research with human-ECA interactions [90, 69], we propose using head gesture-based controls for two conventional windows interfaces: dialog boxes and document browsing.

Dialog boxes are special windows that are used by computer programs or by the operating system to display information to the user, or to get a response if needed [113]. We will focus our attention on two types of dialog boxes: *notification* dialog boxes and *question* dialog boxes.

Notification dialog boxes are one-button windows that show information from an application and wait for the user to acknowledge the information and click a confirmation button. During human-to-human interactions, the process of ensuring common understanding is called grounding [22]. Grounding is also present during interactions with embodied conversational agents, and human participants naturally head nod as a non-verbal feedback for grounding [69]. From these observations, we can expect human participants to naturally accept head nodding as a way to answer notification dialog boxes.

Question dialog boxes are two-button windows that display a question from the application and wait for positive or negative feedback from the user. This type of dialog box includes both confirmation and rejection buttons. If we look again at interactions that humans have with other humans or with embodied agents, head nods and head shakes are a natural way in many cultures to signify positive and negative feedback, so untrained users should be able to use these kinds of interfaces quite efficiently.

An interesting characteristic of notification and question dialog boxes is that quite often they appear while the user is performing a different task. For example, some email clients will notify the user of new email arrivals using a dialog box saying "You've got mail!". Another example is operating systems and applications that question the user about installing software updates. In both cases, the user may already be working on another task such as reading emails or browsing a document, and want to answer the dialog box without changing focus. Answering a dialog box using a head gesture makes it possible for users to keep keyboard and mouse focus undisturbed.

Based on our observations, we hypothesize that head gestures are a natural and efficient way to respond to dialog boxes, especially when the user is already performing a different task. We suggest a gesture-based interface design where notification dialog boxes can be acknowledged by head nodding and question dialog boxes can be answered by head nods or head shakes.

Similarly, people use head nods as a grounding cue when listening to information from another person. We conjecture that reading may be similar to listening to information, and that people may find it natural to use head nod gestures to turn pages. We design a prototype gesture-based page-forward control to browse a document and evaluate it in a user study as described below.

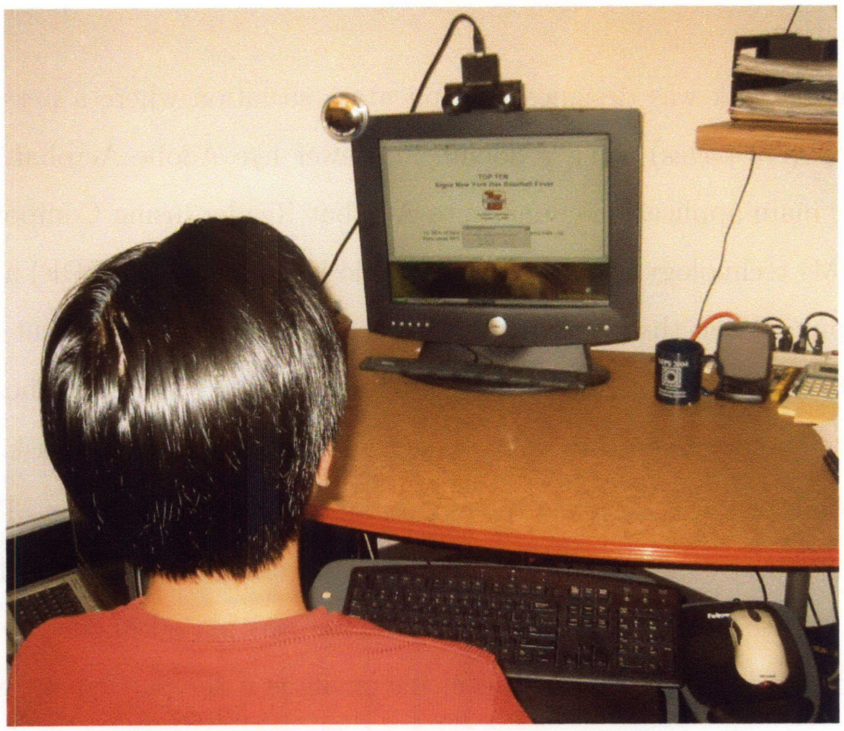


Figure 2-10: Experimental setup. A stereo camera is placed on top of the screen to track the head position and orientation.

Experimental Study

The physical setup consists of a desk with a 21" screen, a keyboard, and a mouse. A stereo camera was installed on top of the screen to track the head gaze and recognize head gestures (see Figure 2-10). This camera was connected to a laptop that ran the head gesture recognition system. The recognition system sends recognition results to the main application, which is displayed on the desktop screen in a normal fashion. No feedback about the recognition results is shown on this screen.

We designed our experimental system to evaluate the two gesture-based widgets described in Section 2.4.1: dialog box answering and document browsing. The main experiment consisted of two tasks: (1) reading a short text and (2) answering three related questions. Both tasks were performed under three different experimental interaction phases: conventional input only, head gesture input only and user-selected input method. For each interaction, the text and questions were different. During both tasks, dialog boxes would appear at different times asking a question or stating new information.

The reading task was designed to replicate a situation where a person reads an informal text ($\sim$3 pages) using a document viewer like Adobe Acrobat Reader. At startup, our main application connects to Acrobat Reader, using Component Object Model (COM) technology, displays the Portable Document File (PDF) and waits for the user input. When the participant reached the end of the document, he/she was instructed to close Acrobat Reader and automatically the window for the second task would start. The document browsing widget was tested during this task.

The writing task was designed to emulate an email writing process. The interface was similar to most email clients and included the conventional fields: "To:", "CC:", "Subject:" and the email body. A "Send" button was placed in the top left corner. The questions were already typed inside the email as if the participant was replying to a previous email.

The dialog boxes appearing during both tasks were designed to replicate reminders sent by a calendar application (i.e. Microsoft Outlook), alerts sent by an email client, and questions asked during impromptu moments about software updates or assistant help. Four to eight dialog boxes would appear during each experiment. The position and text displayed on the dialog box changed between appearances. Participants were asked to answer each dialog box that appeared on the screen. Two types of dialog boxes were displayed: one "OK" button and two "Yes/No" buttons.

Both tasks were repeated three times with three different experimental interaction phases. During the first interaction, the participants were asked to use the mouse or the keyboard to browse the PDF document, answer all dialog boxes and reply to the email. This interaction phase was used as a baseline where participants were introduced to both tasks and they could remember how it feels to interact with conventional input devices.

Between the first and second interaction, a short tutorial about head gestures for user interface was performed where participants practiced the new techniques for dialog box answering and document browsing as described in Section 2.4.1. Participants were free to practice it as long as they wanted, but most participants were ready to start the second phase after one minute.

During the second phase, participants were asked to browse the PDF document and answer dialog boxes using head nods and head shakes. During the email task, participants had to use the keyboard for typing and could use the mouse for navigating in the email, but they were asked to answer any dialog box with a head gesture. This interaction phase was designed to introduce participants to gesture-based widgets.

During the third phase of the experiment, participants were told that they could use any input technique to perform the browsing and email tasks. This interaction was designed so that participants could freely choose between keyboard, mouse, or head gestures. In contrast to the previous two phases, this phase should give us an indication of which interaction technique or combination of techniques is preferred.

This phase was also designed to compare the accuracy of the head recognizer with the judgement of a human observer. For this reason, during this third phase of the experiment a human observer was recognizing intentional head nods from each participant in a Wizard-of-Oz manner. The vision-based head gesture recognizer was still running during this phase and its results were logged for later comparison.

The study was a within-subject design, where each participant performed more than one interaction phase. A total of 19 people participated in our experiment. All participants were accustomed to using the keyboard and mouse as their main input devices and none of them had used head gesture in a user-interface before. Twelve participants completed the first two conditions and only seven participants completed all three conditions because of a problem with our log files. Each condition took two to three minutes to complete on average. All participants completed a short questionnaire about their experience and preference at the end of the experiment.

The short questionnaire contained two sets of questions where participants were asked to compare keyboard, mouse, and head gestures. The first set of questions focused on document browsing while the second set addressed dialog box answering. Both sets had the same structure: two questions about efficiency and natural interaction followed by a section for general comments. Each question asked the participant to grade all three types of user interfaces (keyboard, mouse, and head gesture) from

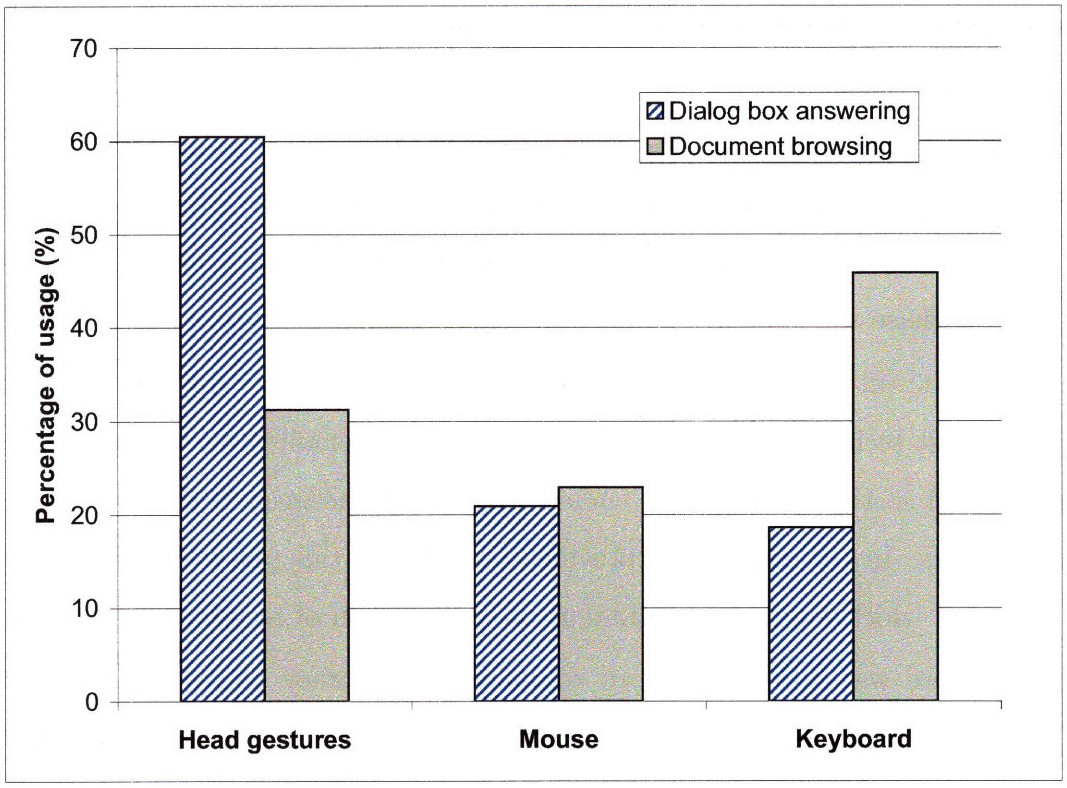


Figure 2-11: Preferred choices for input technique during third phase of the experiment.

1 to 5 where 5 is the highest score. The first question asked participants to grade the input techniques on efficiency. The second question asked participants to grade the input techniques on how natural it was.

Results and Discussion

In this section, we first present the results of the user study, discuss its implication, and finally present results about context-based recognition.

We analyzed the choices each participant made during the third phase of the experiment. During this part of the experiment, the participant was free to decide which input device to use. Figure 2-11 shows how participants decided to answer dialog boxes and browse documents. For the dialog boxes, 60.4% of the time they used a head gesture to answer the dialog box while using mouse and keyboard only 20.9% and 18.6% respectively. For document browsing, 31.2% of the time they used a head gesture to answer the dialog box while using mouse and keyboard only 22.9% and 45.8% respectively.

Using a standard analysis of variance (ANOVA) on all 7 subjects who participated to the third phase, results on the dialog box answering widget showed a significant difference among the means of the three input techniques: $p = 0.060$. Pairwise comparisons show a significant difference for pairs gesture-mouse and gesture-keyboard, with respectively $p = 0.050$ and $p = 0.083$, while the pair mouse-keyboard showed no significant difference: $p = .45$. Pairwise comparisons for the document browsing show no significant difference between all pairs, with $p = 0.362$, $p = 0.244$, and $p < 0.243$ for gesture-mouse, gesture-keyboard, and mouse-keyboard respectively.

We compared the results from vision-based head gesture recognizer with the Wizard-of-Oz results on three participants. The vision-based system recognized 91% of the head nods with a false positive rate of 0.1. This result shows that a vision-only approach can recognize intentional head gestures, but suggests the use of contextual information to reach a lower false positive rate. Context-based recognition is discussed later in Chapter 4.

We also measured the qualitative results from the questionnaire. Figure 2-12 shows how 19 participants scored each input device for efficiency and naturalness when interacting with dialog boxes. The average scores for efficiency were 3.6, 3.5 and 4.2, for keyboard, mouse, and head gestures respectively. In the case of naturalness, the average scores were 3.4, 3.7 and 4.2.

Figure 2-13 shows how 19 participants scored each input device for efficiency and naturalness for document browsing. The average scores for efficiency were 4.3, 3.8 and 2.6, for keyboard, mouse and head gestures respectively. In the case of naturalness, the average scores were 4.1, 3.9 and 2.7.

One important fact when analyzing this data is that our participants were already trained to use mouse and keyboard. This previous training affected their choices. The results from Figures 2-11 and 2-12 suggest that head gestures are perceived as a natural and efficient way to answer and acknowledge dialog boxes. Participants did not seem to appreciate as much the head gesture for document browsing. Some

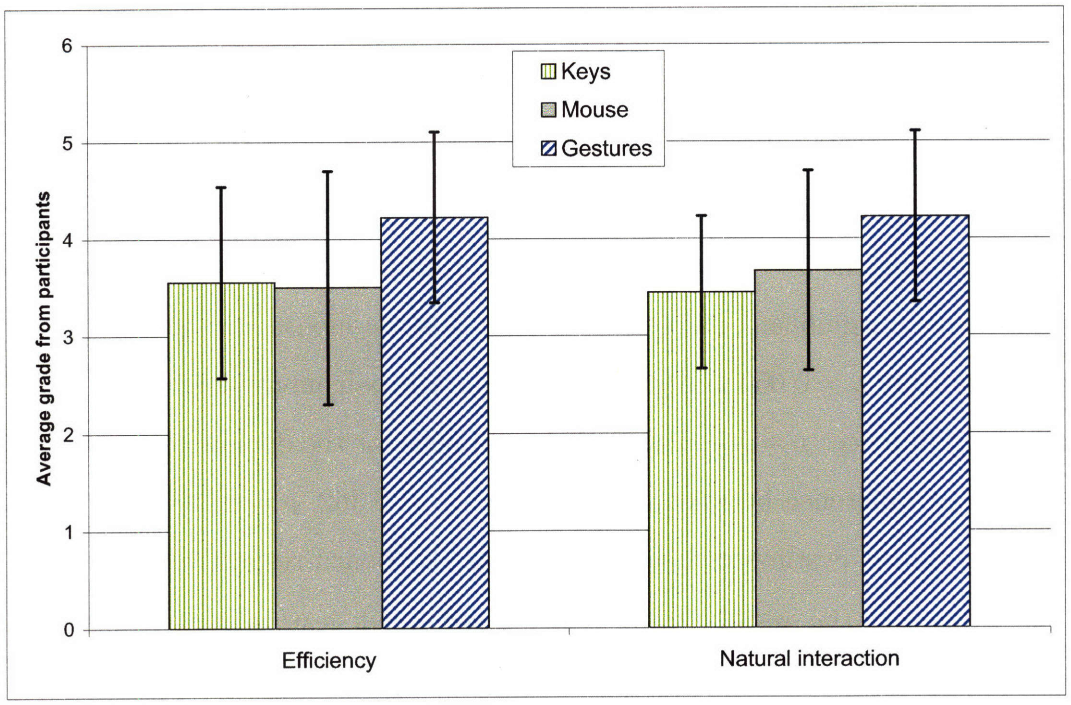


Figure 2-12: Survey results for dialog box task. All 19 participants graded the naturalness and efficiency of interaction on a scale of 1 to 5, 5 meaning best.

participants stated in their questionnaire that they wanted to have a more precise control over the scrolling of PDF documents. Since the head gesture paradigm only offered control at the page level, we think that this paradigm would apply better to a slide-show application like PowerPoint.

An interesting fact that came from our post-analysis of the user study is that some participants performed head shakes at the notification dialog box (the dialog box with only "OK"). This probably means that they did not want to be disturbed at that specific moment and expressed their disapproval by a head shake.

This user study with gesture-based interactions shows that head nods and head shakes can be useful visual feedback for non-embodied interfaces. Human participants naturally used head gestures over conventional input devices like keyboard and mouse when answering dialog boxes. In the following section, we summarize our findings about visual feedback recognition for interactive interfaces.

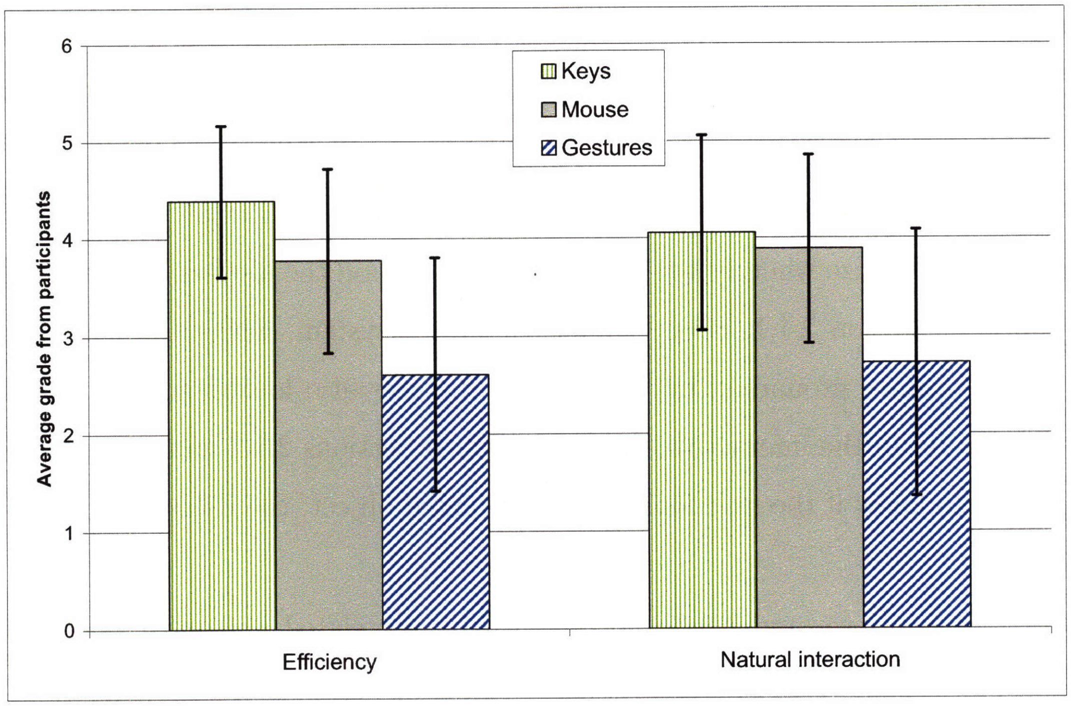


Figure 2-13: Survey results for document browsing task. All 19 participants graded the naturalness and efficiency of interaction on a scale of 1 to 5, 5 meaning best.

2.5 User Study Summary

We are interested in visual feedback that human participants perform naturally when interacting with an interactive system. We also want to find out what visual feedback, if recognized properly by the system, would make the interactive interface more useful and efficient.

The user studies presented in this chapter give us a better understanding of what facial feedback can be useful for interactive systems. In this section, we investigated four types of visual feedback: head gaze, eye gaze, head gestures and eye gestures.

Head Gaze As shown in the Look-to-Talk experiment and in previous work [61, 109], gaze is an important cue in multi-party conversation since human participants naturally turn their head toward the person, avatar or device they want to talk to. For an interactive system, to recognize this kind of behavior can help distinguish if the speaker is addressing the system or not. During a meeting scenario, an embodied agent like Sam (see Section 2.2.1) can help the participants when directly triggered. Also, head gaze can determine if the speaker is talking to the avatar or to another person.

Head gaze is also an important cue for estimating the attention of the user and can help to recognize whether the participant understood the last explanation. Mack, described in Section 2.1.1, is an example of such a system, where the head gaze is used to determine grounding by observing if the user also looked at the map when pointing to it. In the interactive system described Sections 2.1.2 and 2.3.1, Mel uses head gaze to know if the user looked at the pointed object, the iGlassware.

Eye Gaze Eye gaze and head gaze are usually correlated and most observations made for head gaze also apply to eye gaze. Estimating both head and eye gaze can be used by an interactive system to know when someone is talking to it. When trying to estimate the focus of attention of the user, it is particularly useful to estimate the eye gaze of the participant if the targets are close to each other (i.e. small field-of-view).

The "conversational tooltips" experiment described in Section 2.2.2 suggests that head gaze can be used to estimate the focus of interest of the user. If the targets were closer to each other, the use of eye gaze would definitely improve the accuracy of the system.

Head Gestures Head nodding is a natural gesture for grounding. Even when interacting with an avatar who cannot recognize head nods, human participants did perform head nods (see Section 2.1.2). The user study described in Section 2.3.1 shows that people head nod more often when the interactive interface is able to perceive gestures and gives feedback of it awareness. This result demonstrates that adding perceptual abilities to a humanoid robot that the human is aware of and gets feedback about provides a way to affect the outcome of the human-robot interaction.

In our experiment with gesture-based interactions (see Section 2.4.1), we showed that head nods and head shakes can be useful for non-embodied interfaces. Human participants naturally used head gestures over conventional input devices like keyboard and mouse when answering dialog boxes.

Eye Gestures In Section 2.2.3, we presented a user study where human participants naturally performed gaze aversion gestures when interacting with an embodied agent. A gaze aversion gesture while a person is thinking may indicate the person is not finished with their conversational turn. If the embodied agent senses the aversion gesture, it can correctly wait for mutual gaze to be re-established before taking its turn.

Chapter 3

Visual Feedback Recognition

Visual feedback such as head nodding and gaze aversion are naturally performed by human participants when interacting with an embodied agent. As discussed in the previous chapter, the recognition of visual feedback can improve the performance of embodied and non-embodied interactive interfaces. In this chapter, we describe our algorithms for accurate, online head gaze, eye gaze, head gesture and eye gesture recognition using a monocular or stereo camera.

Estimating head gaze accurately for an extended period of time without drifting is a great challenge. In Section 3.1, we present a new Adaptive View-based Appearance Model (AVAM) that can be acquired online during tracking and used to accurately estimate head gaze over a long period of time[1]. The main novelty of our approach relies on the fact that estimating the pose of a newly acquired frame will also improve the quality of the view-based appearance model. Given that the head gaze path crosses itself during tracking, our AVAM model will be able to estimate the user gaze with bounded drift.

In Section 3.2, we present our approach for eye gaze estimation based on a pre-acquired view-based appearance model. The eye appearance model was built using eye images from 16 subjects looking at 35 different targets under different lighting conditions. With our approach, eye gaze estimation can be performed from low

[1]This work was done in collaboration with Ali Rahimi. It was originally published at CVPR 2003 [66].

resolution images. Our approach is user independent and can handle glasses and changes in lighting condition.

In Section 3.3, we introduce a new algorithm for visual gestures recognition, the *Frame-based Hidden Conditional Random Field* (FHCRF). Given the estimated head gaze and eye gaze, our FHCRF model can accurately recognize and discriminate visual gestures like head nodding and gaze aversion from other natural gestures. Our discriminative model learns both sub-gesture patterns and the dynamics between gestures to achieve better performance. In our results we demonstrate that using the FHCRF model for visual gesture recognition outperforms models based on Support Vector Machines (SVMs), Hidden Markov Models (HMMs), and Conditional Random Fields (CRFs).

3.1 Head Gaze Tracking

Accurate drift-free head gaze tracking is an important goal of many computer vision applications. Traditional models for frame-to-frame head gaze tracking accumulate drift even when viewing a previous pose. In this section we present an adaptive view-based appearance model that can use existing two-frame registration algorithms to track objects over long distances with bounded drift.

The adaptive view-based appearance model maintains views (key frames) of the object under various poses. These views are annotated with the pose of the head, as estimated by the tracker. Tracking against the appearance model entails registering the current frame against previous frames and all relevant key frames. The adaptive view-based appearance model can be updated by adjusting the pose parameters of the key frames, or by adding or removing key frames. These online updates are non-committal so that further tracking can correct earlier mistakes induced into the model.

View-based models can capture non-Lambertian reflectance (e.g. specular reflection on a shinny surface) in a way that makes them well suited for tracking rigid bodies. We show that our appearance model has bounded drift when the object's pose trajectory crosses itself. We compare our pose tracking results with the orientation estimate of an *Inertia Cube*^{2} inertial sensor [46]. On a Pentium 4 3.2GHz, our tracker implementation runs at 25Hz.

The following section discusses previous work on head gaze tracking. In Section 3.1.2, we describe our adaptive view-based appearance model. In Section 3.1.3, we show how to recover the pose (translation and rotation) of the head given the current AVAM. In Section 3.1.4, we show how we populate and adjust the appearance model after each frame was tracked. Section 3.1.5 presents the monocular- and stereo-view registration algorithms tested with our AVAM model. Then, in Section 3.1.6 we discuss our head gaze tracking experiments. The generality of our approach is demonstrated by tracking the 6 degree-of-freedom (DOF) pose of an object from another object class.

3.1.1 Related Work

Many techniques have been proposed for tracking a user's head based on passive visual observation. To be useful for interactive environments, tracking must be accurate enough to localize a desired region, robust enough to cope with illumination and scene variation, and fast enough to serve as an interactive controller. Examples of 2-D approaches to face tracking include color-based [115], template-based [52] and eigenface-based [39] techniques.

Techniques using 3-D models have greater potential for accurate tracking but require knowledge of the shape of the face. Early work presumed simple shape models (e.g., planar [10], cylindrical [54], or ellipsoidal [5]). Tracking can also be performed with a 3-D face texture mesh [88] or 3-D face feature mesh [114].

Very accurate shape models are possible using the active appearance model methodology [26], such as was applied to 3-D head data in [11]. However, tracking 3-D active appearance models with monocular intensity images is currently a time-consuming process, and requires that the trained model be general enough to include the class of tracked users.

Many different representations have been used for tracking objects based on aggregate statistics about the subject, or they can be generative rendering models for the appearance of the subject. Trackers which model the appearance of the subject using aggregate statistics of their appearance include [9] and [74]. These use the distribution of skin-color pixels to localize the head; the distribution can be adapted to fit the subject as tracking goes on. To recover pose, these techniques rely on characteristics of the aggregate distribution, which is influenced by many factors, only one of which is pose. Thus the tracking does not lock onto the target tightly.

Graphics-based representations model the appearance of the target more closely, and thus tracking can lock onto the subject more tightly. Textured geometric 3D models [55, 5] can represent the target under different poses. Because the prior 3D shape models for these systems do not adapt to the user, they tend to have limited tracking range.

Deformable 3D models fix this problem by adapting the shape of the model to the subject [48, 63, 20, 29]. These approaches maintain the 3D structure of the subject in a state vector which is updated recursively as images are observed. These updates require that correspondences between features in the model and features in the image be known. Computing these correspondences reliably is difficult, and the complexity of the update grows quadratically with the number of 3D features, making the updates expensive [48].

Linear subspace methods have been used in several face tracking systems. [40] models the change in appearance due to lighting variations and affine motion with a subspace representation. The representation is acquired during a separate training phase, with the subject fronto-parallel to the camera, and viewed under various lighting conditions. Cootes and Taylor track using a linear subspace for shape and texture [27]. The manifold underlying the appearance of an object under varying poses is highly non-linear, so these methods work well with relatively small pose changes only.

In this section, we introduce the Adaptive View-based Appearance Model for head gaze tracking. Our model represents the subject with a subset of the frames seen so far in the input sequence. These key frames are annotated with their estimated pose, and collectively represent the appearance of the subject as viewed from these estimated poses. Our algorithm runs online, operates without prior training, and does not use an approximate shape model for the subject.

3.1.2 Adaptive View-Based Appearance Model

Our adaptive view-based model consists of a collection of pose-annotated key frames acquired using a stereo camera during tracking (Figure 3-1). For each key frame, the view-based model maintains the following information:

$$\mathcal{M}_s = \{I_s, Z_s, x_s\}$$

where I_s and Z_s are the intensity and depth images associated with the key frame s. It is important to note that the depth image Z_s can be estimated either directly from a stereo camera or from a 3D model. The latter approach is favorable when tracking using monocular images. Section 3.1.5 describes our algorithm for view registration with a 3D ellipsoid model of the head.

The adaptive view-based model is defined by the set $\{\mathcal{M}_1 \dots \mathcal{M}_k\}$, where k is the number of key frames. We model the pose of each key frame as a Gaussian random variable whose distribution is refined during the course of tracking. $x_s = [\ T^x \quad T^y \quad T^z \quad \Omega^x \quad \Omega^y \quad \Omega^z\ ]$ is a 6 dimensional vector consisting of the translation and the three euler angles, representing the mean of each random variable. Although in this chapter we use a rigid body motion representation for the pose of each frame, any representation, such as affine, or translational, could be used. The view-based model also maintains the correlation between these random variables in a matrix $\Lambda_{\mathcal{X}}$, which is the covariance of these poses when they are stacked up in a column vector.

While tracking, three adjustments can be made to the adaptive view-based model: the tracker can correct the pose of each key frame, insert or remove a key frame.

Adding new frames into this appearance model entails inserting a new $\mathcal{M}_s$ and upgrading the covariance matrix. Traditional 3D representations, such as global mesh models, may require expensive stitching and meshing operations to introduce new frames.

Adaptive view-based models provide a compact representation of objects in terms of the pose of the key frames. The appearance of the object can be tuned by updating a few parameters. In Section 3.1.3, we show that our model can be updated by solving a linear system of the order of the number of key frames in the model. On the other hand, 3D mesh models require that many vertices be modified in order to affect a significant change in the object representation. When this level of control is not necessary, a 3D mesh model can be an expensive representation.

View-based appearance models can provide robustness to variation due to non-Lambertian reflectance. Each point on the subject is exposed to varying reflectance conditions as the subject moves around (see Figure 3-1). The set of key frames which contains these points capture these non-Lambertian reflectances. Representing similar non-Lambertian reflectance with a 3D model is more difficult, requiring that an albedo model be recovered, or the texture be represented using view-based textures.

The following section discusses how to track rigid objects using our adaptive view-based appearance model.

3.1.3 Tracking and View-based Model Adjustments

In our framework, tracking and pose adjustments to the adaptive view-based model are performed simultaneously. As the tracker acquires each frame, it seeks to estimate the new frame's pose as well as that of the key frames, using all data seen so far. That is, we want to approximate the posterior density:

$$p(x_t, x_{\mathcal{M}}|y_{1..t}), \tag{3.1}$$

where x_t is the pose of the current frame, $y_{1..t}$ is the set of all observations from the registration algorithm made so far, and $x_{\mathcal{M}}$ contains the poses of the key frames in the view-based model, $x_{\mathcal{M}} = \{x_1 \ldots x_k\}$.

Each incoming frame (I_t, Z_t) is registered against several base frames. These base frames consist of key-frames chosen from the appearance model, and the previous

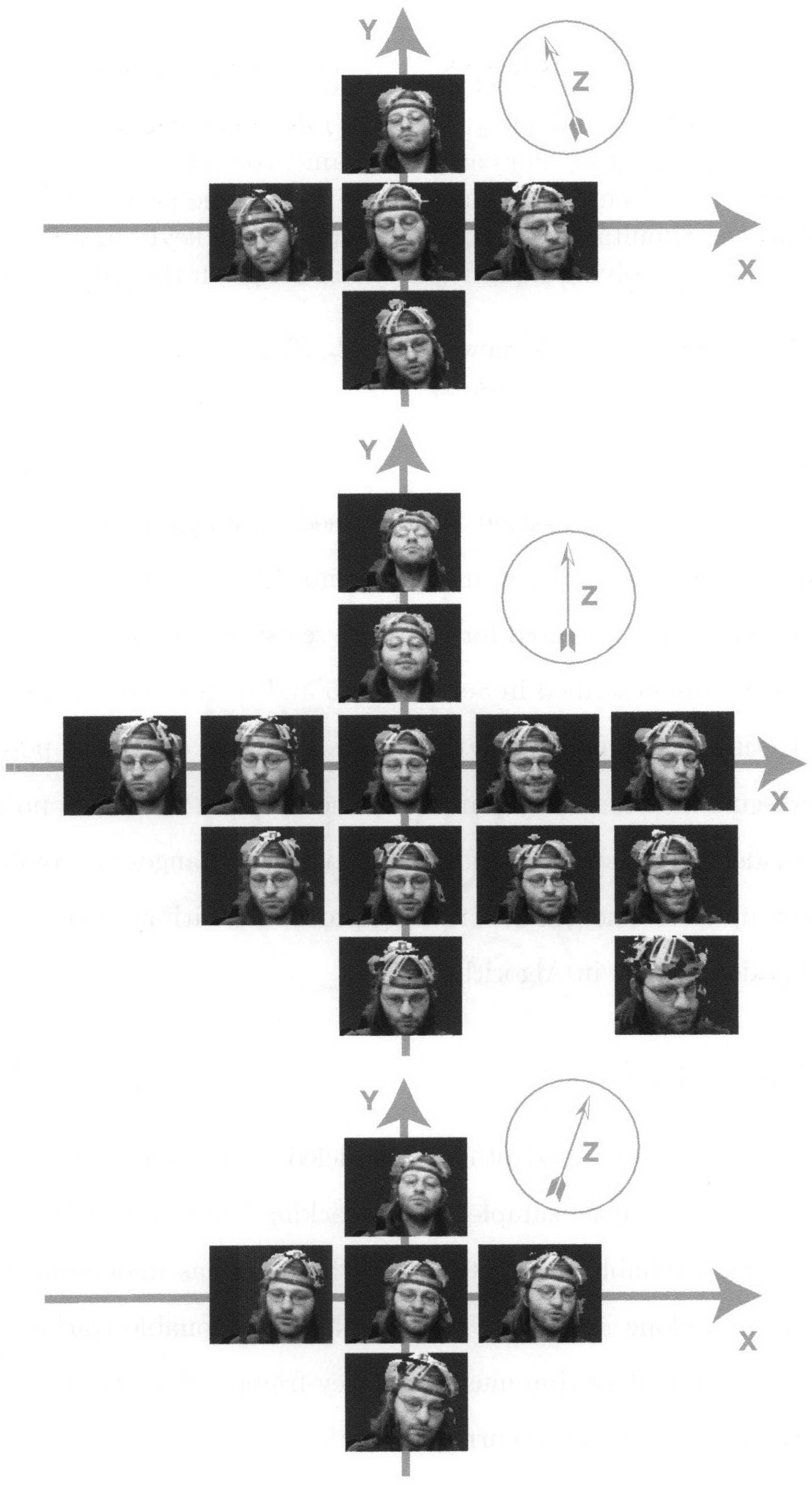


Figure 3-1: The view-based model represents the subject under varying poses. It also implicitly captures non-Lambertian reflectance as a function of pose. Observe the reflection in the glasses and the lighting difference when facing up and down.

Algorithm 1 Tracking with an Adaptive View-based Appearance Model

```
for each new frame $(I_t, Z_t)$ do
    Base frame selection: Select the $n_b$ closest keyframes from the model and include the previous frame $(I_{t-1}, Z_{t-1}, x_{t-1})$ as a base frame
    Pair-wise registration: For each base frame, compute the relative transformation $y_s^t$ between the current frame and the base frame (see Section 3.1.5)
    Pose uptate: Simultaneously update the pose of all keyframes and compute the current pose $x_t$ by solving Equations 3.3 and 3.4 given the pair-wise registrations $\{y_s^t\}$
    Keyframe selection: Add new frame $(I_t, Z_t, x_t)$ to the keyframe model if no keyframe exists around this pose $x_t$
end for
```

frame (I_{t-1}, Z_{t-1}). The registration is performed only against key-frames that are likely to yield sensible pose-change measurements. Note that with monocular cameras, the intensity image I_t is used for pair-wise registration using the ellipsoid-based registration algorithm described in Section 3.1.5 and then the depth image Z_t is created by projecting the ellipsoid into the image using the estimated pose. The next subsections discuss how these key-frames are chosen, then show how pose-change estimates are modelled as Gaussians and finally how pose-changes are combined using a Gauss-Markov update. The generic tracking procedure with an adaptive view-based appearance model is given in Algorithm 1.

Selecting Base Frames

Occlusions, lighting effects, and other unmodeled effects limit the range of many registration algorithms. For example, when tracking heads, our 6 DOF registration algorithm returns a reliable pose estimate if the head has undergone a rotation of at most 10 degrees along any axis. Thus to obtain reasonable tracking against the appearance model, the algorithm must select key-frames whose true poses are within tracking range of the pose of the current frame.

To find key-frames whose pose is similar to the current frame, we look for key-frames whose appearance is similar to that of the current frame. This assumes that the primary factor governing appearance is pose. We compute the change in appearance between those two images, and tentatively accept a key-frame as a base frame if the change in their appearances is within a threshold. To compute the appearance distance, the intensity images of the frames are aligned with respect to translation. The L2 distance between the resulting images is used as the final appearance distance.

This scheme works well if there is a one-to-one mapping between appearance and pose. Head gaze estimation usually fall in this category. In some situations, different poses may yield the same appearance. This happens with objects with repetitive textures, such as floor tiles [87] or a calibration cube all of whose sides look identical. To disambiguate these situations, key-frames that sufficiently resemble the current frame are chosen as base frames only if their pose is likely to be within tracking range of the current frame [66].

Modelling Results Pairwise Registration

Once suitable base frames have been chosen from the view model, a registration algorithm computes their pose difference with respect to the current frame (see Section 3.1.5 for our monocular- and stereo-view registration algorithms). In this section, we model the observation as the true pose difference between two frames, corrupted by Gaussian noise. This Gaussian approximation is used in the following section to combine pose-change estimates to update the distribution of Equation 3.1.

The registration algorithm operates on the current frame (I_t, Z_t), which has unknown pose x_t and a base frame (I_s, Z_s) acquired at time s, with pose x_s. It returns an observed pose-change estimate y_s^t. We presume that this pose-change is probabilistically drawn from a Gaussian distribution $\mathcal{N}(y_s^t | x_t - x_s, \Lambda_{y|xx})$. Thus pose-changes are assumed to be additive and corrupted by Gaussian noise.

We further assume that the current frame was generated by warping a base frame and adding white Gaussian noise. Under these circumstances, if the registration algorithm reports the mode of

$$\epsilon(y_s^t) = \sum_{d \in R} \| I_t(d + u(d; y_s^t)) - I_s(d) \|^2,$$

where d is the 2d-coordinates of a pixel in region of interest R and $u(d; y_s^t)$ is the image-based flow, then the result of [82] can be used to fit a Gaussian noise model to the reported pose-change estimate. Using Laplace's approximation, it can be shown that the likelihood model for the pose-change $x_t - x_s$ can be written as $\mathcal{N}(y_s^t | x_t - x_s, \Lambda_{y|xx})$, where

$$\Lambda_{y|xx} = \frac{1}{\epsilon(y_s^t)} \frac{\partial}{\partial^2 y^2} \epsilon(y_s^t). \tag{3.2}$$

Updating Poses

This section shows how to incorporate a set of observed pose-changes into the posterior distribution of Equation 3.1. By assuming that these observations are the true pose-change corrupted by Gaussian noise, we can employ the Gauss-Markov equation.

Suppose that at time t, there is an up-to-date estimate of the pose x_{t-1} and of the frames in the model, so that $p(x_{t-1}, x_{\mathcal{M}} | y_{1..t-1})$ is known. Denote the new pose-change measurements as $y_{1..t} = \{y_{1..t-1}, y_{t-1}^t, y_{M_1}^t, y_{M_2}^t, \ldots\}$, where $M_1, M_2, \ldots$ are the indices of key frames selected as base frames. We would like to compute $p(x_t, x_{\mathcal{M}} | y_{1..t})$.

The update first computes a prior for $p(x_t | y_{1..t-1})$ by propagating the marginal distribution for $p(x_{t-1} | y_{1..t-1})$ one step forward using a dynamical model. This is similar to the prediction step of the Kalman filter.

The variables involved in the update are x_t, the previous frame pose x_{t-1} and the key-frames chosen as base frames x_{M_1}, x_{M_2}, etc. These are stacked together in a variable $\mathcal{X}$:

$$\mathcal{X} = \begin{bmatrix} x_t & x_{t-1} & x_{M_1} & x_{M_2} & \cdots \end{bmatrix}^\top .$$

The covariance between the components of $\mathcal{X}$ is denoted by $\Lambda_{\mathcal{X}}$. The rows and columns of $\Lambda_{\mathcal{X}}^{old}$ corresponding to the poses of the key-frames are mirrored in $\Lambda_{\mathcal{M}}$. Together, $\mathcal{X}$ and $\Lambda_{\mathcal{X}}$ completely determine the posterior distribution over the pose of the key-frames, the current frame, and the previous frame.

Following the result of the previous section, a pose-change measurement y_s^t between the current frame and a base frame in $\mathcal{X}$ is modeled as having come from:

$$\begin{aligned} y_s^t &= C\mathcal{X} + \omega, \\ C &= \begin{bmatrix} I & 0 & \cdots & -I & \cdots & 0 \end{bmatrix}, \end{aligned}$$

where ω is Gaussian with covariance $\Lambda_{y|xx}$. Each pose-change measurement y_s^t is used to update all poses using the Kalman Filter update equation:

$$[\Lambda_{\mathcal{X}}^{new}]^{-1} = [\Lambda_{\mathcal{X}}^{old}]^{-1} + C^\top \Lambda_{y|xx}^{-1} C \tag{3.3}$$

$$\mathcal{X}_{new} = \Lambda_{\mathcal{X}}^{new} \left([\Lambda_{\mathcal{X}}^{old}]^{-1} \mathcal{X}_{old} + C^\top \Lambda_{y|xx}^{-1} y_s^t \right) \tag{3.4}$$

After individually incorporating the pose-changes y_s^t using this update, $\mathcal{X}_{new}$ is the mean of the posterior distribution $p(x_t, x_{t-1}, M | y_{1..t})$ and $\mathrm{Cov}[\mathcal{X}_{new}]$ is its variance. This distribution can be marginalized by picking out the appropriate elements of $\mathcal{X}_{new}$ and $\Lambda_{\mathcal{X}}^{new}$.

Now that we have shown how to estimate the pose of the current frame given an adaptive view-based appearance model, in the following section we demonstrate how to populate and modify the AVAM after each pose estimation.

3.1.4 Acquisition of View-Based Model

This section describes an online algorithm for populating the view-based model with frames and poses. After estimating the pose x_t of each frame as per the updates of previous section, the tracker decides whether the frame should be inserted into the appearance model as a key-frame.

A key-frame should be available whenever the object returns to a previously visited pose. To identify poses that the object is likely to revisit, the 3 dimensional space of rotations is tesselated into adjacent regions maintaining a representative key-frame. Throughout tracking, each region is assigned the frame that most likely belongs to it. This ensures that key-frames provide good coverage of the pose space, while retaining only those key-frames whose pose can be determined with high certainty.

The probability that a particular frame belongs to a region centered at x_r is:

$$\Pr[x_t \in B(x_r)] = \int_{x \in B(x_r)} \mathcal{N}(x|x_t, \Lambda_t)dx, \tag{3.5}$$

where $B(x)$ is the region centered around a location x, and Λ_t is the pose covariance of the current frame and can be read from $\Lambda_{\mathcal{M}}$.

If this frame belongs to a region with higher probability than any other frame so far, it is the best representative for that region, and so the tracker assigns it there. If the frame does not belong to any region with sufficiently high probability, or all regions already maintain key-frames with higher probability, the frame is discarded.

The above criteria exhibit three desirable properties: 1) frames are assigned to regions near their estimated pose, 2) frames with low certainty in their pose are penalized, because the integral of a Gaussian under a fixed volume decreases with the variance of the Gaussian, and 3) key-frames are replaced when better key-frames are found for a given region.

When a frame is added to the appearance model, $\mathcal{X}$ and $\Lambda_{\mathcal{M}}$ must be updated. This involves creating a new slot as the last element of $\mathcal{X}$ and moving the first component of $\mathcal{X}$ (which corresponds to x_t) to that slot. These changes are similarly reflected in $\Lambda_{\mathcal{M}}$. The slot for x_t in $\mathcal{X}$ is initialized to zero. This new x_t is initially assumed to be very uncertain and independent of all frames observed so far, so its corresponding rows and columns in $\Lambda_{\mathcal{M}}$ are set to zero. Following these operations, the updates from the previous section can be applied to subsequent frames.

3.1.5 Pair-wise View Registration

As discussed in Section 3.1.3, our adaptive view-based appearance model relies on a view registration algorithm to estimate the relative motion between two frames. Given the current frame and a base frame, the registration algorithm estimates the pose change y_s^t between these frames. In the following two subsections we describe two registration algorithms. The first algorithm uses as input both intensity and depth images (computed from stereo images) to estimate the transformation between two frames. The second algorithm uses intensity images and a simple ellipsoid 3D model of the head to estimate the relative motion using images from a monocular camera.

If stereo information is available, the stereo-view registration algorithm has several advantages to the monocular algorithm. The stereo-based approach is potentially more robust to motion in the background (e.g., a second person moving in the background) since the segmentation provided from the depth image can easily be used to remove such outliers. Also, the depth information helps distinguish between small rotation and small translation. Most of the experiments in this thesis were done using the stereo-view registration algorithm.

Stereo-view Registration Algorithm

The registration parameters are computed in several steps: First the centers of mass of the regions of interest $\mathcal{R}_t$ and $\mathcal{R}_s$ are aligned in 3D translation. This translational component is then refined using 2D cross-correlation in the image plane.. Finally, a finer registration algorithm [64] based on Iterative Closest Point (ICP) [19, 6] and the Brightness Constancy Constraint Equation (BCCE) [44] is applied. ICP finds corresponding points between two 3D point clouds and tries to minimize the error (usually the euclidian distance) between the matched points. BCCE provides a constraint on the motion parameters between two intensity images without requiring correspondences.

The correlation step provides a good initialization point for the iterative ICP and BCCE registration algorithm. Centering the regions of interest reduces the search window of the correlation tracker, making it more efficient.

The ICP algorithm iteratively computes correspondences between points in the depth images and finds the transformation parameters which minimize the distance between these pixels. By using depth values obtained from the range images, the BCCE can also be used to recover 3D pose-change estimates [42]. Combining these approaches is advantageous because BCCE registers intensity images whereas ICP is limited to registering range imagery. In addition, we have empirically found that

BCCE provides superior performance in estimating rotations, whereas ICP provides more accurate translation estimates.

To combine these registration algorithms, their objective functions are summed and minimized iteratively. At each step of the minimization, correspondences are computed for building the ICP cost function. Then the ICP and BCCE cost functions are linearized, and the locally optimal solution is found using a robust least-squares solver [45]. This process usually converges within 3 to 4 iterations. For more details, see [64].

Monocular-view Registration Algorithm

We initialize the head gaze tracker using using a 2D face detector based on Adaboost [110]. The 3D ellipsoid model is then fit to the face based on the width of the detected face and the camera focal length. From this model, the depth image for the first frame Z_0 can be computed by applying ray-tracing to the 3D model.

Given a base frame I_s, Z_s and the current image I_t (note that we can't compute the depth Z_t until we compute the pose for this frame), the transformation between the two frames is estimated using a two step process as described in the previous section. First, correlation is used to estimate the 2D translation between the base frame and the current frame. This initialization point is then use as a start point for our iterative version of BCCE (also known as the Normal Flow Constraint [97]).

At each iteration, the transformation between the two frames is computed by minimizing the BCCE and then the base frame is warped for the next iteration. The iterative algorithm stop when the maximum number of iterations is reached or if the correlation between the warped base frame and the current frame is smaller then a threshold. The position and orientation of the ellipsoid model are updated based on the estimated motion and the depth image Z_t is computed using ray-tracing.

3.1.6 Experiments

This section presents three experiments where our adaptive view-based appearance model is applied to tracking objects undergoing large movements in the near-field (~1m) for several minutes. All three experiments use the stereo-based registration algorithm (described in Section 3.1.5) to track the object and create an appearance model. The monocular view registration algorithm was tested independently during the user study with gaze aversion described in Section 2.2.3. In the first experiment, we compare qualitatively 3 approaches for head pose tracking: differential tracking, first frame as keyframe and our adaptive view-based model. In the second experiment, we present a quantitative analysis of our view-based tracking approach by comparing with an inertial sensor *Inertia Cube*^{2} . Finally, we show that the view-based appearance model can track general objects including a hand-held puppet. All the experiments were done using a Videre Design stereo camera [30].

Head Pose Tracking

We tested our view-based approach with sequences obtained from a stereo camera [30] recording at 5Hz. The tracking was initialized automatically using a face detector [110]. The pose space used for acquiring the view-based model was evenly tesselated in rotation. The registration algorithm used about 2500 points per frame. On a Pentium 4 3.2GHz, our C++ implementation of the complete rigid object tracking framework, including frame grabbing, 3D view registration and pose updates, runs at 25Hz.

Figure 3-2 shows tracking results from a 2 minute test sequence. In this sequence the subject performed head rotations of up-to 110 degrees and head translations of up-to 80cm, including some translation along the Z axis. We compared our view-based approach with a differential tracking approach which registers each frame with its previous frame, concatenating the pose changes. To gauge the utility of multiple key-frames, we show results when the first frame in the sequence is the only key-frame.

The left column of Figure 3-2 shows that the differential tracker drifts after a

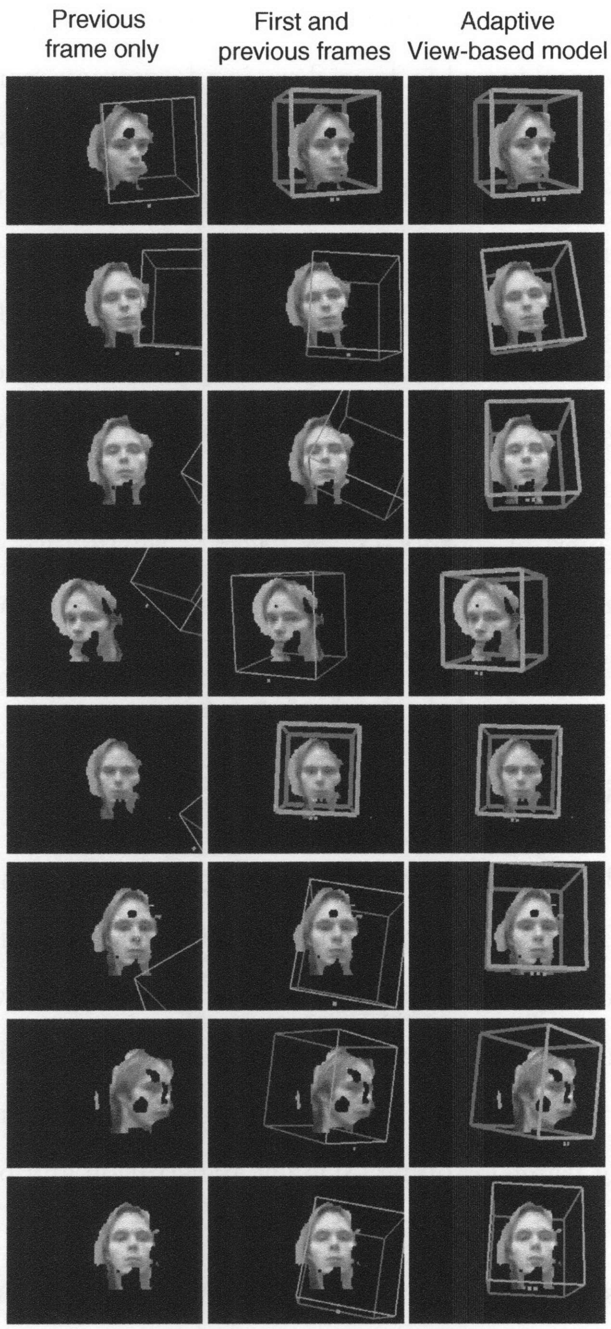


Figure 3-2: Comparison of face tracking results using a 6 DOF registration algorithm. Rows represent results at 31.4s, 52.2s, 65s, 72.6, 80, 88.4, 113s and 127s. The thickness of the box around the face is inversely proportional to the uncertainty in the pose estimate (the determinant of x_t). The number of indicator squares below the box indicate the number of base frames used during tracking.

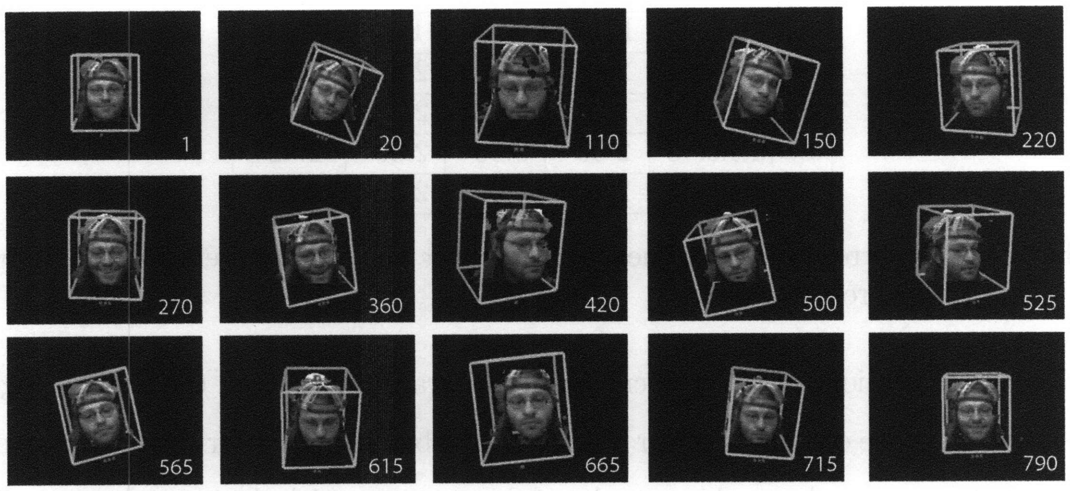


Figure 3-3: Head pose estimation using an adaptive view-based appearance model.

short while. When tracking with only the first frame and the previous frame (center column), the pose estimate is accurate when the subject is near-frontal but drifts when moving outside this region. The view-based approach (right column) gives accurate poses during the entire the sequence for both large and small movements. In this test sequence the tracker typically chose 2 to 3 base frames (including the previous frame) to estimate pose.

Ground Truth Experiment

To analyze quantitatively our algorithm, we compared our results to an *Inertia Cube*^{2} sensor from InterSense[46]. *Inertia Cube*^{2} is an inertial 3-DOF (Degree of Freedom) orientation tracking system. The sensor was mounted on the inside structure of a construction hat. Since the *Inertia Cube*^{2} estimates orientation by sensing gravity and the earth's magnetic field, its orientation estimates along the X and Z axis (where Z points outside the camera and Y points up) are mostly driftless; however, its estimates along the Y axis can suffer from drift. InterSense reports an absolute pose accuracy of 3°RMS when the sensor is moving.

We recorded 4 sequences with ground truth poses using the *Inertia Cube*^{2} sensor. The sequences were recorded at 6 Hz and have an average length of 801 frames ($\sim$133sec). During recording, subjects performed head rotations of up-to 125 degrees

	Pitch	Yaw	Roll	**Total**
Sequence 1	2.88°	3.19°	2.81°	2.97°
Sequence 2	1.73°	3.86°	2.32°	2.78°
Sequence 3	2.56°	3.33°	2.80°	2.92°
Sequence 4	2.26°	3.62°	2.39°	2.82°

Table 3.1: RMS error for each sequence. Pitch, yaw and roll represent rotation around X, Y and Z axis, respectively.

and head translations of up-to 90cm, including translation along the Z axis. Figure 3-3 shows the pose estimates of our adaptive view-based tracker for the first sequence. Figure 3-4 compares the tracking results of this sequence with the inertial sensor. The RMS errors for all 4 sequences is shown in Table 3.1. Our results demonstrate that our tracker is accurate to within the resolution of the *Inertia Cube*^{2} sensor.

General Object Tracking

Since our tracking approach doesn't use any prior information about the object when using the stereo-based view registration algorithm, our algorithm works on different object classes without changing any parameters. Our last experiment uses the same tracking technique described in this chapter to track a puppet. The position of the puppet in the first frame was defined manually. Figure 3-5 presents the tracking results.

Our results show that using adaptive view-based appearance model we can robustly track the head pose over a large range of motion. As we will see in the following section, for eye gaze estimation, the range of motion is smaller and a pre-acquired view-based appearance model is sufficient for estimating eye gaze.

3.2 Eye Gaze Estimation

Our goal is to estimate eye gaze during multimodal conversation with an embodied agent. To ensure a natural interaction, we want a recognition framework with the following capabilities:

- User-independent

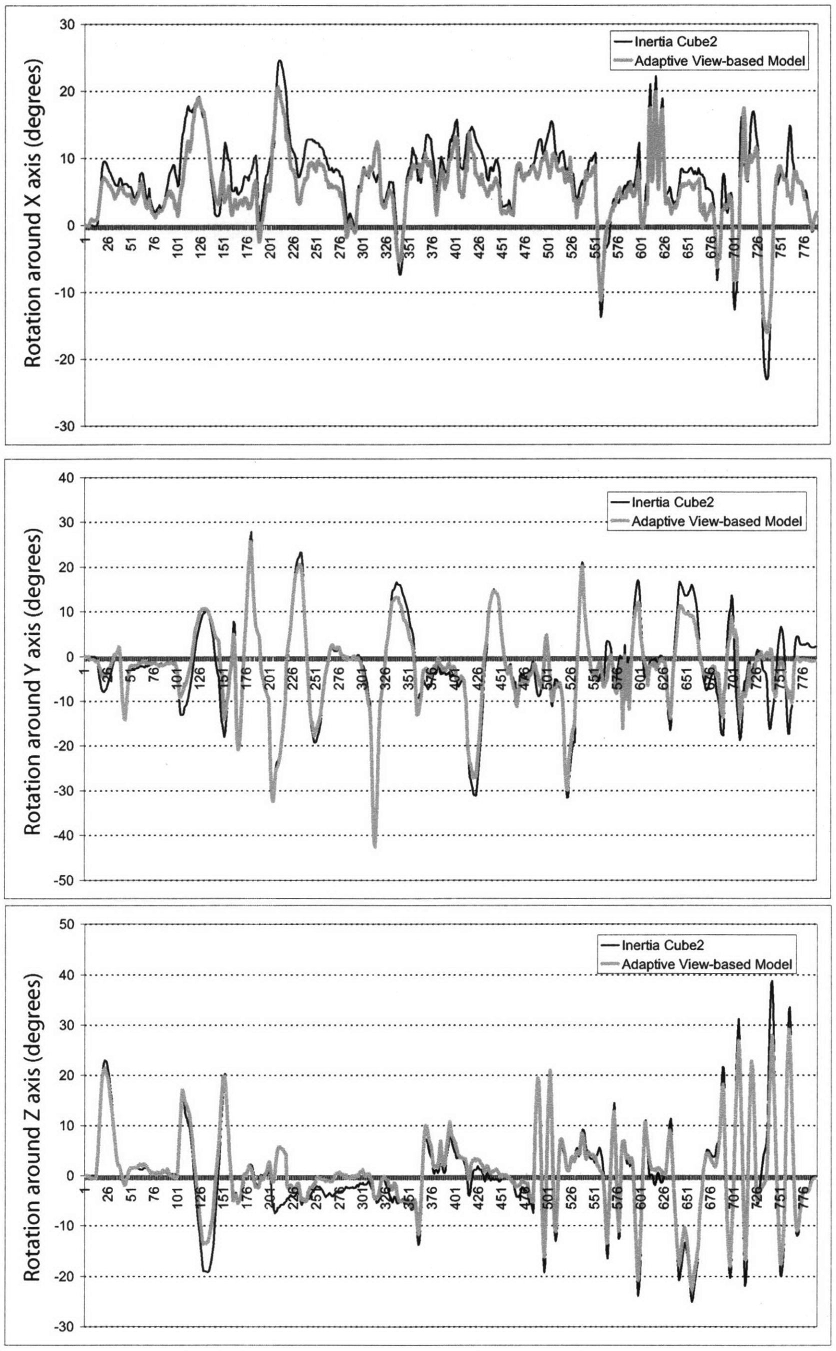


Figure 3-4: Comparison of the head pose estimation from our adaptive view-based approach with the measurements from the *Inertia Cube*^{2} sensor.

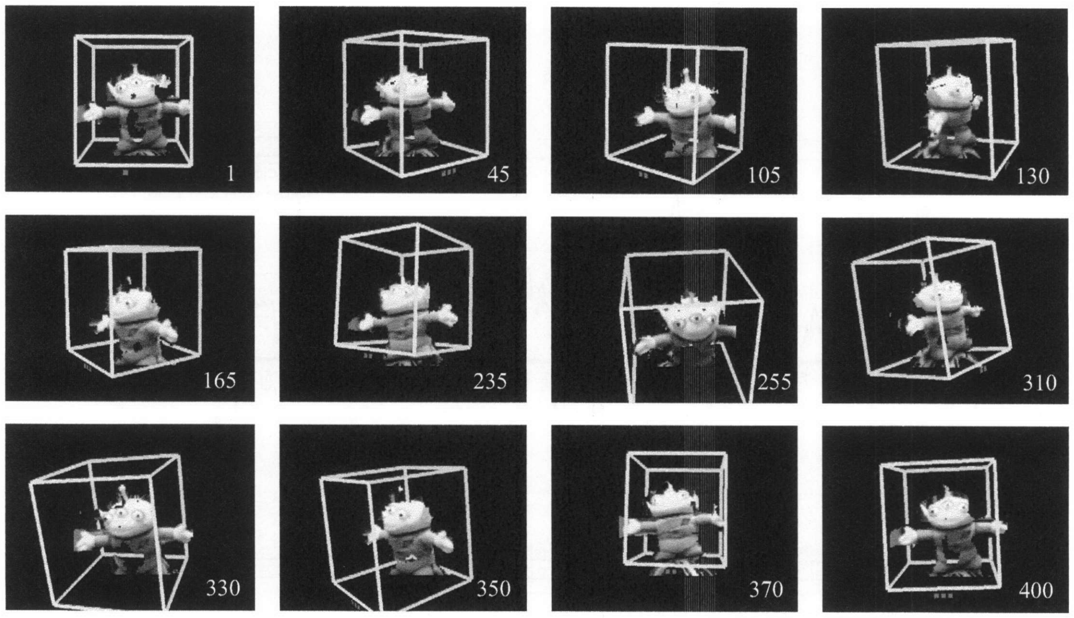


Figure 3-5: 6-DOF puppet tracking using the adaptive view-based appearance model.

- Non-intrusive
- Automatic initialization
- Robust to eye glasses
- Works with monocular cameras
- Works with low resolution images
- Takes advantage of other cues (e.g., head tracking)

Eye gaze tracking has been a research topic for many years [117, 62]. Some recent systems can estimate the eye gaze with an accuracy of less than a few degrees; these video-based systems require high resolution images and usually are constrained to small fields of view (4x4 cm) [73, 18]. Many systems require an infra-red light and filtered camera [118, 43]. In this section we develop a passive vision-based eye gaze estimator sufficient for inferring conversational gaze aversion cues; as suggested in [116], we can improve tracking accuracy by integration with head pose tracking.

As discussed earlier in Chapter 2, one of our goals is to recognize eye gestures that help an ECA to differentiate when a user is thinking from when a user is waiting

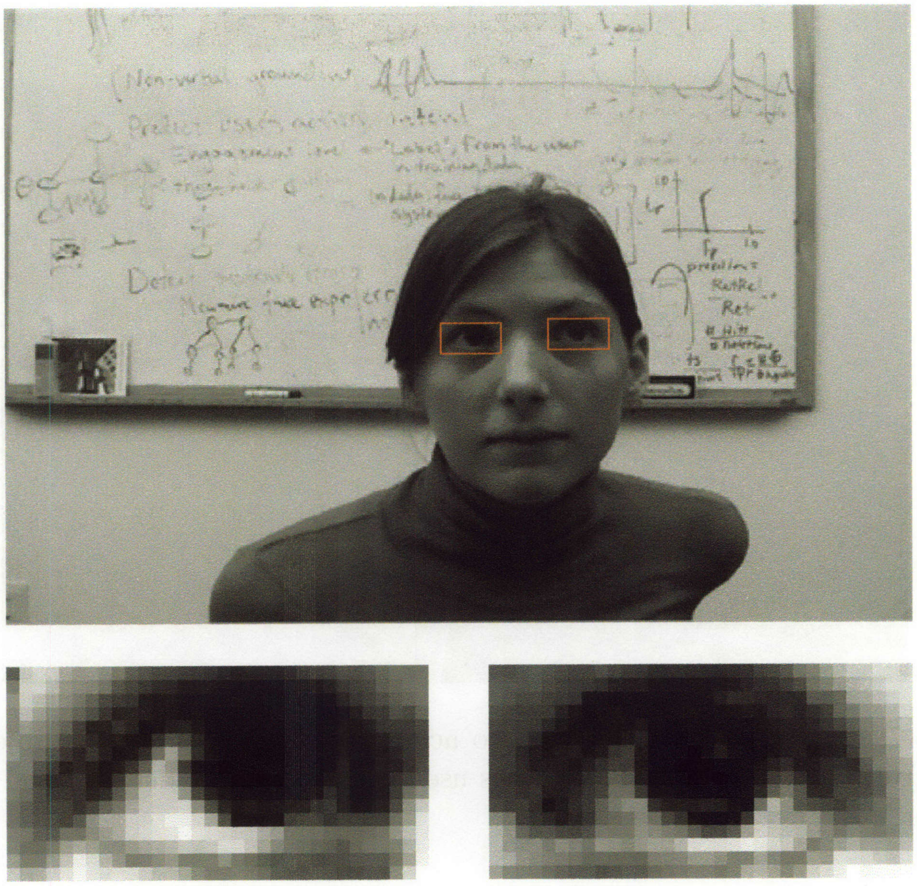


Figure 3-6: Example image resolution used by our eye gaze estimator. The size of the eye samples are 16x32 pixels.

for more information from the ECA. We built an eye gaze estimator that produces sufficient precision for gesture recognition and works with a low-resolution camera. Figure 3-6 shows an example of the resolution used during training and testing.

Our approach for eye gaze estimation is a three-step process: (1) detect the location of each eye in the image using a cascade of boosted classifiers, (2) track each eye location over time using a head pose tracker, and (3) estimate the eye gaze based on a view-based appearance model.

3.2.1 Eye Detection

For eye detection, we first detect faces inside the entire image and then search inside the top-left and top-right quarters of each detected face for the right and left eyes, respectively. Face and eye detectors were trained using a cascaded version

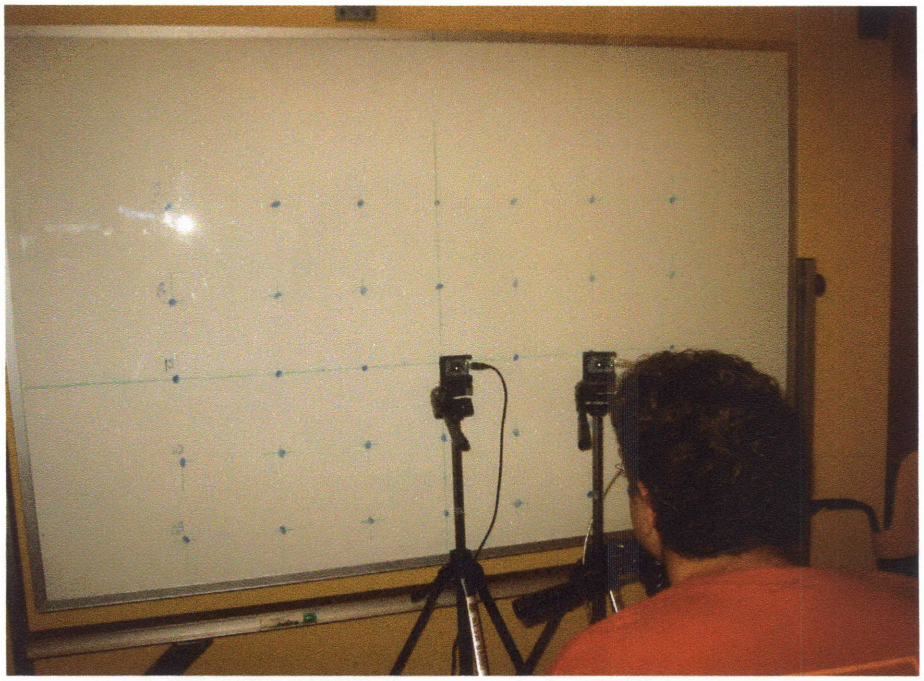


Figure 3-7: Experimental setup used to acquire eye images from 16 subjects with ground truth eye gaze. This dataset was used to train our eye detector and our gaze estimator.

of Adaboost [110]. For face detection, we used the pre-trained detector from Intel OpenCV.

To train our left and right eye detectors, we collected a database of 16 subjects looking at targets on a whiteboard. This dataset was also used to train the eye gaze estimator described in Section 3.2.3. A tripod was placed 1 meter in front of the whiteboard. Targets were arranged on a 7x5 grid so that the spacing between each target was 10 degrees (see Figure 3-7). The top left target represented an eye direction of -30 degrees horizontally and +20 degrees vertically. Two cameras were used to image each subject: one located in front of the target (0,0) and another in front of the target (+20,0).

Participants were asked to place their head on the tripod and then look sequentially at the 35 targets on the whiteboard. A keyboard was placed next to the participant so that he/she could press the space bar after looking at a target. The experiment was repeated under 3 different lighting conditions (see Figure 3-8). The

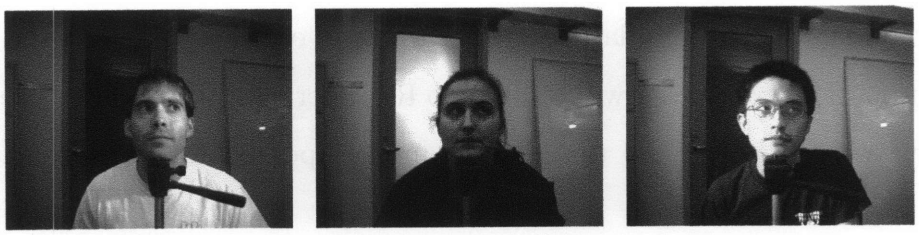


Figure 3-8: Samples from the dataset used to train the eye detector and gaze estimator. The dataset had 3 different lighting conditions.

location and size of both eyes were manually specified to create the training set. Negative samples were selected from the non-eye regions inside each image.

3.2.2 Eye Tracking

The results of the eye detector are sometime noisy due to missed detections, false-positives and jitter in the detected eye location. For these reasons we need a way to smooth the estimated eye locations and keep a reasonable estimate of the eye location even if the eye detector doesn't trigger.

Our approach integrates eye detection results with a monocular 3D head pose tracker to achieve smooth and robust eye tracking, that computes the 3D position and orientation of the head at each frame. We initialize our head tracker using the detected face in the first frame. A 3D ellipsoid model is then fit to the face based on the width of the detected face and the camera focal length. The position and orientation of the model are updated at each frame after tracking is performed.

Our approach for head pose tracking is based on the Adaptive view-based appearance model (described in the previous section) and differs from previously published ellipsoid-based head tracking techniques [4] in the fact that we acquire extra key-frames during tracking and adjust the key-frame poses over time. This approach makes it possible to track head pose over a larger range of motion and over a long period of time with bounded drift. The view registration is done using an iterative version of the Normal Flow Constraint [97].

Given the new head pose estimate for the current frame, the region of interest

(ROI) around both eyes is updated so that the center of the ROI reflects the observed head motion. The eye tracker will return two ROIs per eye: one from the eye detector (if the eye was detected) and the other from the updated ROI based on the head velocity.

3.2.3 Gaze Estimation

To estimate the eye gaze given a region of interest inside the image, we created two view-based appearance models [77, 68], one model for each eye. We trained the models using the dataset described in Section 3.2.1, which contains eye images of 16 subjects looking at 35 different orientations, ranging [-30,30] horizontally and [-20,20] vertically.

We define our view-based eigenspaces models $\mathcal{P}_l$ and $\mathcal{P}_r$, for the left and right eye respectively, as:

$$\mathcal{P} = \{\bar{I}_i, \mathcal{V}_i, \varepsilon_i\}$$

where $\bar{I}_i$ is the mean intensity image for view i, ε_i is the eye gaze of that view and $\mathcal{V}_i$ is the eigenspace matrix. The eye gaze is represented as $\varepsilon = \left[\begin{array}{cc} R^x & R^y \end{array}\right]$, a 2-dimensional vector consisting of the horizontal and vertical rotation.

To create the view-based eigenspace models, we first store every segmented eye image in a one-dimensional vector. We can then compute the average vectors $\bar{I}_i = \frac{1}{n}\sum_{j=1}^{n} I_i^j$ and stack all the normalized intensity vectors into a matrix:

$$\mathcal{I}_i = \left[\begin{array}{c} (I_i^1 - \bar{I}_i) \; (I_i^2 - \bar{I}_i) \cdots \end{array}\right]^T$$

To compute the eigenspaces $\mathcal{V}_i$ for each view, we find the SVD decomposition $\mathcal{I}_i = U_i D_i \mathcal{V}_i^T$.

At recognition time, given a seed region of interest (ROI), our algorithm will search around the seed position for the optimal pose with the lowest reconstruction error e_i^*. For each region of interest and each view of the appearance model i, we find the vector $\vec{w}_i$ that minimizes:

$$e_i = |I'_t - \bar{I}_i - \vec{w}_i \cdot \mathcal{V}_{I_i}|^2, \tag{3.6}$$

The lowest reconstruction error e_i^* will be selected and the eye gaze ε_i associated with the optimal view i will be returned. In our implementation, the search region was [+4,-4] pixels along the X axis and [+2,-2] pixels along the Y axis. Also, different scales for the search region were tested during this process, ranging from 0.7 to 1.3 times the original size of the ROI.

Gaze estimation was done independently for each seed ROI returned by the eye tracker described in the previous section. ROIs associated with the left eye are processed using the left view-based appearance model and similarly for the right eye. If more then one seed ROI was used, then the eye gaze with the lowest reconstruction error is selected. The final eye gaze is approximated based on a simple average of the left and right eye gaze estimates.

3.2.4 Experiments

To test the accuracy of our eye gaze estimator we ran a set of experiments using the dataset described earlier in Section 3.2.1. In these experiments, we randomly selected 200 images, then retrained the eye gaze estimator and compared the estimated eye gaze with the ground truth estimate.

We tested two aspects of our estimator: its sensitivity to noise and its performance using different merging techniques. To test our estimator's sensitivity to noise, we added varying amounts of noise to the initial region of interest. Figure 3-9 shows the average error on the eye gaze for varying levels of noise. Our eye gaze estimator is relatively insensitive to noise in the initialized region of interest, maintaining an average error of under 8 degrees for as much as 6 pixel noise.

We also tested two techniques to merge the left and right eye gaze estimates: (1) picking the eye gaze estimate from the eye with the lowest reconstruction error and (2) averaging the eye gaze estimates from both eyes. Figure 3-9 also summarizes the

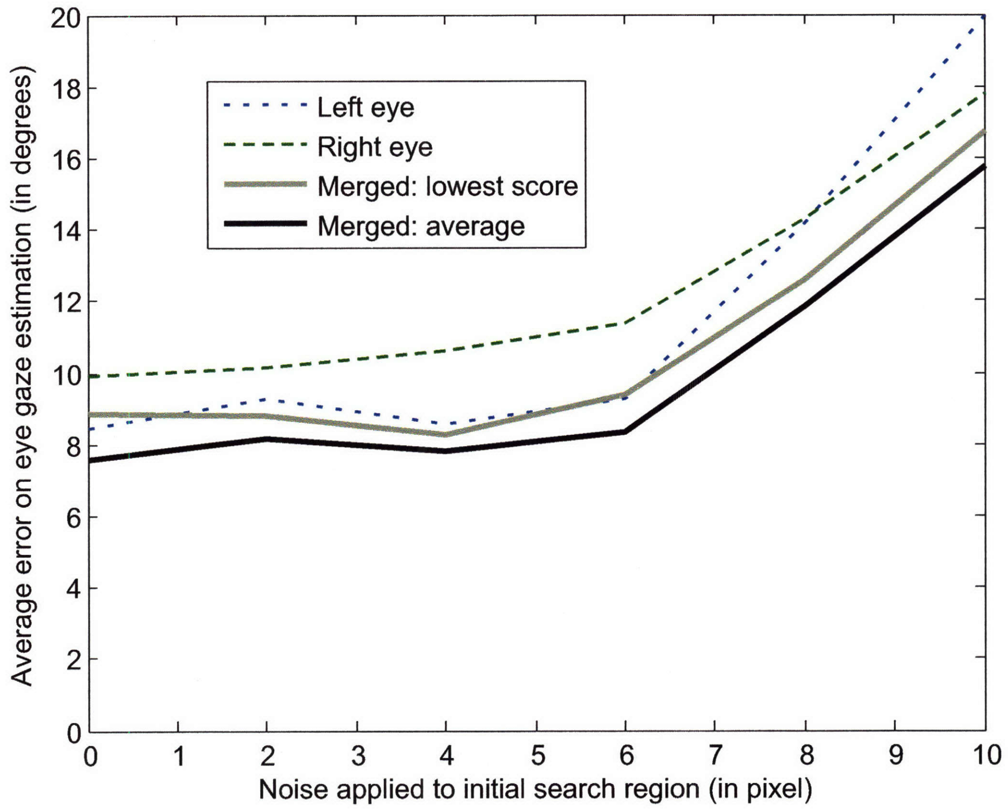


Figure 3-9: Average error of our eye gaze estimator when varying the noise added to the initial region of interest.

result of this experiment. We can see that the averaging technique consistently works better than picking the lowest reconstruction error. This result confirms our choice of using the average to compute eye gaze estimates.

The eye gaze estimator was applied to unsegmented video sequence of all 6 human participants from the user study described in Section 2.2.3. Each video sequence lasted approximately 10-12 minutes, and was recorded at 30 frames/sec, for a total of 105,743 frames. During these interactions, human participants would rotate their head up to +/-70 degrees around the Y axis and +/-20 degrees around the X axis, and would also occasionally translate their head, mostly along the Z axis. The following eye gesture recognition results are on the unsegmented sequences, including extreme head motion.

It is interesting to look at the qualitative accuracy of the eye gaze estimator for a sample sequence of images. Figure 3-10 illustrates a gaze aversion gesture where

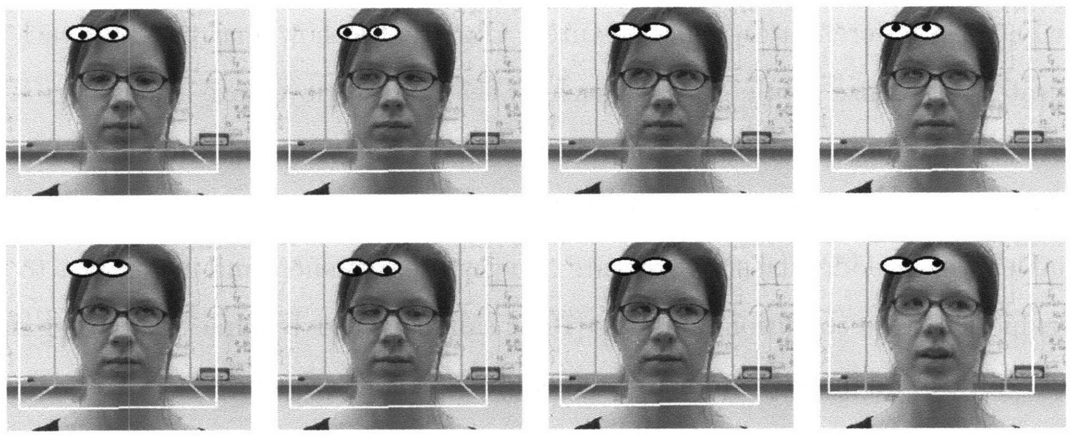


Figure 3-10: Eye gaze estimation for a sample image sequence from our user study. The cartoon eyes depict the estimated eye gaze. The cube represents the head pose computed by the head tracker.

the eye gaze estimates are depicted by cartoon eyes and the head tracking result is indicated by a white cube around the head. Notice that the estimator works quite well even if the participant is wearing eye glasses.

3.3 Visual Gesture Recognition

Visual feedback tends to have a distinct internal sub-structure as well as regular dynamics between individual gestures. A gesture with internal substructure means that it is composed of a set of basis motions combined in an orderly fashion. For example, head-nod gesture has an internal sub-structure that consists of moving the head up, down then back to its starting position. Furthermore the transitions between gestures are not uniformly likely. For example, the head-nod to head-shake transition is usually less likely than a transition between a head-nod and still gesture.

In this section, we introduce a new visual gesture recognition algorithm which can capture both the sub-gesture patterns and the dynamics between gestures. Our Frame-based Hidden Conditional Random Field (FHCRF) model is a discriminative approach for gesture recognition[1]. Instead of generatively modeling each gesture independently as would Hidden Markov Models [111, 33], the FHCRF model focuses

[1]This work was done in collaboration with Ariadna Quattoni.

on what best differentiates visual gestures. Our results show that this approach can accurately recognize subtle gestures such as head nods or eye gaze aversion.

Also, our approach offers several advantages over previous discriminative models. In contrast to Conditional Random Fields (CRFs) [56], it incorporates hidden state variables which model the sub-structure of a gesture sequence, and in contrast to previous hidden-state conditional models [79] it can learn the dynamics between gesture labels and can be directly applied to label unsegmented sequences.

The CRF approach models the transitions between gestures, thus capturing extrinsic dynamics, but it lacks the ability to learn the internal sub-structure. On the other hand, sequence-based HCRF models (as shown in Figure 3-11) can capture intrinsic structure through the use of intermediate hidden states but cannot learn the dynamics between gestures. Sequence-based HCRFs model disjoint gesture segments and were trained and tested with segmented data.

We propose a frame-based HCRF model that combines the strengths of CRFs and sequence-based HCRFs by capturing both extrinsic dynamics and intrinsic structure. It learns the extrinsic dynamics by modeling a continuous stream of labels per frame, and it learns internal sub-structure by utilizing intermediate hidden states. Since frame-based HCRFs include a label variable per observation frame they can be naturally used for recognition on un-segmented sequences; thereby, overcoming one of the main weaknesses of the sequence-based HCRF model.

By assuming a deterministic relationship between class labels and hidden states, our model can be efficiently trained. Our results demonstrate that FHCRF models for context-based recognition outperform models based on Support Vector Machines (SVMs), HMMs, or CRFs.

3.3.1 Related Work

Recognition of head gestures has been demonstrated by tracking eye position over time. Kapoor and Picard presented a technique to recognize head nods and head shakes based on two Hidden Markov Models (HMMs) trained and tested on 2D coordinate results from an eye gaze tracker [49]. Kawato and Ohya developed a technique for head gesture recognition using between eye templates [50]. Compared to eye gaze, head gaze can be estimated more accurately in low resolution images and can be estimated over a larger range [65]. Fugie *et al.* also used HMMs to perform head nod recognition [33]. They combined head gesture detection with prosodic recognition of Japanese spoken utterances to determine strongly positive, weak positive, and negative responses to yes/no type utterances. HMMs [81] and related models have been used to recognize arm gestures [12] and sign languages [1, 96].

Recently many researchers have worked on modeling eye gaze behavior for the purpose of synthesizing a realistic ECA. Colburn *et al.* use hierarchical state machines to model eye gaze patterns in the context of real-time verbal communication [25]. Fukayama *et al.* use a two-state Markov model based on amount of gaze, mean duration of gaze, and gaze points while averted [34]. Lee *et al.* use an eye movement model based on empirical studies of saccade and statistical models of eye-tracking data [57]. Pelachaud and Bilvi proposed a model that embeds information on communicative functions as well as on statistical information of gaze patterns [76].

A significant amount of recent work has shown the power of discriminative models for specific sequence labeling tasks. In the speech and natural language processing community, Conditional Random Field Models (CRFs) [56] have been used for tasks such as word recognition, part-of-speech tagging, text segmentation and information extraction. In the vision community, Sminchisescu *et al.* [94] applied CRFs to classify human motion activities (i.e. walking, jumping, etc) and showed improvements over an HMM approach. Kumar *et al.* [53] used a CRF model for the task of image region labeling. Torralba *et al.* [103] introduced Boosted Random Fields, a model that combines local and global image information for contextual object recognition. An advantage of CRFs are that they can model arbitrary features of observation sequences and can therefore accommodate overlapping features.

When visual phenomena have distinct sub-structure models that exploit hidden state are advantageous. Hidden-state conditional random fields (HCRFs), which can estimate a class given a segmented sequence, have been proposed in both the vision and speech community. In the vision community, HCRFs have been used to model

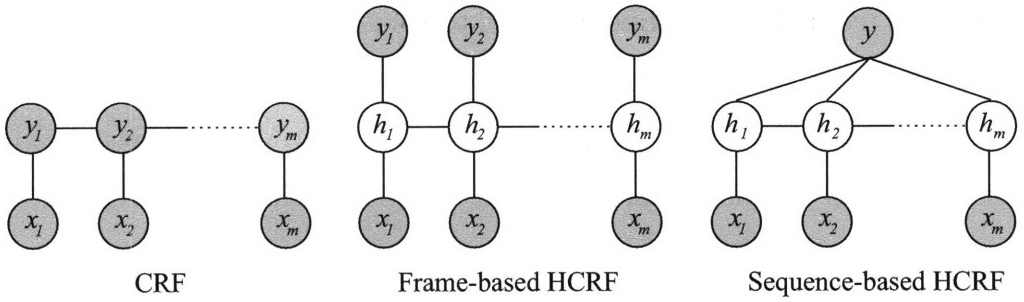


Figure 3-11: Comparison of our model (frame-based HCRF) with two previously published models (CRF [56] and sequence-based HCRF [38, 112]). In these graphical models, x_j represents the j^{th} observation (corresponding to the j^{th} frame of the video sequence), h_j is a hidden state assigned to x_j , and y_j the class label of x_j (i.e. head-nod or other-gesture). Gray circles are observed variables. The frame-based HCRF model combines the strengths of CRFs and sequence-based HCRFs in that it captures both extrinsic dynamics and intrinsic structure and can be naturally applied to predict labels over unsegmented sequences. Note that only the link with the current observation x_j is shown, but for all three models, long range dependencies are possible.

spatial dependencies for object recognition in cluttered images [79] and for arm and head gesture recognition from segmented sequences [112]. In the speech community a similar model was applied to phone classification [38]. Since they are trained on sets of pre-segemented gestures, these *sequence-based HCRF* models do not capture the dynamics between gesture labels, only the internal structure. In both [38, 112], sequence-based HCRFs were applied to segmented sequences, leaving segmentation to a pre-processing step.

Sutton *et al.* [100] presented a dynamic conditional random field (DCRF) model whose structure and parameters are repeated over a sequence. They showed results for sequence segmentation and labeling where the model was trained using loopy belief propagation on a fully-observed training set. As stated by the authors, training a DCRF model with unobserved nodes (hidden variables) makes their approach difficult to optimize.

In this section we develop a discriminative gesture recognition algorithm which incorporates hidden-state variables, can model the dynamics between gestures, can be trained efficiently, and can be directly applied to unsegmented sequences. As depicted in Figure 3-11, our model includes a gesture label variable for each observation in a discriminative field which links hidden state variables; we call the resulting model a Frame-based Hidden-state Conditional Random Field (FHCRF). Our frame-based HCRF model combines the strengths of CRFs and sequence-based HCRFs in that it captures both extrinsic dynamics and intrinsic structure.

We train our model on labeled data, yielding a classifier which can be run directly on unsegmented visual sequences. We have found that restricting our model to have a deterministic relationship between observed gesture frame labels and hidden states significantly simplifies model training, but is still powerful enough to dramatically improve recognition performance over conventional discriminative sequence methods. The proposed algorithm has a similar computational complexity to a fully observable CRF.

3.3.2 Frame-based Hidden-state Conditional Random Fields

Several problems in vision can be thought of as discrete sequence labeling tasks over visual streams. We focus on problems where the goal is to predict a class label at each point in time for a given sequence of observations. In the case of human gesture recognition, a sequence of video frames is given and the goal is to predict a gesture label per frame. We are interested in visual sequences that exhibit both structure within each class label (i.e. intrinsic structure) and distinct transitions between class labels (i.e. extrinsic dynamics).

FHCRF Model

Our task is to learn a mapping between a sequence of observations $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ and a sequence of labels $\mathbf{y} = \{y_1, y_2, ..., y_m\}$.[1] Each y_j is a gesture label for the j^{th} frame of a video sequence and is a member of a set $\mathcal{Y}$ of possible gesture labels, for example, $\mathcal{Y} = \{\texttt{head-nod}, \texttt{other-gesture}\}$. Each frame observation x_j is represented by a feature vector $\phi(x_j) \in \mathbf{R}^d$, for example, the head velocity at each frame.

[1]Note that the sequence length m can vary, and did vary in our experiments. For convenience we use notation where m is fixed. Variable sequence lengths lead to minor changes in the model.

Our training set consists of n labeled sequences $(\mathbf{x_i}, \mathbf{y_i})$ for $i = 1...n$, where each $\mathbf{y_i} = \{y_{i,1}, y_{i,2}, ..., y_{i,m}\}$ and each $\mathbf{x_i} = \{x_{i,1}, x_{i,2}, ..., x_{i,m}\}$. For each sequence, we also assume a vector of "sub-structure" variables $\mathbf{h} = \{h_1, h_2, ..., h_m\}$. These variables are not observed on the training examples and will therefore form a set of hidden variables in the model. Each h_j is a member of a set $\mathcal{H}$ of possible hidden states.

Given the above definitions, we define a latent conditional model:

$$P(\mathbf{y} \mid \mathbf{x}, \theta) = \sum_{\mathbf{h} \in \mathcal{H}^m} P(\mathbf{y}, \mathbf{h} \mid \mathbf{x}, \theta) = \sum_{\mathbf{h} \in \mathcal{H}^m} P(\mathbf{y} \mid \mathbf{h}, \theta) P(\mathbf{h} \mid \mathbf{x}, \theta). \tag{3.7}$$

where θ are the parameters of the model.

Our model assumes that there is a deterministic relationship between frame-based labels y_j and hidden states h_j. Given a frame-based label y_j, the states that a hidden variable h_j can take are constrained to a subset $\mathcal{H}_{y_j}$ of possible hidden states. In other words, the label y_j partitions the space of possible hidden states $\mathcal{H}$. For example, suppose that we have only 3 hidden states per label. Then for a frame with label **head-nod**, the subset of possible hidden states will be $\mathcal{H}_{\texttt{head-nod}} = \{1, 2, 3\}$ while a frame with label `other-gesture` will be constrained to the subset $\mathcal{H}_{\texttt{other-gesture}} = \{4, 5, 6\}$.

Figure 3-12 shows a two-class classification example where internal and external structure of the signal was learned automatically. In this example, hidden states 1,2 and 3 are associated to the `gesture` label while hidden states 4, 5 and 6 are associated to label `non-gesture`. This example is discussed in more detail later in this section.

This deterministic relationship between $\mathbf{y}$ and $\mathbf{h}$ is formally expressed as:

$$P(\mathbf{y} \mid \mathbf{h}, \theta) = \begin{cases} 1 \texttt{ if } \forall h_j \in \mathcal{H}_{y_j} \\ 0 \texttt{ otherwise} \end{cases} \tag{3.8}$$

Using the above relationship we can rewrite Equation 3.7:

$$P(\mathbf{y} \mid \mathbf{x}, \theta) = \sum_{\mathbf{h}: \forall h_j \in \mathcal{H}_{y_j}} P(\mathbf{h} \mid \mathbf{x}, \theta) \tag{3.9}$$

We define $P(\mathbf{h}|\mathbf{x},\theta)$ using the usual conditional random field formulation, that is:

$$P(\mathbf{h}|\ \mathbf{x},\theta) = \frac{1}{\mathcal{Z}(\mathbf{x},\theta)} \exp\left(\sum_k \theta_k \mathbf{F}_k(\mathbf{h},\mathbf{x})\right) \tag{3.10}$$

where the partition function $\mathcal{Z}$ is defined as

$$\mathcal{Z}(\mathbf{x},\theta) = \sum_{\mathbf{h}} \exp\left(\sum_k \theta_k \cdot \mathbf{F}_k(\mathbf{h},\mathbf{x})\right)$$

, F_k is defined as

$$\mathbf{F}_k(\mathbf{h},\mathbf{x}) = \sum_{j=1}^{n} f_k(h_{j-1}, h_j, \mathbf{x}, j),$$

and each feature function $f_k(h_{j-1}, h_j, \mathbf{x}, j)$ is either a state function $s_k(h_j, \mathbf{x}, j)$ or a transition function $t_k(h_{j-1}, h_j, \mathbf{x}, j)$. State functions s_k depend on a single hidden variable in the model while transition functions t_k can depend on pairs of hidden variables.

Learning Model Parameters

Following previous work on CRFs [53, 56], we use the following objective function to learn the parameter θ^*:

$$L(\theta) = \sum_i \log P(\mathbf{y}_i \mid \mathbf{x}_i, \theta) - \frac{1}{2\sigma^2}||\theta||^2 \tag{3.11}$$

The first term in Eq. 3.11 is the log-likelihood of the data. The second term is the log of a Gaussian prior with variance σ^2, i.e., $P(\theta) \sim \exp\left(\frac{1}{2\sigma^2}||\theta||^2\right)$.

We use gradient ascent to search for the optimal parameter values, $\theta^* = \arg\max_\theta L(\theta)$, under this criterion. Given the deterministic relationship between $\mathbf{y}$ and $\mathbf{h}$, the gradient of our objective function $L(\theta)$ with respect to the parameters θ_k associated to a state function s_k can be written as:

$$\frac{\partial L(\theta)}{\partial \theta_k} = \sum_{j,a} P(h_j = a \mid \mathbf{y}, \mathbf{x}, \theta) s_k(j,a,\mathbf{x}) - \sum_{\mathbf{y}',j,a} P(h_j = a, \mathbf{y}' \mid \mathbf{x}, \theta) s_k(j,a,\mathbf{x}) \quad (3.12)$$

where

$$P(h_j = a \mid \mathbf{y}, \mathbf{x}, \theta) = \frac{\sum\limits_{\mathbf{h}:h_j=a \wedge \forall h_j \in \mathcal{H}_{y_j}} P(\mathbf{h} \mid \mathbf{x}, \theta)}{\sum\limits_{\mathbf{h}:\forall h_j \in \mathcal{H}_{y_j}} P(\mathbf{h} \mid \mathbf{x}, \theta)} \quad (3.13)$$

Notice that given our definition of $P(\mathbf{h}|\mathbf{x}, \theta)$ in Equation 3.10, the summations in Equation 3.13 are simply a constrained versions of the partition function $\mathcal{Z}$ over the conditional random field for $\mathbf{h}$. This can be easily shown to be computable in $O(m)$ using belief propagation [75], where m is the length of the sequence.

The gradient of our objective function with respect to the parameters θ_k associated to a transition function t_k can be derived the same way. The marginal probabilities on edges necessary for this gradient, $P(h_j = a, h_k = b \mid \mathbf{y}, \mathbf{x}, \theta)$, can also be computed efficiently using belief propagation. Thus both inference and training are efficient operations using our model. In our experiments, we performed gradient ascent with the BFGS optimization technique. All the models required less than 300 iterations to converge.

Inference

For testing, given a new test sequence $\mathbf{x}$, we want to estimate the most probable sequence labels $\mathbf{y}^*$ that maximizes our conditional model:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}, \theta^*) \quad (3.14)$$

where the parameter values θ^* were learned from training examples. Using Equation 3.9, the previous equation can be rewritten as:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{\mathbf{h}:\forall h_i \in \mathcal{H}_{y_i}} P(\mathbf{h} \mid \mathbf{x}, \theta) \tag{3.15}$$

To estimate the label y_j^* of frame j, the marginal probabilities $P(h_j = a \mid \mathbf{x}, \theta)$ are computed for all possible hidden states $a \in \mathcal{H}$. Then the marginal probabilities are summed according to the grouping of the hidden states subsets $\mathcal{H}_{y_j}$ and the label associated with the optimal subset is chosen. As discussed in the previous subsection, the marginal probabilities can efficiently be computed using belief propagation. Note that the sum of the marginal probabilities is an approximation to Equation 3.15, but using this approach instead of computing the Viterbi path has the advantage that we can vary the decision threshold and plot ROC curves for each model.

Feature Functions

In our model, $|\mathcal{H}| \times |\mathcal{H}|$ transitions functions t_k are defined one for each hidden state pair (h', h''). Each transition function is expressed as,

$$t_k(h_{j_1}, h_j, \mathbf{x}, j) = \begin{cases} \texttt{1 if } h_{j-1} = h' \texttt{ and } h_j = h'' \\ \texttt{0 otherwise} \end{cases}$$

It is worth noticing that the weights θ_k associated with the transition functions model both the intrinsic and extrinsic dynamics. Weights associated with a transition function for hidden states that are in the same subset $\mathcal{H}_{y_i}$ will model the substructure patterns while weights associated with the transition functions for hidden states from different subsets will model the external dynamic between gestures.

The number of state functions, s_k, will be equal to the length of the feature vector, ϕ, times the number of possible hidden states, $|\mathcal{H}|$. In the case of head gesture recognition where the rotational velocity (yaw, roll and pitch) is used as input, the length of our feature vector, ϕ, will be 3 dimensions per frame. If our model has 6 hidden states (3 per label) then the total number of state functions, s_k, (and total number of associated weights, θ_k) will be $3 \times 6 = 18$.

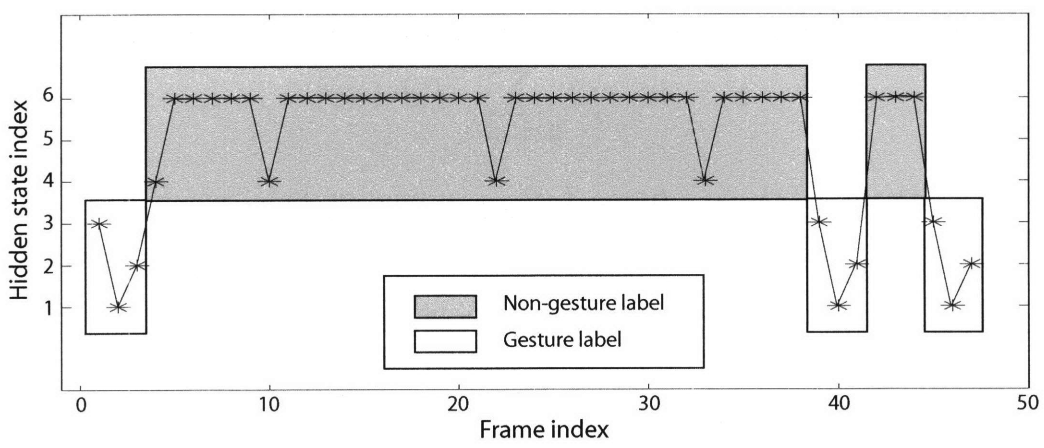


Figure 3-12: This figure shows the MAP assignment of hidden states given by the frame-based HCRF model for a sample sequence from the synthetic data set. As we can see the model has used the hidden states to learn the internal sub-structure of the gesture class.

Synthetic Example

We illustrate how our frame-based HCRF model can capture both extrinsic dynamics and intrinsic structure using a simple example. A synthetic data set was constructed containing sequences from a `gesture` and `non-gesture` class. Subsequences belonging to the gesture class consist of three sub-gesture samples simulating the beginning, middle and end of the gesture. These samples are created by sampling from three Gaussian distributions in a deterministic order. The non-gesture subsequences are generated by picking k samples from a mixture of seven Gaussians, where k is the length of the subsequence, picked at random between 1 and 10.

Both the training and testing data sets consisted of 200 1-dimensional sequences of variable length (30-50 samples per sequence). Synthetic sequences were constructed by concatenating subsequences where the class of each of the subsequences was randomly picked between the gesture and non-gesture classes.

Given this data set we trained both a frame-based HCRF model with three hidden states and a CRF. The CRF model was only able to recognize 72% (equal error rate) of the test examples with this simple synthetic data set, while our frame-based HCRF model has perfect performance. Figure 3-12 shows the sequence of hidden labels assigned by our frame-based HCRF model for a sequence in the testing set. As this figure suggests the model has learned the intrinsic structure of the class using one of its hidden states to recognize each of the sub-structures. A model that can learn internal sub-structure and dynamics has the potential to outperform a model that cannot. In the following section we show an experiment on a non-synthetic gesture recognition data set.

3.3.3 Experiments

We want to analyze the performance of our FHCRF model for visual feedback recognition with different types of interactive systems. In our experiments we focus on head and eye gestures. Our datasets came from three user studies described in Chapter 2: (1) head gestures when interacting with a physical avatar (Section 2.3.1), (2) head gesture-based widgets for a non-embodied interface (Section 2.4.1), and (3) eye gestures for interaction with a virtual embodied agent (Section 2.2.3).

In this section, we first describe all three datasets used in our experiments, then present the models used to compare the performance of our FHCRF model, and finally we give more explanation on the methodology used during our experiments.

Datasets

MelHead This dataset consisted of head velocities from 16 human participants interacting with Mel, the interactive robot described in Section 2.3.1. These participants were part of the `MelNodsBack` group. Each interaction lasted between 2 to 5 minutes. Head pose tracking was performed online using our adaptive view-based appearance model described in Section 3.1. At each frame (~25Hz), the tracker logged with timestamps the 3D position and orientation of the head.

Human participants were video recorded while interacting with the robot to obtain ground truth labels including the start and end points of each head nod. From these ground truth labels, each frame was assigned to one of the two labels: head-nod or other-gesture. A total of 274 head nods were naturally performed by the participants while interacting with the robot. All other types of head gestures (i.e. head shakes, look away, no motion,...) were labeled as other-gestures.

WidgetsHead This dataset consisted of head velocities from 12 participants who interacted with the gesture-based widgets described in Section 2.4.1. As for the first dataset, the head pose was also estimated using the adaptive view-based appearance model described in Section 3.1. The video sequences were manually annotated for ground truth. Each frame was labeled as either head-nod or other-gesture. From this dataset of 79 minutes of interaction, 269 head nods were labeled. All other types of head gestures (e.g. head shakes, look away, and no motion) were labeled as other-gestures.

AvatarEye This dataset consisted of eye gaze estimates from 6 human participants interacting with a virtual embodied agent as described in Section 2.2.3. Our goal is to recognize gaze aversion gestures from video sequences. Each video sequence lasted approximately 10-12 minutes, and was recorded at 30 frames/sec, for a total of 105,743 frames. During these interactions, human participants would rotate their head up to +/-70 degrees around the Y axis and +/-20 degrees around the X axis, and would also occasionally translate their head, mostly along the Z axis. The following eye gesture recognition results are on the unsegmented sequences, including extreme head motion.

For each video sequence, eye gaze was estimated using the view-based appearance model described in Section 3.2.3 and for each frame a 2-dimensional eye gaze estimate was obtained. The dataset was labeled with the start and end points of each gaze aversion gestures. Each frame was labeled as either gaze-aversion or as other-gesture which included sections of video where people were looking at the avatar or performing deictic gestures.

Models

In our experiments, our FHCRF model is compared with three other models: Conditional Random Field (CRF), Hidden Markov Model (HMM) and Support Vector Machine (SVM). Note that for HMM we tested two configurations: HMM with a sliding window (referred to as HMM in the results section) and concatenated HMM (referred to as HMM-C).

Conditional Random Field As a first baseline, we trained a single CRF chain model where every gesture class has a corresponding state label. During evaluation, marginal probabilities were computed for each state label and each frame of the sequence using belief propagation. Note that we used marginal probabilities instead of computing the Viterbi path to be consistent with the FHCRF output. Also, the marginal probabilities are used to create ROC curves where we vary the decision threshold. The optimal label for a specific frame is typically selected as the label with the highest marginal probability. In our case, to be able to plot ROC curves of our results, the marginal probability of the primary label (i.e. head-nod or gaze-aversion) was thresholded at each frame, and the frame was given a positive label if the marginal probability was larger than the threshold.

During training and validation, the objective function of the CRF model contains a regularization term similar to the regularization term shown in Equation 3.11 for the FHCRF model. This regularization term was validated with values ranging from 0.001 to 1000 on the logarithmic scale.

Support Vector Machine As a second baseline, a multi-class SVM was trained with one label per gesture using a Radial Basis Function (RBF) kernel. Since the SVM does not encode the dynamics between frames, the training set was decomposed into frame-based samples, where the input to the SVM is the head velocity or eye gaze at a specific frame. The output of the SVM is a margin for each class. This SVM margin measures how close a sample is to the SVM decision boundary. The margin was used for plotting ROC curves.

During training and validation, two parameters were validated: C, the penalty parameter of the error term in the SVM objective function, and γ, the RBF kernel parameter. Both parameters were validated with values ranging from 0.001 to 1000 on the logarithmic scale.

Hidden Markov Model As a third baseline, an HMM was trained for each gesture class. Since HHMs are designed for segmented data, we trained each HMM with segmented subsequences where the frames of each subsequence all belong to the same gesture class. This training set contained the same number of frames as the one used for training the 3 other models except frames were grouped into subsequences according to their label. As we stated earlier, we tested two configurations of Hidden Markov Models: an HMM evaluated over a sliding window (referred to as HMM in our experiments) and concatenated HMMs (refered to as HMM-C). For the first configuration, each trained HMM is tested separately on the new sequence using a sliding window of fixed size (32 frames). The class label associated with the HMM with the highest likelihood is selected for the frame at the center of the sliding window.

For the second configuration, the HMMs trained on subsequences are concatenated into a single HMM with the number of hidden states equal to the sum of hidden states from each individual HMM. For example, if the recognition problem has two labels (e.g., `gesture` and `other-gesture`) and each individual HMM is trained using 3 hidden states then the concatenated HMM will have 6 hidden states. To estimate the transition matrix of the concatenated HMM, we compute the Viterbi path of each training subsequence, concatenate the subsequences into their original order, and then count the number of transitions between hidden states. The resulting transition matrix is then normalized so that its rows sum to one. At testing, we apply the forward-backward algorithm on the new sequence, and then sum at each frame the hidden states associated with each class label. The resulting HMM-C can seen as a generative version of our FHCRF model.

During training and validation, we varied the number of states from 1 to 6 and the number of Gaussians per mixture from 1 to 3.

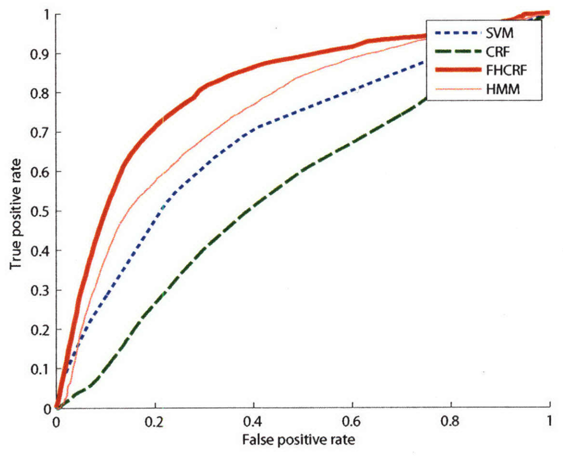
Frame-based Hidden-state Conditional Random Fields Our FHCRF model was trained using the objective function described in the previous section. During testing, to be able to plot ROC curves, we computed the marginal probabilities for each label as shown in Equation 3.15.

During training and validation we validated two parameters: the number of hidden states per label and the regularization term. We varied the number of hidden states from 2 to 6 states per label. The regularization term was validated with values ranging from 0.001 to 1000 on the logarithmic scale.

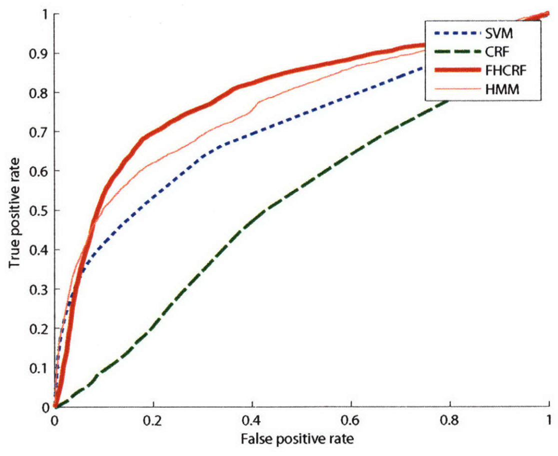
Methodology

For all three datasets, the experiments were performed using a K-fold testing approach where K sequences were held out for testing while all other sequences were used for training and validation. This process is repeated N/K times, where N is the total number of sequences. For the `MelHead`, `WidgetsHead` and `AvatarEye` datasets, K was 4, 3 and 1 respectively. For validation, we did a holdout cross-validation where a subset of the training set is kept for validation. The size of this validation set was equal to 4, 3 and 1 for the `MelHead`, `WidgetsHead` and `AvatarEye` datasets respectively. The optimal validation parameters were picked based on the equal error rate for the validation set.

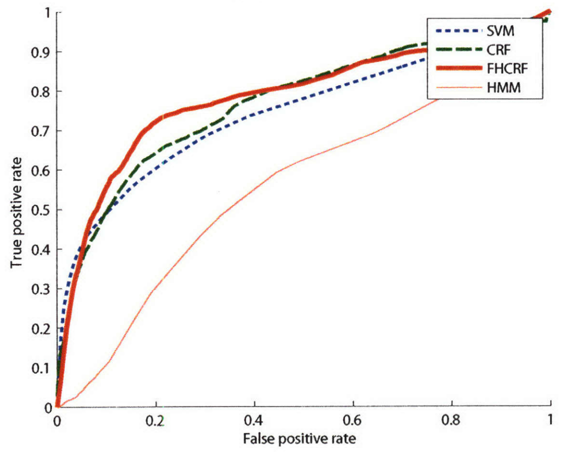
To reduce the training time and have a balanced training set, the training dataset was preprocessed to create a smaller training dataset with equal number of `gesture` and `non-gesture` subsequences. Each `gesture` subsequence included one ground truth gesture from the training dataset. Since the goal was to learn the transition between gestures, the `gesture` subsequences also included non-gesture frames before and after the ground truth gesture. The size of the gap before and after the gesture was randomly picked between 2 and 50 frames. `Non-gesture` subsequences were randomly extracted from the original sequences of the training set. Every frame in the `non-gesture` subsequences had the same `non-gesture` label. The length of these



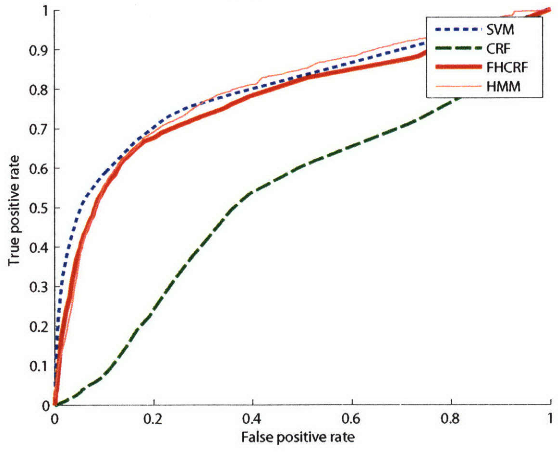
(a) W=0



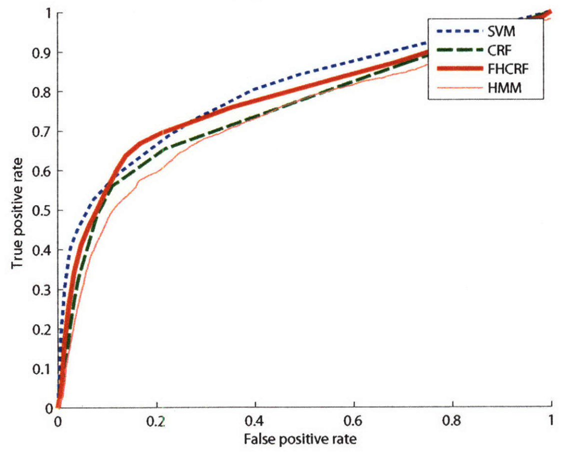
(b) W=2



(c) W=5



(d) W=10



(e) W=16 (FFT)

Figure 3-13: ROC curves for the`MelHead` dataset. Each graph represents a different window size: (a) W=0, (b) W=2, (c) W=5 and (d) W=10. The last graph (e) shows the ROC curves when FFT is applied on a window size of 32(W=16). In bold is the largest EER value.

Model	W=0	W=2	W=5
SVM	66.0% (***p=0.0014***)	66.3% (p=0.1271)	69.3% (p=0.1665)
CRF	55.3% (***p=0.0007***)	53.2% (***p=0.013***)	57.4% (***p=0.0222***)
HMM	69.6% (p=0.1704)	69.5% (p=0.0876)	70.4% (p=0.3361)
HMM-C	69.8% (p=0.0577)	66.9% (***p=0.0491***)	66.1% (***p=0.0194***)
FHCRF	**75.9%**	74.0%	75.1%

Model	W=10	FFT (W=16)
SVM	74.8% (p=0.4369)	73.6% (p=0.5819)
CRF	56.3% (p=0.1156)	69.8% (***p=0.0494***)
HMM	73.7% (p=0.7358)	68.4% (p=0.4378)
HMM-C	70.8% (p=0.6518)	70.4% (p=0.4429)
FHCRF	72.4%	73.4%

Table 3.2: Accuracy at equal error rate for different window sizes on the `MelHead` dataset. The best model is FHCRF with window size of 0. Based on paired t-tests between FHCRF and each other model, p-values that are statistically significant are printed in italic.

subsequences varied between 30-60 frames. The resulting training set had the same number of `gesture` and `non-gesture` subsequences which is equal to the number of ground truth gestures in the original training set.

Each experiment was also repeated with different input feature window sizes. A window size equal to zero (W=0) means that only the velocity at the current frame was used to create the input feature. A window size of two (W=2) means that the input feature vector at each frame is a concatenation of the velocities from five frames: the current frame, the two preceding frames, and the two future frames.

3.3.4 Results and Discussion

In this section, the results of our experiments for head gesture and eye gesture recognition are presented. We compared all four models (SVM, CRF, HMM and FHCRF) on three datasets. For the ROC curves shown in this section, the true positive rates were computed by dividing the number of recognized frames by the total number of ground truth frames. Similarly, the false positive rates were computed by dividing the number of falsely recognized frames by the total number of non-gesture frames.

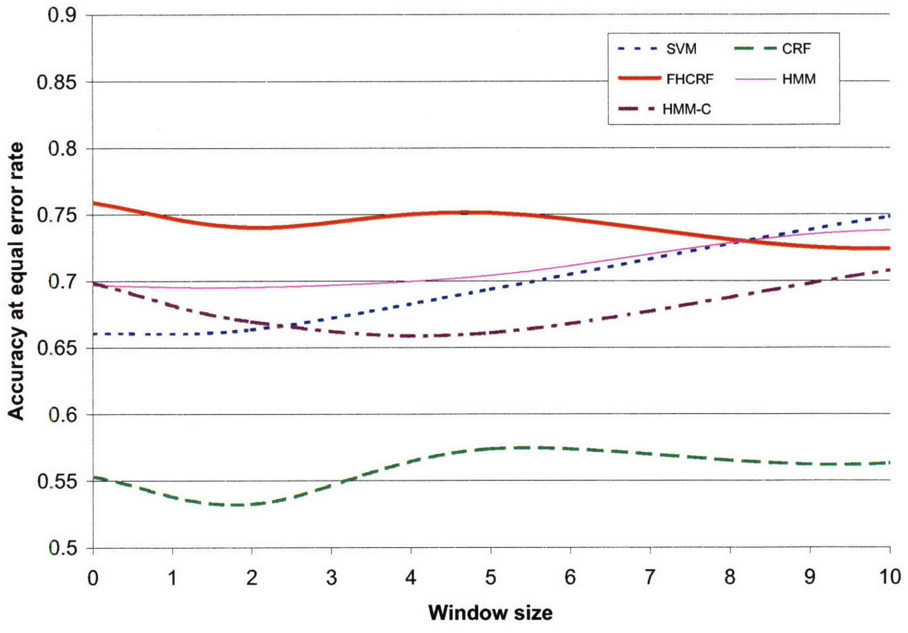
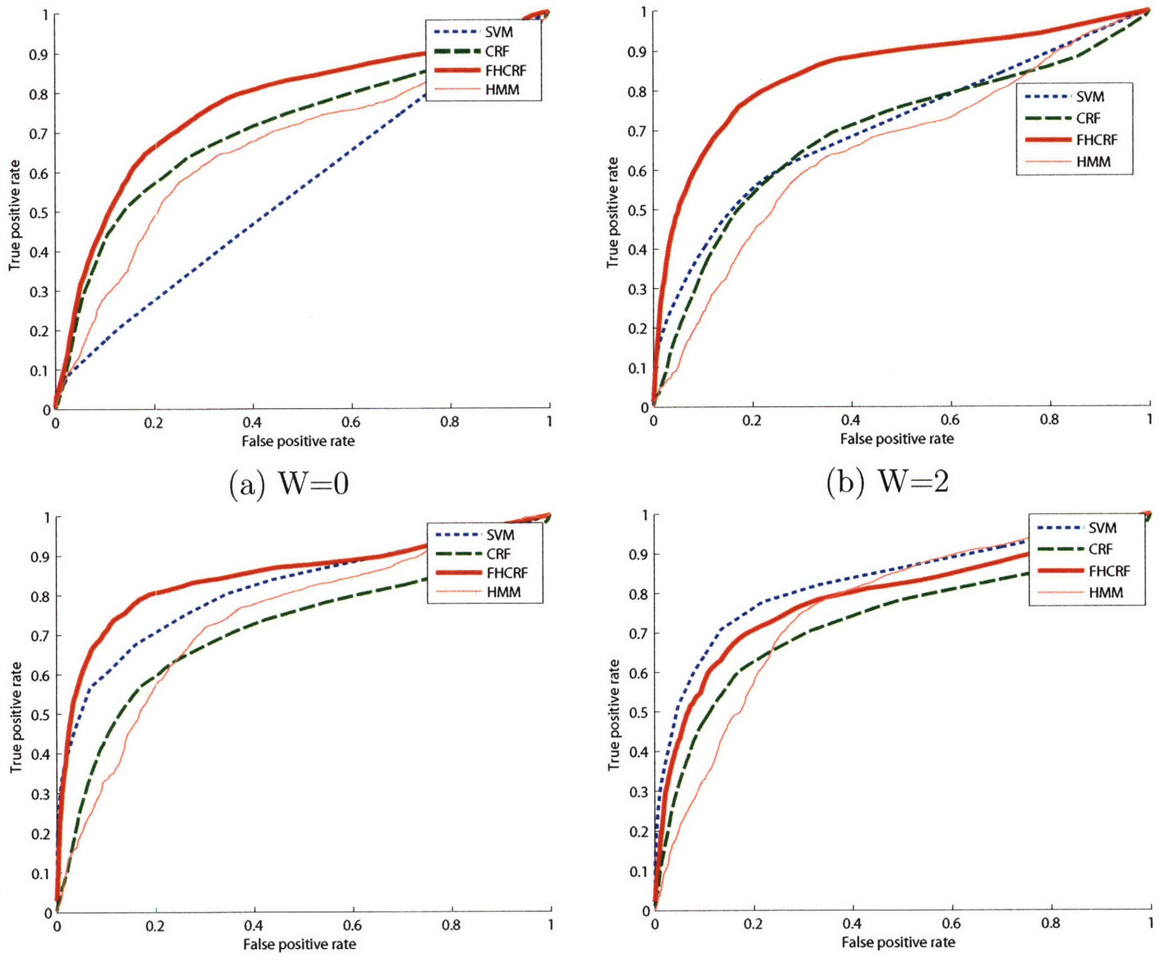


Figure 3-14: Accuracy at equal error rates as a function of the window size, for dataset `MelHead`.

MelHead

In the `MelHead` dataset, the human participants were standing in front of the robot, and were able to move freely about their environment, making this dataset particularly challenging.

Figure 3-13 displays the ROC curves of the four models after training with different window sizes. The last graph of Figure 3-13 shows the performance of all four models when trained on input vectors created by applying an FFT filter over a window of size 32 (W=16). Table 3.2 provides a summary of Figure 3-13 and displays the recognition rate at which both the true positive rate and the true negative rate are equal. This recognition rate (true positive rate) is known as equal error rate (EER). A paired t-test was performed between the FHCRF model and all three other models. The p-values are shown in Table 3.2 next to the equal error rates for the SVM, CRF and HMM models. The difference between FHCRF and SVM is statistically significant at W=0 but is not statistically significant for W=10 which is when SVM accuracy is slightly higher then FHCRF. Figure 3-14 shows a plot of the EER values displayed



(a) W=0 (b) W=2

(c) W=5 (d) W=10

Figure 3-15: ROC curves for the `WidgetsHead` dataset. Each graph represents a different window size: (a) W=0, (b) W=2, (c) W=5 and (d) W=10.

in Table 3.2 as a function of the window size for each model.

In Table 3.2, the largest EER for all the models and all window sizes is the FHCRF model with a window size of 0. As expected the FHCRF model performs quite well even when given a small window size. This is a direct result of the FHCRF model's capability to simultaneously model the intrinsic and extrinsic dynamics of the input signal. With only the velocity of the current frame as input, the FHCRF is able to outperform other models with input vectors that are 10 to 20 times larger. This makes FHCRF more suitable for online applications where knowledge for future frames is not available.

Model	W=0	W=2	W=5
SVM	53.1% (***p=0.0109***)	65.5% (***p=0.0107***)	74.7% (p=0.129)
CRF	67.0% (p=0.1161)	66.8% (***p=0.0076***)	68.1% (p=0.083)
HMM	64.5% (***p=0.0546***)	63.2% (***p=0.0071***)	69.03% (***p=0.0016***)
HMM-C	68.9% (***p=0.0132***)	72.1% (p=0.0716)	76.9% (p=0.4142)
FHCRF	73.7%	**80.1%**	79.6%

Model	W=10	W=15
SVM	77.5% (p=0.4953)	77.8% (***p=0.0484***)
CRF	70.07% (p=0.1338)	71.7% (p=0.6592)
HMM	71.3% (p=0.3359)	69.6% (p=0.6616)
HMM-C	79.2% (p=0.2751)	73.8% (p=0.9377)
FHCRF	75.2%	69.3%

Table 3.3: Accuracy at equal error rate for different window sizes on the `WidhetsHead` dataset. The best model is FHCRF with window size of 2. Based on paired t-tests between FHCRF and each other model, p-values that are statistically significant are printed in italic.

WidgetsHead

Figure 3-15 shows the ROC curves of the four models after training with different window sizes. Table 3.3 displays the equal error rates (EER) for each ROC curve in Figure 3-15. A paired t-test was performed between the FHCRF model and all 3 other models. The p-values are shown in Table 3.3 next to the equal error rates for the SVM, CRF and HMM models. For a window size of 2, the difference between the accuracy of FHCRF and every other model is statistically significant. Figure 3-16 plots the EER values of Table 3.3 as a function of the window size for each model.

By looking at Figure 3-16, we can see that for this dataset, the FHCRF works best with a small window size between 2 and 5. Again the FHCRF outperforms the other models, even when they are using a larger window size.

AvatarEye

Figure 3-17 shows the ROC curves of the four models after training with different window sizes. Table 3.4 displays the equal error rates (EER) for each ROC curve in Figure 3-17. A paired t-test was performed between the FHCRF model and all 3 other models. The p-values are shown in Table 3.4 next to the equal error rates for

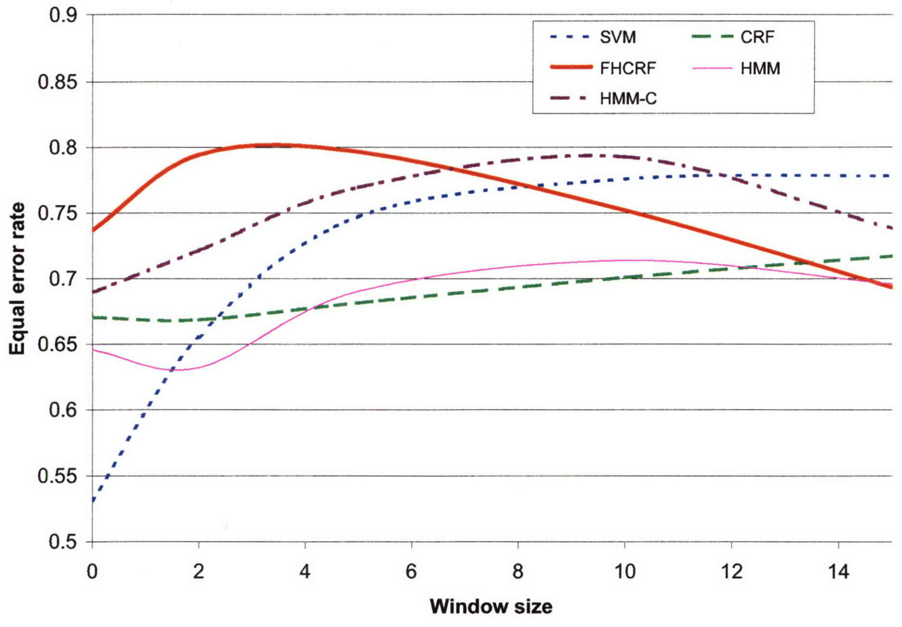


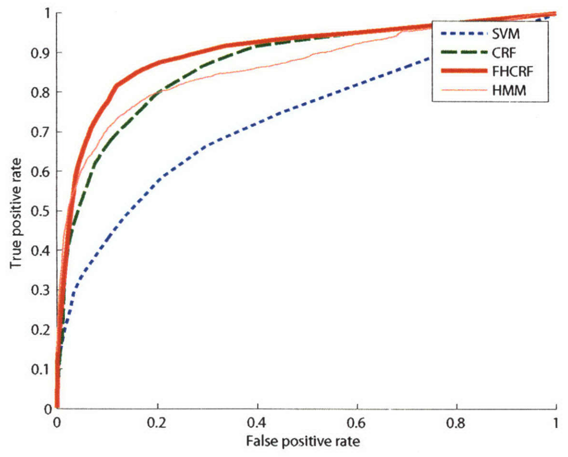
Figure 3-16: Accuracy at equal error rates as a function of the window size, for dataset `WidgetsHead`.

the SVM, CRF and HMM models. We can see in Table 3.4 the FHCRF outperforms all three other models for both window sizes. Figure 3-18 shows a plot of the EER values displayed in Table 3.4 as a function of the window size for each model. Note that a problem with memory allocation made it impossible to run the experiment with W=10.

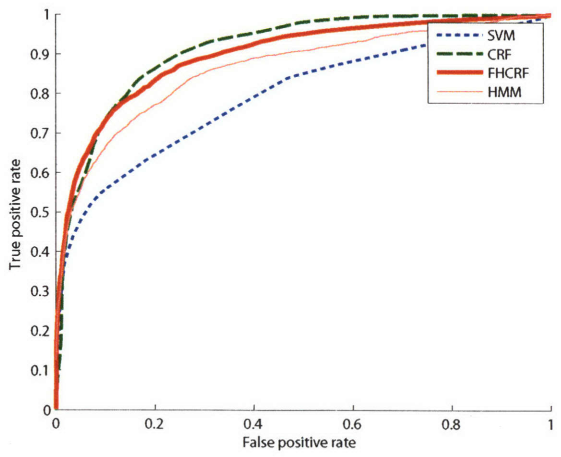
The `AvatarEye` dataset had only 6 participants and 77 eye gestures. We can see in Figure 3-18 how this small dataset affects the FHCRF model when the window size increases. This effect was not as prominent for larger datasets, as observed in Figures 3-16 and 3-14. Even with this small dataset, FHCRF outperforms the three other models with a maximum accuracy of 85.1% for a window size of 0.

3.4 Summary

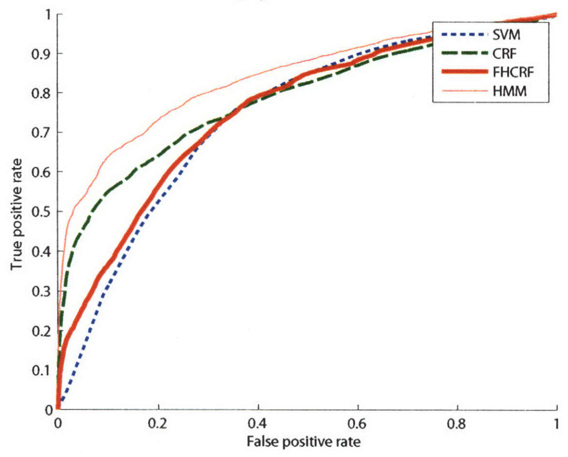
In this chapter we presented and evaluated new algorithms for recognizing head gaze, eye gaze, head gestures and eye gestures. For head tracking, we introduced



(a) W=0



(b) W=2



(c) W=5

Figure 3-17: ROC curves for the`AvatarEye` dataset. The ROC curves are shown for a window size of 0 (a), 2 (b) and 5 (c).

Model	W=0	W =2	W=5
SVM	68.4% (***p=0.0055***)	71.4% (***p=0.0305***)	70.7% (p=0.7954)
CRF	79.9% (p=0.1548)	83.6% (p=0.8234)	72.8% (p=0.6756)
HMM	80.0% (p=0.1086)	79.4% (p=0.308)	78.0% (***p=0.0273***)
HMM-C	72.0% (***p=0.021***)	68.3% (***p=0.0301***)	69.0% (p=0.7916)
FHCRF	**85.1%**	82.9%	70.0%

Table 3.4: Accuracy at equal error rates for different window sizes on the AvatarEye dataset. The best model is FHCRF with window size of 0. Based on paired t-tests between FHCRF and each other model, p-values that are statistically significant are printed in italic.

a new tracking approach that utilizes an adaptive view-based appearance model. This tracker registers each incoming frame against the key-frames of the view-based model using a two-frame 3D registration algorithm. Pose-changes recovered from registration are used to simultaneously update the model and track the subject. The approach was successfully tested on a real-time 6-DOF head tracking task using stereo cameras and observed an RMS error within the accuracy limit of an attached inertial sensor. During all our experiments, the tracker had bounded drift, could model non-Lambertian reflectance and could be used to track objects undergoing large motion for a long time.

Our approach for eye gaze estimation is based on a pre-acquired view-based appearance model. The eye appearance model was built using eye images from 16 subjects looking at 35 different targets under different lighting conditions. The main advantages of this technique are that it can estimate eye gaze from low resolution images, our approach is user independent, and handle glasses and changes in lighting condition.

For both head and eye gesture recognition, we presented a new model for simultaneous segmentation and labeling of gesture sequences. Our frame-based HCRF is a latent state discriminative model that is designed to capture both intrinsic and extrinsic sequence dynamics. Our proposed approach incorporates hidden state variables which model the sub-structure of a gesture sequence. In contrast to previous hidden-state conditional models it can also learn the dynamics between gesture labels and be directly applied to label unsegmented sequences. One advantage of our model

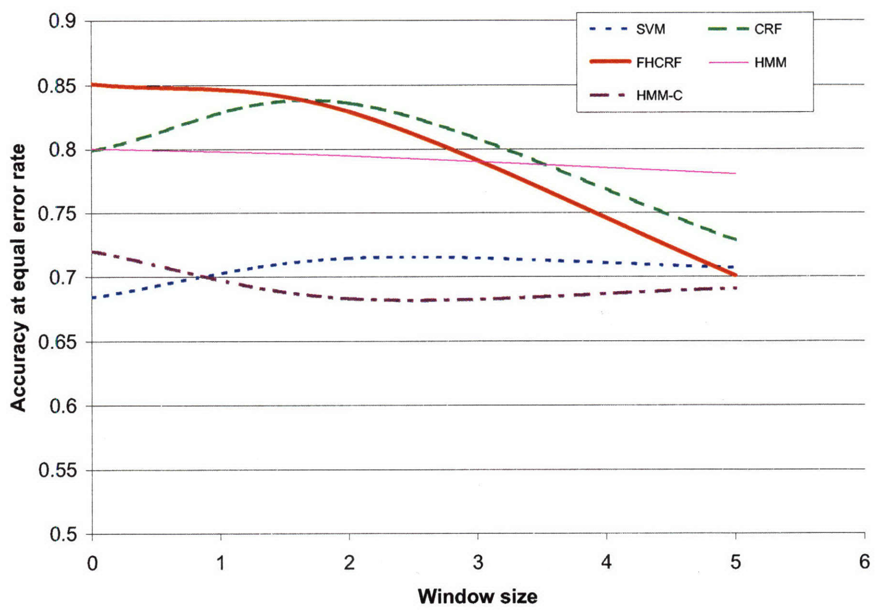


Figure 3-18: Accuracy at equal error rates as a function of the window size, for dataset `AvatarEye`.

is that it can be efficiently trained by using standard belief propagation techniques. We evaluated our FHCRF model on the tasks of head and eye gesture recognition from unsegmented video streams. By modeling latent sub-structure the FHCRF can learn to label unsegmented sequences with high accuracy even when using simple velocity features.

Chapter 4

Visual Feedback Anticipation: The Role of Context

During face-to-face conversation, people use visual feedback to communicate relevant information and to synchronize rhythm between participants. When people interact naturally with each other, it is common to see indications of acknowledgment, agreement, or disinterest given with a simple head gesture. When recognizing such visual feedback, people use more than their visual perception. For instance, knowledge about the current topic and expectations from previous utterances help guide recognition of nonverbal cues. Our goal is to enable computer interfaces with the ability to perceive visual feedback gestures, and to exploit contextual information from the current interaction state when performing visual feedback recognition.

In Chapter 3 we presented a head pose tracking system which can return the position and orientation of a user's head through automatic passive observation. We showed how we can use a view-based appearance model to estimate the eye gaze of a user, and we described a new method for recognition of visual gestures using discriminatively trained statistical classifiers. In this section we show how the use of contextual information proves critical when incorporating visual feedback recognition in conversational dialogue interfaces and in traditional window-based graphical user interfaces.

When interacting with a computer in a conversational setting, dialogue state can provide a useful context for recognition. In the last decade, many embodied conversational agents (ECAs) have been developed for face-to-face interaction, using both physical robots and virtual avatars. A key component of these systems is the dialogue manager, usually consisting of a history of the past events, a list of current discourse moves, and an agenda of future actions. The dialogue manager uses contextual information to decide which verbal or nonverbal action the agent should perform next (i.e., context-based synthesis). Contextual information has proven useful for aiding speech recognition: in [58], a speech recognizer's grammar changes dynamically depending on the agent's previous action or utterance. In a similar fashion, we have developed a context-based visual recognition module that builds upon the contextual information available in the dialogue manager to improve performance of visual feedback recognition (Figure 4-1).

As shown in Section 2.4.1, visual gesture recognition can also be useful for non-conversational interfaces. A perceptive interface sensitive to head gesture can lead to more natural notification and navigation interactions. Computer interfaces often interrupt a user's primary activity with a notification about an event or condition, which may or may not be relevant to the main activity. Currently, a user must shift keyboard or mouse focus to attend to the notification, page through the displayed information and dismiss the notification before returning to the main activity. Requiring keyboard or mouse events to respond to notifications can clearly cause disruption to users during certain activities or tasks. In Section 2.4.1, we explored two types of head gesture-based window controls: dialog box acknowledgment/agreement, and document browsing. The first allowed a user to effectively accept or reject a dialog box or other notification window by nodding or shaking their head. The second component allowed a user to page through a document using head nods. For both types of gesture-based window control, contextual information from the system events (i.e. keystrokes or mouse events) can improve recognition of visual gestures.

In this chapter, we present a prediction framework for incorporating context with vision-based head gesture recognition. Contextual features are derived from the utterances of an ECA or the event state of a traditional user interface. Figure 4-1 shows

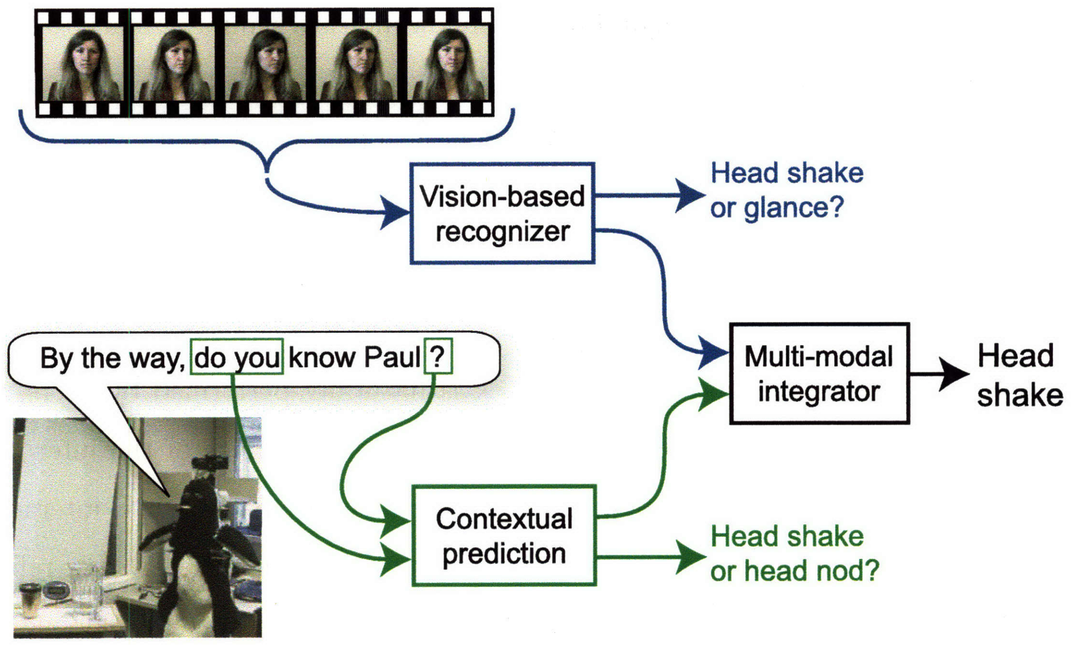


Figure 4-1: Contextual recognition of head gestures during face-to-face interaction with a conversational robot. In this scenario, contextual information from the robot's spoken utterance helps disambiguating the listener's visual gesture.

an example where our framework allows us to correctly predict that a glance is not as likely to occur as a head shake or nod. Based on the fact that robot's spoken utterance contains the words "do you" and ends with a question mark, our *contextual predictor* anticipates a head nod or a head shake. If the *vision-based recognizer* saw a gesture that looks like a head shake or a head glance, then the *multi-modal integrator* will correctly recognize the gesture as a head shake. Figure 4-3 presents our framework for context-based gesture recognition.

Context has been previously used in computer vision to disambiguate recognition of individual objects given the current overall scene category [104]. Fugie *et al.* combined head gesture detection with prosodic recognition of Japanese spoken utterances to determine strongly positive, weak positive and negative responses to yes/no type utterances [33]. To our knowledge, the use of dialogue or interface context for visual gesture recognition has not been explored before for conversational or window-based interaction.

In the following sections we describe the contextual information available in conversational dialogue systems and traditional window interfaces, our approach to context-based gesture recognition using a discriminative classifier cascade, and our experiments in recognizing head and eye gestures using contextual information.

4.1 Context in Conversational Interaction

As shown in Chapter 2, some visual feedback is naturally performed by users, and recognizing these types of feedback can improve the performance of the interactive interface. We discussed in Section 2.1.2 how human participants when interacting with a robot or an ECA often nod without speaking any phrases [7, 92]. For reliable recognition of visual feedback, people use knowledge about the current dialogue during face-to-face conversational interactions to anticipate visual feedback from their interlocutors. Our goal is to equip computer interfaces with the ability to similarly perceive visual feedback gestures. As depicted in Figure 4-1, knowledge of an ECA's spoken utterance can help predict which visual feedback is most likely.

We can use a conversational agent's knowledge about the current dialogue to improve recognition of visual feedback (i.e., head gestures, eye gestures). Figure 4-2 shows a simplified architecture which captures modules common to several different systems [69, 84]. The dialogue manager merges information from the input devices with the history and the discourse model. The dialogue manager contains two main sub-components, an agenda and a history: the agenda keeps a list of all the possible actions the agent and the user (i.e., human participant) can do next. This list is updated by the dialogue manager based on its discourse model (prior knowledge) and on the history. Dialogue managers generally exploit contextual information to produce output for the speech and gesture synthesizer, and we can use similar cues to predict when user visual feedback is most likely.

We extract information from the dialogue manager rather than directly accessing internal ECA states. Our proposed method extracts contextual features from the messages sent to the audio and gesture synthesizers. This strategy allows us to extract dialogue context without any knowledge of the internal representation. Therefore, our

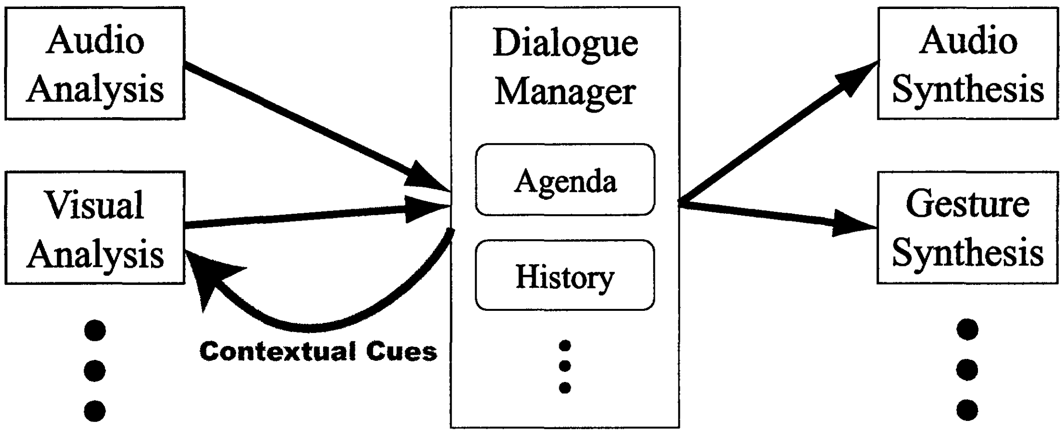


Figure 4-2: Simplified architecture for embodied conversational agent. Our method integrates contextual information from the dialogue manager inside the visual analysis module.

method can be applied to most ECA architectures.

We highlight four types of contextual features easily available from the dialogue manager: lexical features, prosody and punctuation, timing, and gesture displays.

Lexical features Lexical features are computed from the words said by the embodied agent. By analyzing the word content of the current or next utterance, one should be able to anticipate and distinguish certain visual feedback gestures. For example, if the current spoken utterance starts with "Do you", the listener will most likely answer using affirmation or negation. In this case, visual feedback in the form of a head nod or a head shake is likely to occur (as shown in Figure 4-1). On the other hand, if the current spoken utterance starts with "What", then it is less likely to see the listener head shake or head nod; other visual feedback gestures (e.g., pointing) are more likely.

Prosody and punctuation Prosody can also be an important cue to predict gesture displays. We use punctuation features output by the dialogue system as a proxy for prosody cues. Punctuation features modify how the text-to-speech engine will pronounce an utterance. Punctuation features can be seen as a substitute for more complex prosodic processing that is not yet available from most speech synthesizers.

A comma in the middle of a sentence will produce a short pause, which will most likely trigger some feedback from the listener. A question mark at the end of the sentence represents a question that should be answered by the listener. When merged with lexical features, the punctuation features can help recognize situations (e.g., yes/no questions) where the listener will most likely use head gestures to answer.

Timing Timing is an important part of spoken language and knowledge about when a specific word is spoken or when a sentence ends influence visual feedback. This information can aid the ECA to anticipate visual grounding feedback. For example, people tend to naturally head nod to the speaker when a pause occurs. The timing of these pauses is important for visual feedback recognition. In natural language processing (NLP), lexical and syntactic features are predominant. In contrast, for face-to-face interaction with an ECA, timing is also an important feature.

Gesture display Synthesized gestures are a key capability of ECAs and they can also be leveraged as a context cue for gesture interpretation. As described in [17], visual feedback synthesis can improve the engagement of the user with the ECA. The gestures expressed by the ECA influence the type of visual feedback from the human participant. For example, if the agent makes a deictic pointing gesture, the user is more likely to look at the location that the ECA is pointing to.

The dialogue manager sends the next spoken utterance, a time stamp and an approximated duration to the visual analysis module. The next spoken utterance contains the words, punctuation, and gesture tags used to generate the ECA's actions. The utterance information is processed by the contextual predictor to extract the lexical, punctuation, timing, and gesture features. Approximate duration of utterances is generally computed by speech synthesizers and made available in the synthesizer API.

4.2 Context in Window System Interaction

We investigate the use of context-based visual feedback recognition to interact with conventional window system components. In Section 2.4.1, we presented a user study with two gesture-based window controls: dialog box acknowledgment/agreement, and document browsing. Our results suggest that head gestures are a natural and efficient way to respond to dialog boxes, especially when the user is already performing a different task. With the proposed approach, notification dialog boxes can be acknowledged by head nodding and question dialog boxes can be answered by head nods or head shakes.

With a window system interface, there are several sources of potential errors with a gesture-based recognition system. We use contextual features to distinguish between visually confounded states and reduce false positives during the interaction with conventional input devices. Contextual features should be easily computed using pre-existing information in the user interface system.

For traditional, window-based interfaces, the interaction context is defined by the event state of the user interface system. We highlight two types of contextual features easily available from the window manager: input device events and display events.

Input device events Recent events from a conventional input device like the keyboard or the mouse can help to determine whether a head gesture is voluntary or not. For example when people search for their cursor on the screen, they perform fast short movements similar to head nods or head shakes, and when people switch attention from the screen back to the keyboard in order to place their fingers on the right keys, the resulting motion can appear like a head nod. These types of false positives can cause difficulty, especially for users who are not aware of the tracking system.

Display events Knowledge from what is displayed on screen can help predicting when a user's input is more likely. A simple example of such a contextual feature

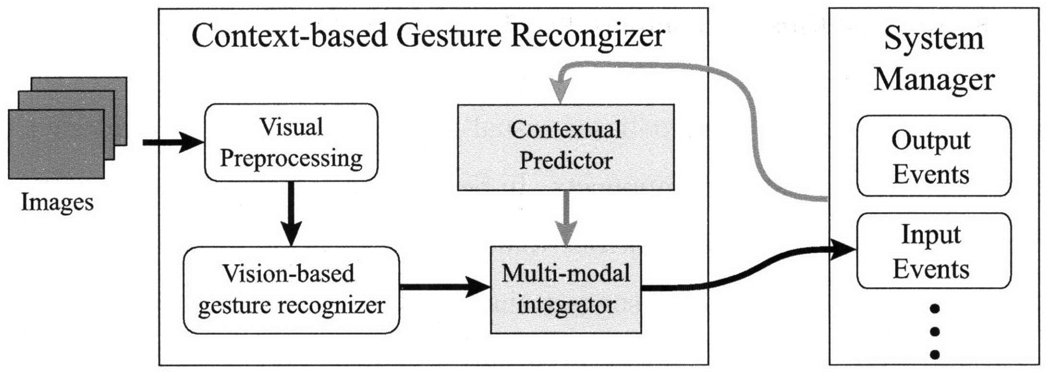


Figure 4-3: Framework for context-based gesture recognition. The contextual predictor translates contextual features into a likelihood measure, similar to the visual recognizer output. The multi-modal integrator fuses these visual and contextual likelihood measures. The system manager is a generalization of the dialogue manager (conversational interactions) and the window manager (window system interactions).

is the time since a dialog box was displayed. This contextual feature can help head gesture recognition because the user's input is most likely after such an event but also because people respond to a dialog box usually after reading its content ($\sim$2.5 second delay in our experiments). So the time since the display event can be as important as the event itself.

These contextual features can be easily computed by listening to the input and output events sent inside the message dispatching loop of the application or operating system (see Figure 4-3).

4.3 Context-based Gesture Recognition

We use a two-stage discriminative classification scheme to integrate interaction context with visual observations and detect gestures. A two-stage scheme allows us the freedom to train the context predictor and vision-based recognizer independently, potentially using corpora collected at different times. Figure 4-3 depicts our complete system.

Our context-based recognition framework has three main components: vision-based recognizer, contextual predictor and multi-modal integrator. In the vision-based gesture recognizer, we compute likelihood measurements of head gestures. In the contextual predictor, we learn a measure of the likelihood of certain visual gestures given the current contextual feature values. In the multi-modal integrator, we merge context-based predictions with observations from the vision-based recognizer to compute the final recognition estimates of the visual feedback.

The input of the contextual predictor is a feature vector $\mathbf{x}_j$ created from the concatenation of all contextual features at frame j. Each contextual value is a real value encoding a specific aspect of the current context. For example, one contextual feature can be a binary value (0 or 1) telling if the last spoken utterance contained a question mark. The details on how these contextual features are encoded are described in Section 4.4 for both conversational interfaces and window system interfaces.

The contextual predictor should output a likelihood measurement at the same frame rate as the vision-based recognizer so the multi-modal integrator can merge both measurements. For this reason, feature vectors $\mathbf{x}_j$ should also be computed at every frame j (even though the contextual features do not directly depend on the input images). One of the advantages of our late-fusion approach is that, if the contextual information and the feature vectors are temporarily unavailable, then the multi-modal integrator can recognize gestures using only measurements made by the vision-based recognizer. It is worth noting that the likelihood measurements can be a probabilities (as output by FHCRFs) or a "confidence" measurement (as output by SVMs).

As shown in 4-3, the vision-based gesture recognizer takes inputs from a visual pre-processing module. This module performs two main tasks: tracking head gaze and estimating eye gaze. To track head gaze, we use the adaptive view-based appearance model (AVAM) described in Section 3.1. This approach has the advantage of being able to track subtle movements of the head for a long periods of time. While the tracker recovers the full 3-D position and velocity of the head, we found features based on angular velocities were sufficient for gesture recognition. To estimate eye gaze, we use the view-based appearance model described in Section 3.2. The eye gaze estimator uses the input image as well as the head gaze estimate to compute the eye gaze of the user. The eye gaze estimates and head rotational velocity are passed to the vision-based gesture recognizer.

The multi-modal integrator merges context-based predictions with observations from the vision-based recognizer. We adopt a late fusion approach because data acquisition for the contextual predictor is greatly simplified with this approach, and initial experiments suggested performance was equivalent to an early, single-stage integration scheme. Most recorded interactions between human participants and conversational robots do not include estimated head position; a late fusion framework gives us the opportunity to train the contextual predictor on a larger data set of linguistic features.

Our integration component takes as input the margins from the contextual predictor described earlier in this section and the visual observations from the vision-based head gesture recognizer, and recognizes whether a head gesture has been expressed by the human participant. The output from the integrator is further sent to the dialogue manager or the window manager so it can be used to decide the next action of the ECA or to trigger the window-based interface.

We experimented with two approaches for discriminative classification: Frame-based Hidden-state Conditional Random Fields (FHCRF) and Support Vector Machines (SVMs). Our goal was to show how our approach generalizes to different learning algorithms. The contextual predictor and the multi-modal integrator were trained using these algorithms.

FHCRF Using our FHCRF model, the likelihood measurement for a specific gesture g (i.e. `head-nod` or `non-gestures`) is equal to the marginal probability $P(y_j = g \mid \mathbf{x}, \theta^*)$. As described in the subsection **Inference** of Section 3.3.2, this probability is equal to the sum of the marginal probabilities for the hidden states part of the subset $\mathcal{H}_g$:

$$P(y_j = g \mid \mathbf{x}, \theta^*) = \sum_{\mathbf{h}:\forall h_j \in \mathcal{H}_g} P(\mathbf{h} \mid \mathbf{x}, \theta^*) \tag{4.1}$$

where $\mathbf{x}$ is the concatenation of all the feature vectors $\mathbf{x}_j$ for the entire sequence and θ^* are the model parameters learned during training. When testing offline, the marginal probabilities are computed using a forward-backward belief propagation algorithm. To estimate the marginal probabilities online, it is possible to define $\mathbf{x}$ as the concatenation of all feature vectors up to frame j and use the forward-only belief propagation algorithm. Note that our experiments were done offline on pre-recorded video sequences.

SVM Using Support Vector Machine (SVM), we estimate the likelihood measurement of a specific visual gesture using the margin of the feature vector $\mathbf{x}_j$. During training, the SVM finds a subset of feature vectors,called support vectors, that optimally define the boundary between labels. The margin $m(\mathbf{x}_j)$ of a feature vector $\mathbf{x}_j$ can be seen as the distance between the $\mathbf{x}_j$ and the boundary, inside a kernel space $\mathcal{K}$. The margin $m(\mathbf{x}_j)$ can easily be computed given the learned set of support vectors $\mathbf{x}_k$, the associated set of labels y_k and weights w_k, and the bias b:

$$m(x) = \sum_{k=1}^{l} y_k w_k K(\mathbf{x}_k, \mathbf{x}_j) + b \tag{4.2}$$

where l is the number of support vectors and $K(\mathbf{x}_k, \mathbf{x}_j)$ is the kernel function. In our experiments, we used a radial basis function (RBF) kernel:

$$K(\mathbf{x}_k, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_k - \mathbf{x}_j\|^2} \tag{4.3}$$

where γ is the kernel smoothing parameter learned automatically using cross-validation on our training set.

4.4 Contextual Features

In this section we describe how contextual information is processed to compute feature vectors $\mathbf{x}_j$. In our framework, contextual information is infered from the input and output events of the *system manager* (see Figure 4-3). The system manager is a generalization of the dialogue manager (conversational interactions) and the window manager (window system interactions). In this type of architecture, the system manager receives input events from external modules or devices (e.g., a keystroke event or a mouse click event) and sends events to other modules (e.g., the next spoken utterance or dialog display event).

We tested two approaches to send event information to the contextual predictor: (1) an active approach where the system manager is modified to send a copy of each relevant event to the contextual predictor, and (2) a passive approach where an external module listens at all the input and output events processed by the system manager and a copy of the relevant events is sent to the contextual predictor. In the contextual predictor, a pre-processing module receives the contextual events and outputs contextual features. Note that each event is accompanied by a timestamp and optionally a duration estimate.

In our framework, complex events are split into smaller sub-events to increase the expressiveness of our contextual features and to have a consistent event formatting. For example, the next spoken utterance event sent from the conversational manager will be split into sub-events including words, word pairs and punctuation elements. These sub-events will include the original event information (timestamp and duration) as well as the relative timing of the sub-event.

The computation of contextual features should be fast so that context-based recognition can happen online in real-time. We use two types of functions to encode contextual features from events: (1) binary functions and (2) ramp functions.

A contextual feature encoded using a binary function will return 1 when the event starts and 0 when it ends. This type of encoding supposes that we know the duration of the event or that we have a constant representing the average duration. It is well suited for contextual features that are less time sensitive. For example, the presence of the word pair "do you" in an utterance is a good indication of a yes/no question but the exact position of this word pair is not as relevant.

A ramp function is a simple way to encode the time since an event happened. We experimented with both negative slope (from 1 to 0) and positive slope (from 0 to 1) but did not see any significant difference between the two types of slopes. A ramp function is well suited for contextual features that are more time sensitive. For example, a dialog box is most likely to be answered shortly after it is displayed.

The following two sub-sections give specific examples of our general framework for contextual feature encoding applied to conversational and non-conversational interfaces. In Section 4.4.1, we describe how contextual information from the dialog manager is encoded and show an example of the output from the contextual predictor after learning to anticipate head nods and head shakes. In Section 4.4.2, we describe how we encode the input and output events from the window-based interface to create contextual feature vectors.

4.4.1 Conversational Interfaces

The contextual predictor receives the spoken avatar's spoken utterance and automatically processes them to compute contextual features. Four types of contextual features are computed: lexical features, prosody and punctuation features, timing information, and gesture displays. In our implementation, the lexical feature relies on extracted word pair (two words that occur next to each other, and in a particular order) since they can efficiently be computed given the transcript of the utterance.

While a range of word pairs may be relevant to context-based recognition, we currently focus on the single phrase "do you". We found this feature is an effective predictor of a yes/no question in many of our training dialogues. Other word pair features will probably be useful as well (for example, "have you, will you, did you"), and could be learned from a set of candidate word pair features using a feature selection algorithm.

We extract word pairs from the utterance and set the following binary feature:

$$f_{\text{``do you''}} = \begin{cases} 1 & \text{if word pair ``do you'' is present} \\ 0 & \text{if word pair ``do you'' is not present} \end{cases}$$

The punctuation feature and gesture feature are coded similarly:

$$f_? = \begin{cases} 1 & \text{if the sentence ends with "?"} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\text{look\_left}} = \begin{cases} 1 & \text{if a "look left" gesture happened during the utterance} \\ 0 & \text{otherwise} \end{cases}$$

The timing contextual feature f_t represents proximity to the end of the utterance. The intuition is that verbal and non-verbal feedback most likely occurs at pauses or just before. This feature can easily be computed given only two values: t_0, the utterance start-time, and δ_t, the estimated duration of the utterance. Given these two values for the current utterance, we can estimate f_t at time t using:

$$f_t(t) = \begin{cases} 1 - \left|\frac{t-t_0}{\delta_t}\right| & \text{if } t \leq t_0 + \delta_t \\ 0 & \text{if } t > t_0 + \delta_t \end{cases}$$

We selected our features so that they are topic independent. This means that we should be able to learn how to predict visual gestures from a small set of interactions and then use this knowledge on a new set of interactions with a different topic discussed by the human participant and the ECA. However, different classes of dialogues might have different key features, and ultimately these should be learned using a feature selection algorithm (this is a topic of future work).

The contextual features are evaluated for every frame acquired by the vision-based recognizer module. The lexical, punctuation and gesture features are evaluated based on the current spoken utterance. A specific utterance is active until the next spoken utterance starts, which means that in-between pauses are considered to be part of the previous utterance. The top three graphs of Figure 4-4 show how two sample utterances from the `MelHead` dataset (from the user study described in Section 2.3.1) will be coded for the word pair "do you", the question mark and the timing feature.

Figure 4-4 also displays the output of our trained contextual predictor for an-

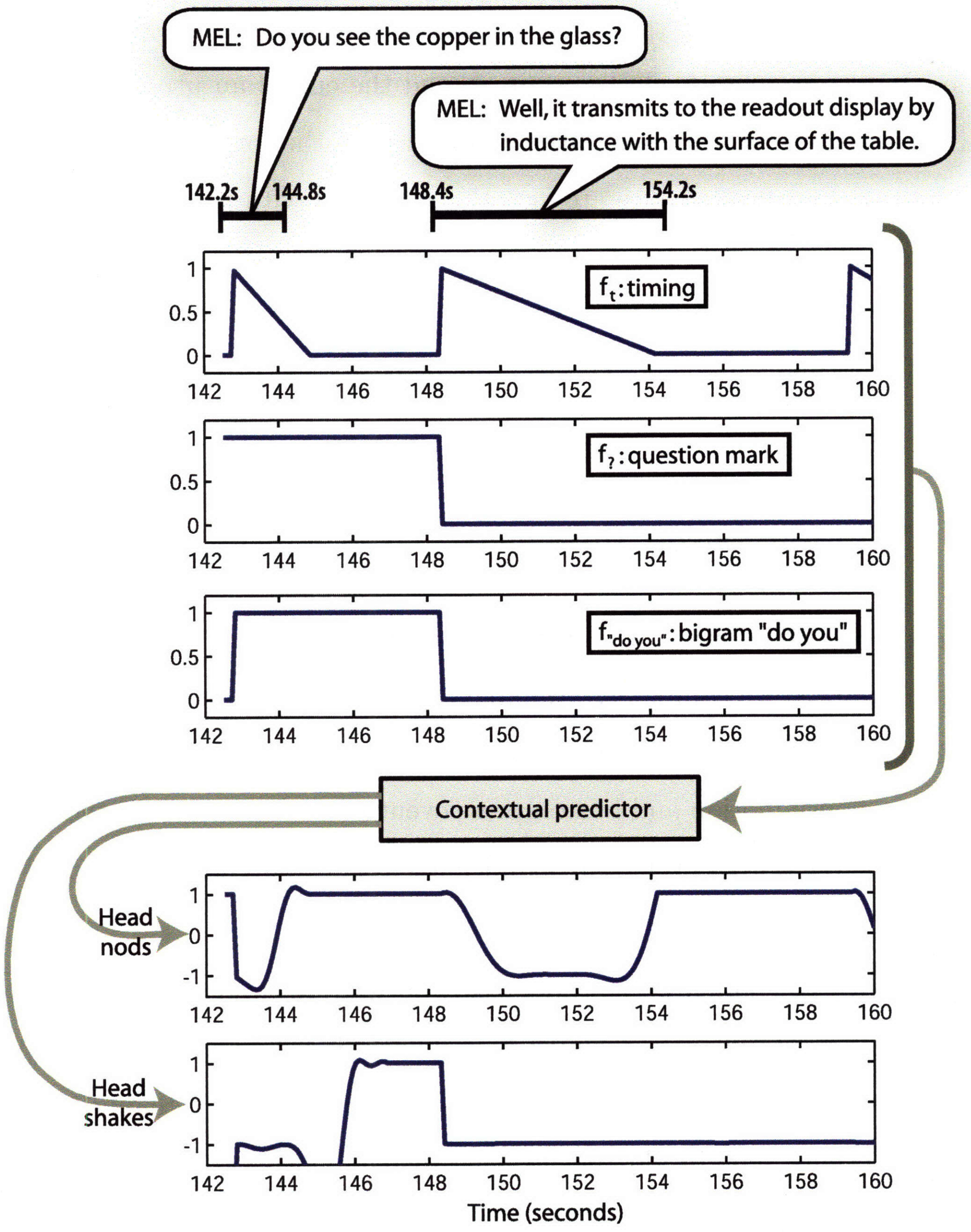


Figure 4-4: Prediction of head nods and head shakes based on 3 contextual features: (1) distance to end-of-utterance when ECA is speaking, (2) type of utterance and (3) lexical word pair feature. We can see that the contextual predictor learned that head nods should happen near or at the end of an utterance or during a pause while head shakes are most likely at the end of a question.

ticipating head nods and head shakes during the dialogue between the robot and a human participant (`MelHead` dataset). Positive margins represent a high likelihood for the gesture. It is noteworthy that the contextual predictor automatically learned that head nods are more likely to occur around the end of an utterance or during a pause, while head shakes are more likely to occur after the completion of an utterance. It also learned that head shakes are directly correlated with the type of utterance (a head shake will most likely follow a question), and that head nods can happen at the end of a question (i.e., to represent an affirmative answer) and can also happen at the end of a normal statement (i.e., to ground the spoken utterance).

4.4.2 Window-based Interface

For non-conversational interfaces, such as window-based systems, we exploit two features of interface state commonly available through a system event dispatch mechanism: dialog box display and mouse motion. The time since the most recent dialog box display, mouse motion and button press event is used as input for the contextual predictor. Such a system event can significantly reduce potential false positives that would otherwise occur using only visual features.

We defined two contextual features based on window system event state: f_d and f_m, defined as the time since a dialog box appeared and time since the last mouse event respectively. These features can be easily computed by listening to the input and display events sent inside the message dispatching loop of the application or operating system (see Figure 4-3). We compute the dialog box feature f_d as

$$f_d(t) = \begin{cases} C_d & \text{if no dialog box was shown} \\ t - t_d & \text{otherwise} \end{cases}$$

where t_d is the time-stamp of the last dialog box appearance and C_d is a default value if no dialog box was previously shown. In the same way, we compute the mouse feature f_m as

$$f_m(t) = \begin{cases} C_m & \text{if no mouse event happened} \\ t - t_m & \text{otherwise} \end{cases}$$

where t_m is the time-stamp of the last mouse event and C_m is a default value if no mouse event happened recently. In our experiments, C_d and C_m were set to 20. The contextual features are evaluated at the same rate as the vision-based gesture recognizer.

4.5 Experiments

We designed our experiments to demonstrate how contextual features can improve visual feedback recognition under the same two axes described in Chapter 2: embodiment and conversional capabilities. Our datasets include interactions with a robot, an avatar and a non-embodied interface. We also experimented with different types of visual feedback: head nods, head shakes and eye gaze aversion gestures. Finally, we tested our context-based recognition framework (described in Section 4.3) with two classification algorithms (SVM and FHCRF) to show the generality of our approach.

The experiments were performed on three different datasets: `MelHead` and `Eye`. For the `MelHead` dataset, the goal is to recognize head nods and head shakes from human participants when interacting with a robot. For the `WidgetsHead` dataset, the goal is to recognize head nods from human participants when interacting with gestures-based widgets. For the `AvatarEye` dataset, the goal is to recognize eye gestures (gaze aversion) from human participants when interacting with a virtual agent. For the `MelHead` and `WidgetsHead` datasets, multi-class SVMs were used to train and test the contextual predictor and multi-modal integrator while FHCRF models were used for the `AvatarEye` dataset. Our goal for using both SVM and FHCRF is to show how our context-based framework generalizes to different learning algorithms.

The following section describes each dataset in more details. Then, Section 4.5.2 explains our experimental procedure. Finally, Section 4.5.3 presents our results and discussion.

4.5.1 Datasets

MelHead This dataset consisted of 16 human participants interacting with Mel, the interactive robot described in Section 2.3.1. The user study consisted of two different scenarios and the 16 participants were randomly assigned to one or the other. In the first scenario, the robot interacted with the participant by demonstrating its own abilities and characteristics. Seven participants interacted with this scenario. In the second scenario, the robot described an invention called iGlassware (~340 utterances). Nine participants interacted with this scenario. The sequences from the first scenario were used for training while the second scenario was used for testing.

The robot's conversational model, based on COLLAGEN [84], determined the next activity on the agenda using a predefined set of engagement rules, originally based on results from a human–human interaction study [91]. Each interaction lasted between 2 and 5 minutes. For ground truth, we hand labeled each video sequence to determine exactly when the participant nodded or shook his/her head. A total of 274 head nods and 14 head shakes were naturally performed by the participants while interacting with the robot.

The contextual cues from the dialogue manager (spoken utterances with start time and duration) were recorded during each interaction and were later automatically processed to create the contextual features, as described in Section 4.4.1.

Head pose tracking was performed online using our adaptive view-based appearance model described in Section 3.1. The rotational velocity signal from the head gaze tracker was transformed into a frequency-based feature by applying a windowed Fast-Fourier Transform (FFT) to each dimension of the velocity independently using a 32-sample, 1-second window. The vision-based head geasture recognizer was a multi-class SVM trained on a previous dataset [67]. This vision-based recognizer was run online during each interaction and the results were logged with the contextual cues.

WidgetsHead This dataset consisted of 12 participants who interacted with the gestures-based widgets described in Section 2.4.1. The video sequences were manually annotated for ground truth. Each frame was labeled as either a head-nod or other-gesture. From this dataset of 79 minutes of interaction, 269 head nods were labeled. All other types of head gestures (i.e. head shakes, look away, no motion,...) were labeled as other-gestures.

The contextual cues from the window system (input and output events with time-stamps) were recorded during each interaction and were later automatically processed to create the contextual features, as described in Section 4.4.2.

As for the `MelHead` dataset, the head pose was also estimated using the adaptive view-based appearance model described in Section 3.1. The vision-based gesture recognizer was also a multi-class SVM [67]. This head geasture recognizer was run online during each interaction and the results were logged with the contextual cues.

AvatarEye This dataset consisted of 6 human participants interacting with a virtual embodied agent as described in Section 2.2.3. Each video sequence lasted approximately 10-12 minutes, and was recorded at 30 frames/sec, for a total of 105,743 frames. During these interactions, human participants would rotate their head up to +/-70 degrees around the Y axis and +/-20 degrees around the X axis, and would also occasionally move their head, mostly along the Z axis.

The dataset was labeled with the start and end points of each gaze aversion gestures. Each frame was labeled either as gaze-aversion or as other-gesture which included section of video where people were looking at the avatar or performing deictic gestures. As for the `MelHead` dataset, the contextual cues from the dialogue manager (spoken utterances with start time and duration) were recorded during each interaction and were later automatically processed to create the contextual features necessary for the contextual predictor. Section 4.4.1 shows how the contextual features are automatically computed.

For each video sequence, the eye gaze was estimated using the view-based appearance model described in Section 3.2.3 and for each frame a 2-dimensional eye gaze estimate was obtained. The eye gaze estimates were logged online with the contextual cues. For this dataset, the vision-based recognizer is a FHCRF model trained and validated offline on the same training and validation sets used for the contextual predictor and the multi-modal integrator.

4.5.2 Methodology

Our hypothesis was that the inclusion of contextual information within the visual gesture recognizer would increase the number of recognized head nods while reducing the number of false detections. For each dataset, we tested three different configurations: (1) using the vision-only approach, (2) using only the contextual information as input (contextual predictor), and (3) combining the contextual information with the results of the visual approach (multi-modal integration). In the rest of this section, we describe how we trained the contextual predictor and the multi-modal integrator for each dataset.

One experimental goal with the `MelHead` dataset was to see if we can train our contextual predictor and multi-modal integrator with one scenario and test it with a different scenario. For this reason, we split the 16 sequences into two sets based on the scenario. The training and validation set had seven participants while the testing set had 9 participants. Validation was performed using a K-fold cross-validation approach.

For the experiments with the `WidgetsHead` and the `AvatarEye` datasets, we used a leave-one-out testing approach where one sequence is held out for testing while all other sequences are used for training and validation, and this process is repeated for each sequence. For both datasets, we did K-fold cross-validation and afterward retrained the model on all training sequences using the optimal set of parameters.

SVMs were used with the `MelHead` and `WidgetsHead` datasets to train the contextual predictor and multi-modal integrator. Two parameters were validated for each SVM: C, the penalty parameter of the error term in the SVM objective function, and γ, the parameter for the RBF kernel. Both parameters were validated with values ranging from 0.001 to 1000 on logarithmic scale.

FHCRF was used with the `AvatarEye` dataset to train the contextual predictor and multi-modal integrator. We validated two parameters for each FHCRF model: the number of hidden states per label and the regularization term. We varied the number of hidden state from 1 to 4 states per label. The regularization term was validated with values ranging from 0.001 to 1000 on the logarithmic scale.

The `AvatarEye` training sets were preprocessed to create a smaller training set with equal number of `gesture` and `non-gesture` subsequences. Each `gesture` subsequence included one ground truth gesture from the training dataset. Since we are interested in learning the transition between gestures, these `gesture` subsequences also included non-gesture frames before and after the ground truth gesture. The size of the gap before and after the gesture was randomly picked between 2 and 50 frames. `Non-gesture` subsequences were randomly extracted from the original sequences of the training set. Every frame in these `non-gesture` subsequences had the same `non-gesture` label. The length of these subsequences varied between 30-60 frames. The resulting training set had the same number of `gesture` and `non-gesture` subsequences, which is equal to the number of ground truth gestures in the original training set.

The training set for `MelHead` and `WidgetsHead` datasets were preproceseed to create a smaller training set with an equal number of `gesture` and `non-gesture` frames. Since SVMs do not model the dynamics between frames, the number of the subsequences does not matter as much as the number of frames. `Gesture` and `non-gesture` samples were randomly picked from the original training set to have the same number for both class labels.

4.5.3 Results and Discussion

In this section, we first present and discuss the results from the head gesture experiments with `MelHead` and `WidgetsHead` datasets, then show the results for the eye gesture experiment with the `AvatartEye` dataset.

MelHead

Figure 4-5 shows the head nod recognition results for a sample dialogue. When only vision is used for recognition, the algorithm makes the first mistake at approximately $t = 101s$ by detecting a false head nod; visual grounding is less likely during the middle of an utterance. By incorporating contextual information, our context-based gesture recognition algorithm is able to reduce the number of false positives.

We computed the true positive rate using the ratio between the number of detected gestures and the total number of ground truth gestures. A head gesture is tagged as detected if the detector triggered at least once during a time window around the gesture. The time window starts when the gesture starts and ends k seconds after the gesture. The parameter k was empirically set to be the maximum delay of the vision-based head gesture recognizer (1.0 second). For the iGlass dataset, the total numbers of ground truth gestures were 91 head nods and 6 head shakes.

The false positive rate is computed at a frame level using the ratio between the number of falsely detected frames and the total number of non-gesture frames. A frame is tagged as falsely detected if the head gesture recognizer triggers and if this frame is outside any time window of a ground truth head gesture. The denominator is the total of frames outside any time window. For the iGlass dataset, the total number of non-gestures frames was 18,246 frames and the total number of frames for all 9 interactions was 20,672 frames.

Figure 4-6 shows head nod detection results for all 9 subjects used during testing. The ROC curves present the detection performance for each recognition algorithm when varying the detection threshold. The area under the curve for each technique are 0.9482 for the vision only, 0.7691 for the predictor and 0.9678 for the integrator.

Figure 4-7 shows head shake detection results for each recognition algorithm when varying the detection threshold. The area under the curve for each technique is 0.9780 for the vision only, 0.4961 for the predictor, and 0.9872 for the integrator.

Table 4.1 summarizes the results from Figures 4-6 and 4-7 by computing the true positive rates for the fixed negative rate of 0.1. Using a standard analysis of variance

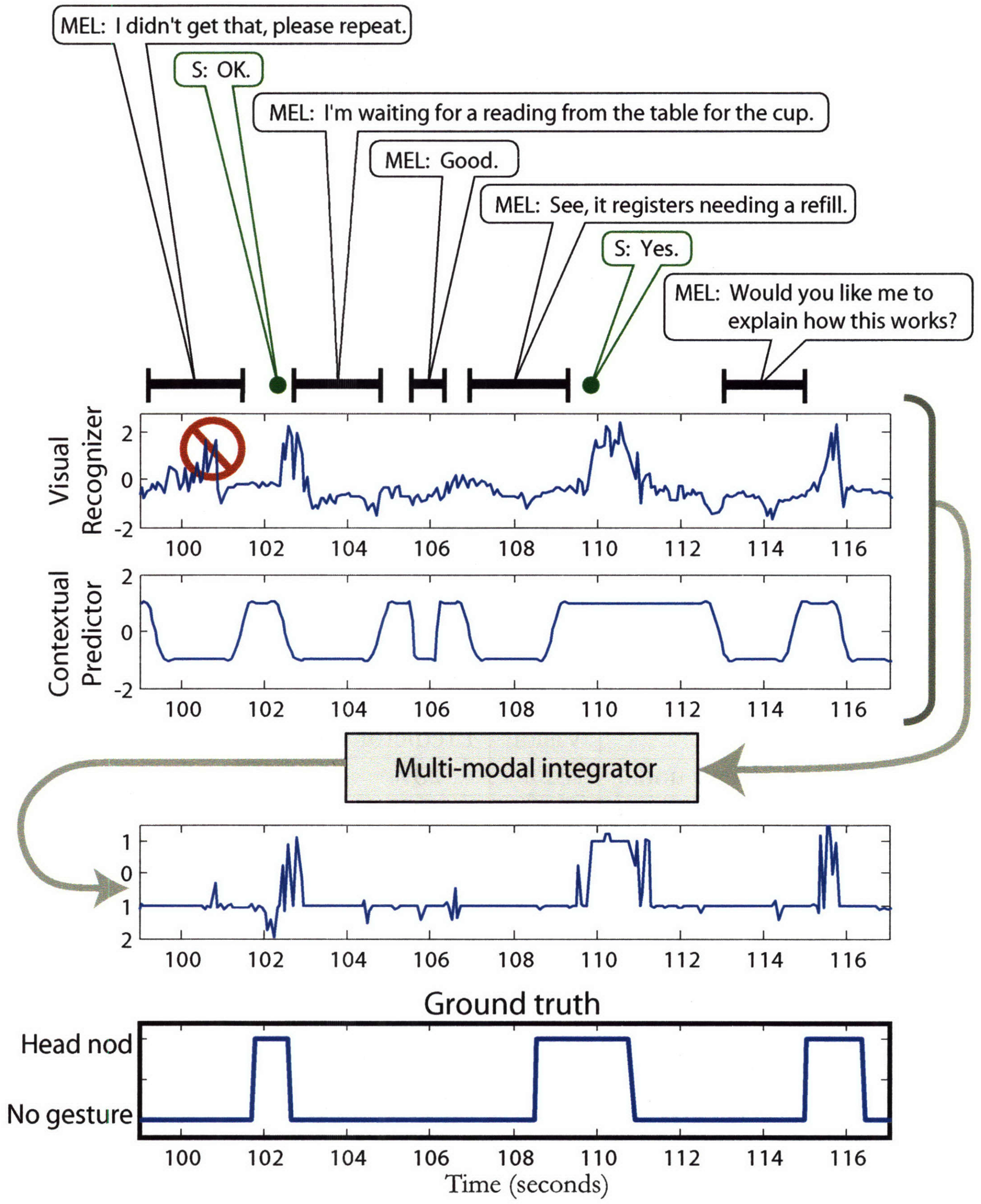


Figure 4-5: Context-based head nod recognition results for a sample dialogue. The last graph displays the ground truth. We can observe at around 101 seconds (circled and crossed in the top graph) that the contextual information attenuates the effect of the false positive detection from the visual recognizer.

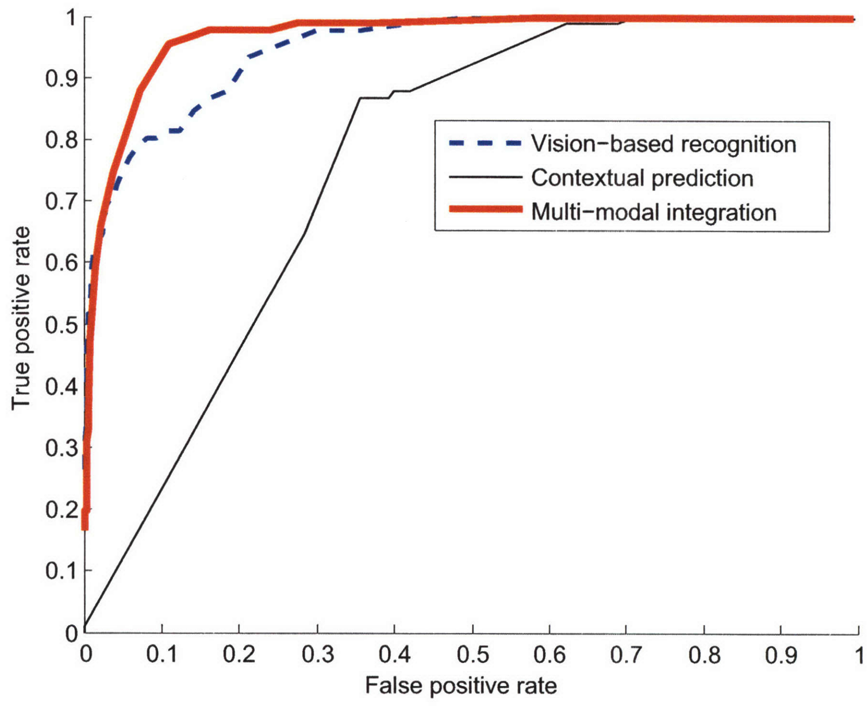


Figure 4-6: Dataset `MelHead`. Head nod recognition curves when varying the detection threshold.

	Vision	Predictor	Integrator
Head nods	81%	23%	**93%**
Head shakes	83%	10%	**98%**

Table 4.1: True detection rates for a fixed false positive rate of 0.1.

(ANOVA) on all the subjects, results on the head nod detection task show a significant difference among the means of the 3 methods of detection: $F(1,8) = 62.40$, $p < 0.001$, $d = 0.97$. Pairwise comparisons show a significant difference between all pairs, with $p < 0.001$, $p = 0.0015$, and $p < 0.001$ for vision-predictor, vision-integrator, and predictor-integrator respectively. For the head shakes results, a larger number of samples would be necessary to attain statistical significance.

WidgetsHead

To analyze the performance of the context-based head gesture recognizer described in Section 4.3 with non-embodied interfaces, we compared the vision-based recognizer

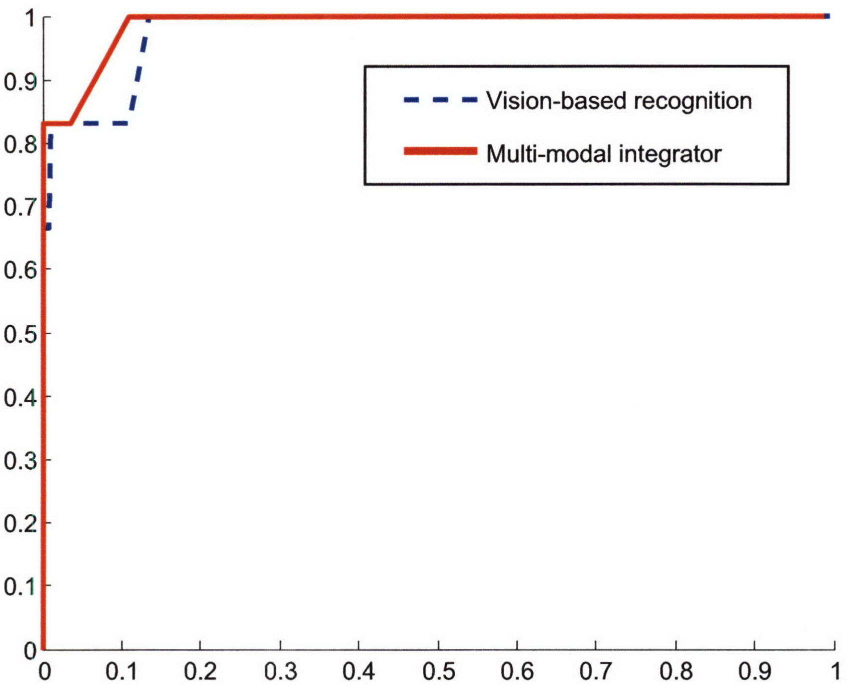


Figure 4-7: Dataset `MelHead`.Head shake recognition curves when varying the detection threshold.

with the multi-modal integrator. Figure 4-8 shows the average head nod detection results for the vision-only technique and the context-based recognizer.

Online, when participants were recorded, the average false positive rate from the vision-based system was 0.058 and the recognition performance was 85.3%. For the same false positive rate, the context-based approach recognized on average 91.0% of the head nods. A paired t-test analysis over all tested subject returns a one-tail p-value of 0.070. Figure 4-8 shows that adding contextual information to the recognition framework does reduce significantly the number of false positives. These results also show that our context-based recognition algorithm can also work with non-conversational interfaces. Adding the input and output events from the window system inside the visual feedback recognition algorithm significantly improved the performance.

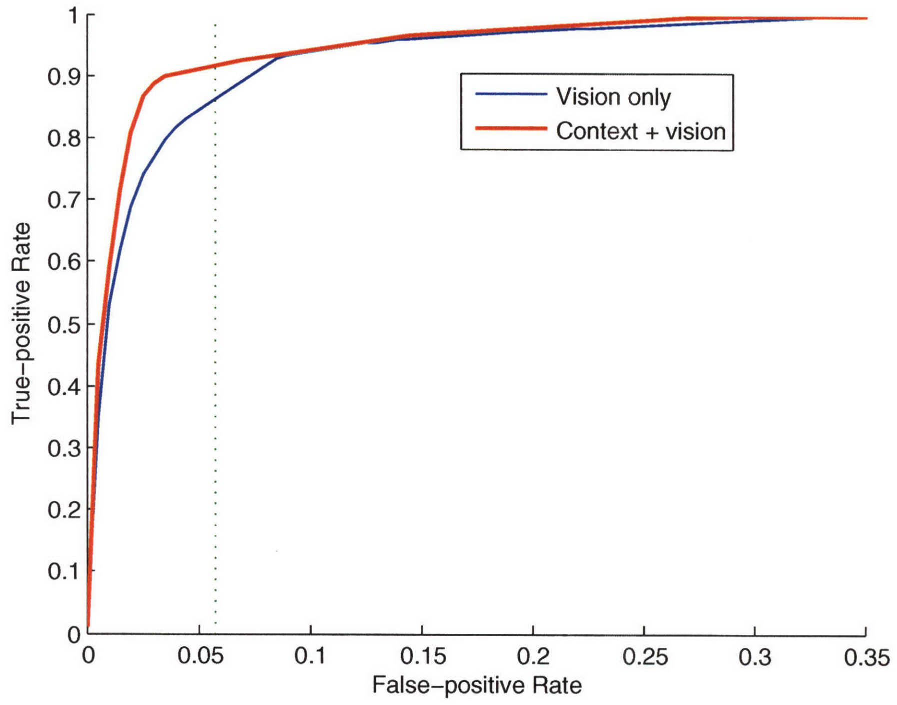


Figure 4-8: Dataset `WidgetsHead`. Average ROC curves for head nods recognition. For a fixed false positive rate of 0.058 (operational point), the context-based approach improves head nod recognition from 85.3% (vision only) to 91.0%.

AvatarEye

The `AvatarEye` dataset was designed to see if we can recognize visual gestures other than head gestures and also to see if our context-based recognition framework can also be applied to a different classification algorithm. This dataset of 6 human participants interacting with an on-screen avatar was used to train three FHCRF models: vision-based recognizer, contextual predictor and multi-modal integrator. Our goal is to recognize eye gaze aversion gestures using the estimated eye gazes from the algorithm described in Section 3.2. The contextual features were computed from spoken utterances of the avatar as described in Section 4.4.1.

Figure 4-9 shows the average gaze aversion recognition results for the vision-only technique and the context-based recognizer. We can see a clear improvement between using only vision versus using both the vision and the contextual information. To evaluate the statistical significance of this results, we show in Table 4.2 three different

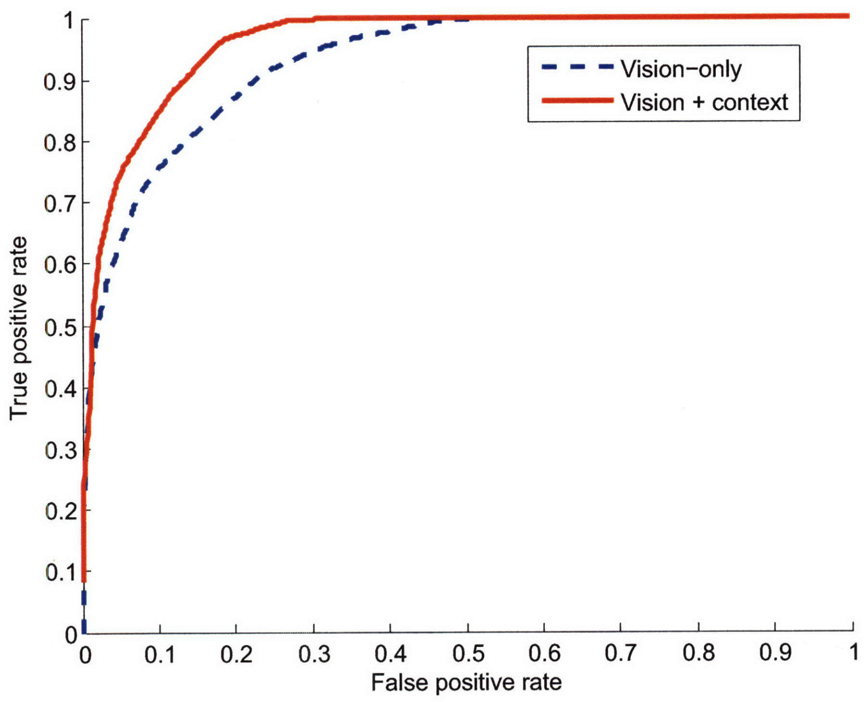


Figure 4-9: Dataset AvatarEye. Average ROC curves for gaze aversion recognition. For a fixed false positive rate of 0.1, the context-based approach improves head nod recognition from 75.4% (vision only) to 84.7%.

measurements: (1) true positive rate for a fixed false positive rate of 0.1, (2) equal error rate, and (3) area under the curve. Each measurement is shown for the vision-based recognizer, the contextual predictor and the multi-modal integrator. The p-value shown next to the vision-based recognizer and the contextual predictor are results from a paired t-test analysis with the multi-modal integrator. It is worth noticing that the difference between the multi-modal integrator and the vision based recognizer is statistically significant for all three measurements: true detection rate at 0.1, equal error rate and area under the curve. For a fixed false positive rate of 0.1, the detection performance improves from 75.4% for the vision-only recognizer to 84.6% for the multi-modal integrator.

Measurements	Vision	Integrator
True detection rate at 0.1	75.4% (p=0.0452)	**84.56%**
Equal error rate	84.7% (p=0.0363)	**89.7%**
Area under the curve	0.927 (p=0.0412)	**0.958**

Table 4.2: Dataset `AvatarEye`. Three analysis measurements to compare the performance of the multi-modal integrator with the vision-based recognizer.

4.6 Summary

Our results show that contextual information can improve user gesture recognition for interactions with embodied conversational agents and interactions with a window system. We presented a prediction framework that extracts knowledge from the spoken dialogue of an embodied agent or from the user-interface system events to predict which visual gesture is most likely. We applied our context-based recognition framework to both head gesture and eye gesture recognition and observe statistical improvement in both cases. By using simple lexical, punctuation, gesture and timing context features, we were able to improve the recognition rate of the vision-only head gesture recognizer from 81% to 93% for head nods and from 83% to 98% for head shakes. Similar improvements were shown when performing context-based recognition during window system interactions. Our experiments with eye gaze aversion recognition showed an improvement from 75.4% to 84.6% when merging conversational cues with the vision observations. We also showed that our context-based recognition framework is general enough to be applicable to SVM classifiers and FHCRF classifiers.

Chapter 5

Conclusion

In this thesis, we introduced new concepts and models that provide human-computer interfaces with the ability to perceive visual feedback. Based on our user studies with virtual and physically embodied agents, we investigate four kinds of visual feedback that are naturally performed by human participants when interacting with interactive interfaces that improve the user experience: head gaze, head gestures (head nods and head shakes), eye gaze and eye gestures (gaze aversions).

We presented the *Adaptive View-based Appearance Model*, a new user independent approach for head gaze estimation which merges differential tracking with view-based tracking. Our results showed that AVAM can track the user's head gaze under large head motion (head rotations of up-to 110 degrees and head translations of up-to 80cm) for a period of several minutes and keep a bounded error under 3 degrees.

We introduced the *Frame-based Hidden-state Conditional Random Field* model, a new discriminative model for visual gesture recognition which can model the substructure of a gesture sequence, learn the dynamics between gesture labels and can be directly applied to label unsegmented sequences. Our results showed that the FHCRF model outperforms previous approaches (i.e. SVM, HMM and CRF) when using a small window size, with equal error rate recognition accuracies of 80.1% and 85.1% for head and eye gesture respectively.

Finally, we introduced the concept of *visual feedback anticipation* where contextual knowledge from the interactive system is analyzed online to anticipate visual feedback from the human participant and improve visual feedback recognition. By using simple lexical, punctuation, gesture and timing context features, we were able to improve the recognition rate of the vision-only head gesture recognizer from 81% to 93% for head nods and from 83% to 98% for head shakes. Our experiments with eye gaze aversion recognition showed an improvement from 75% to 85% using context.

These new models were implemented in a real-time visual feedback recognition library for interactive interfaces (called Watson) that can recognize head gaze, head gestures, and eye gaze using the images of a monocular or stereo camera. The user guide of this recognition library is provided as Appendix A. Watson has been downloaded by more then 70 researchers around the world and was successfully used by MERL, USC, NTT, Media Lab and many other research groups.

5.1 Future Work

As future work, we would like to test our FHCRF model with other visual gesture recognition tasks such as arm gesture recognition. Even though we applied our model to visual gesture recognition, FHCRF is a general discriminative classification technique that can be applied to other non-vision problems. It will be interesting to test the FHCRF model with other sequence labeling problems that exhibit intrinsic and extrinsic sequence dynamics such as phoneme recognition and activity detection.

We also plan to carry out more research with our context-based visual recognition framework. As a first step, we will test our framework with a larger set of contextual features. For example, a larger set of features can be constructed using a conversational interface by considering all the possible word and word pair features computed from the ECA's spoken utterances. In the natural language processing community CRFs have been successfully used for performing text classification using a large number of input features. Similarly, we expect that our FHCRF model will be able to learn salient contextual features when trained using a large input feature set.

In our framework, contextual features from a conversational agent are computed from its current spoken utterance. This is a general approach for contextual feature extraction that is applicable to a wide variety of dialogue system architectures. An interesting extension of our work is the use of contextual features taken from the internal state of the dialogue manager. With this approach, the contextual features are more directly related to the goals of the avatar; for example, the avatar's goal may be for the user to look left at an object of interest or to ask the user what to do next.

In Chapter 4, we showed that contextual information improves visual feedback recognition. While this improvement was statistically significant, it will be interesting to see how this improvement translates when evaluating the engagement between a human participant and an avatar. As noted in some our early user studies (e.g., `look-to-talk`), accurate visual feedback recognition improves the user experience. A user study that compares interactive interfaces built with and without context-based recognition will give us a better understanding of the usefulness of visual feedback anticipation.

One of the main novelties of our approach is that we use contextual knowledge from the avatar to improve visual feedback. A complimentary approach to our framework is the *pure* multi-modal approach where audio and video streams from the user are combined to improve recognition. Examples of these types of approaches are audio-visual speech recognition and the use of prosody from the user's speech to improve head gesture recognition [33]. Another interesting avenue of future work is to explore ways to generalize our approach to include extra modalities such as speech and user prosody.

Appendix A

Watson User Guide

In this appendix, we present the user guide of our real-time visual feedback recognition library. This library, called Watson, can estimate head gaze, eye gaze and head gestures from a live camera (monocular USB or stereo camera) or from pre-recorded sequences.

Version

WATSON

Head Tracking and Gesture Recognition Library

User
Guide

WATSON

User Guide

Louis-Philippe Morency
lmorency@csail.mit.edu

Version 2.1a

August 17th 2006

Table of Contents

Copyright

Chapter 2

Introduction

Watson is a keyframe-based tracking library that uses stereo or monocular images to track the position and orientation of a rigid object.

This guide presents the functionalities and characteristics of Watson tracking library[1]. Watson has been originally created to estimate the position and orientation of the head using a stereo camera but the current version also work with monocular cameras. The current version can track for a long period of time with bounded drift the 6 degrees-of-freedom of the head. Also, the tracker can be reconfigured to estimate the pose of any rigid object and to estimate ego-motion when the background is static.

To get good precision and reduce the possible drift, Watson implements Adaptive View-based Appearance Models technique described in [2] which acquires keyframes of the object online during the tracking. These keyframes represent the object in different pose. When the trajectory of the object crosses one of the recorded keyframe, the pose estimation algorithm will take in account the pose of the keyframe. The pair-wise pose estimation is done using a hybrid technique [3] that combines Iterative Closes Point and Normal Flow Constraint. The complete system can track object for a long period of time with bounded drift.

The following chapter explains the installation procedure for the tracking library. Chapter 3 explains the different parameters of the software. Chapter 4 presents the network protocol used to communicate with Watson via TCP/IP sockets.

Chapter

Installation

Watson can be easily installed on a Microsoft Windows system using the IntallShield installation package. After running the setup, you will need to calibrate you camera if you want to use the tracking system in real-time.

Setup Programs

Main setup

The core installation file watson-x.xx.exe will copy on your machine the following components:

- Watson\bin: Watson demo program (Watson.exe) and DLLs necessary to run the application;
- Watson\Classifier: Features for the frontal and side-view face detectors as well as the eye detectors;
- Watson\HMMs: Learned Hidden Markov Models (HMMs) for head nods and head shakes detection;
- Watson\SVMs: Learned Support Vector Machines (SVMs) for head nods and head shakes detection;
- Watson\EigenSpaces: Learned eigen spaces for eye gaze estimation;
- Watson\include: Include files for the C++ interface;
- Watson\lib: Libraries for the C++ interface;
- Watson\Samples: Samples programs for Watson C++ interface;

- Watson\Sequences\SRI: Configuration files for running the tracker online with a Videre Design stereo camera.

- Watson\Sequences\USB: Configuration files for running the tracker online with a USB monocular camera.

- \Watson\Sequences\ExempleStereo: Pre-recorded stereo video sequence. This sequence can be used to test if the installation of the demo program has been done correctly or to run the tracker with different settings.

- \Watson\Sequences\ExempleAVI: Pre-recorded monocular video sequence. This directory shows how to use Watson to track the head position and orientation from a AVI movie file

Before to be able to run Watson online (directly from a stereo camera), you will need to setup your stereo camera and calibrate it. The following section explains how to do it when you are using a Videre Design stereo camera.

Libraries installed

Watson has been coded to take takes advantage of the MMX and SSE capacity of the Pentium 3 and Pentium 4. When you install the demo program, different DLLs are copied in the \Watson\bin directory

- Intel Integrated Performance Primitives 5.1
- Intel Math Kernel Library 8.1.1
- Intel Open Source Computer Vision Library 1.0
- Small Vision System 4.3a
- GLUT 3.7

If you already installed one of those libraries on your computer and have problem to run Watson, you should check your PATH variable to be sure that there is no conflict between different versions.

We use Qt as our GUI interface because of its speed, simplicity and compatibility with Linux (and now Macintosh too!). We use the version 3.2.3 of Qt. For 3D display we use OpenGL and its extension, GLUT 3.7.

Software updates

When you are updating Watson, most files will be replaced with the newest version. For the sub-directories of \Watson\Sequences, only the files ParamWatson.cfg will be updated. For this reason, you should put all your personalized parameters in ParamWatsonUser.cfg. This way you will be able to use the latest default parameters from ParamWatson.cfg but keep your personalized parameters.

Monocular Camera Calibration

USB camera

Starting with version 2.0, Watson can now track the head pose using a normal USB webcam. To optimize the tracking, some parameters most be set in the control panel of your USB camera. The most important setting to change is the automatic brightness and gain parameters. This parameter should be turned off so that the brightness doesn't change during the recording. This will improve the performance of the optical flow estimation during tracking.

Stereo Camera Calibration

Videre Design

The tracking system has been extensively tested with the Mega-D stereo camera from Videre Design. Recently, the system has been modified to handle the new DCS model from the same company. The following paragraphs will give you some guideline on how to setup the cameras, for more information, please refer to the user guide installed with the Small Vision System.

The first step is to install the Small Vision System (SVS) using the setup file svs42d.exe.. To be able to install the library, you will need a valid license number (please contact Videre Design if you don't have it). It is preferable to install the library before you plug the stereo camera for the first time since the driver of the camera is installed with the SVS library. When you plug your camera in your firewire card the driver setup should start automatically.

When SVS is installed, you need to specify which type of camera you have. If you have a Mega-D, you should run the batch files \svs42\bin\setup_megad.bat and Start->Programs->Watson->Setup cameras->MegaD. If you have a DCS, you should run the batch files \svs\bin\setup_dcs.bat and Start->Programs->Watson->Setup cameras->DCS. Now you are ready to calibrate your camera.

To start the calibration, run the program \svs\bin\smallvcal.exe. If your installation worked correctly, you should be able to start grabbing images by setting the Input to Video and pressing the button Continuous. You should see the left and right image displayed. Now press the menu button Calibrate… to start the calibration. The software uses 10 images of a check board to estimate the intrinsic and extrinsic parameters of the camera. Please read the SVS manual for details on the calibration procedure. When the calibration is done, save your calibration file in the directory \Watson\Sequences\SRI\. A good name for the calibration file is calib-xxxxx.ini where xxxxx represents the serial number of your camera.

The last step before to be able to grab directly from the stereo camera using Watson demo program, is to modify the parameters files of Watson so it use your new calibration file. To do so, open the file \Watson\Sequences\SRI\ParamWatsonUser.cfg, search the field CONFIG_FILENAME: and modify its value to be the name of your calibration file (calib-xxxxx.ini). Now you are ready run Watson directly from your stereo camera!

Other stereo cameras

Watson tracking system has been also tested on Digiclops stereo cameras but unfortunately the demo program (Watson.exe) can only run with Videre Design stereo cameras. To use Watson tracking library with other stereo camera, please look at the C++ interface described in Chapter 6.

Chapter

Software Interface

The demo program gives you a visual interface to test different settings of Watson library. It gives to the user the flexibility to change most tracking parameters. The tracker can be run online, using images directly from the camera, or offline, using images from a prerecorded sequence stored on disk.

Main software functionalities

Watson tracking system can be run online or offline. When the program starts, Watson automatically looks in the current directory for two parameter files: ParamWatsonUser.cfg and ParamWatson.cfg. These files, as described in the following chapter, contain all the default and user-defined parameters necessary for running the tracker. Watson should be started in a directory where ParamWatson.cfg (ParamWatsonUser.cfg is optional) is present.

Grabbing and Tracking

Watson automatically switch between grabbing and tracking when the AutoInit (CTRL+A) option is activated. To start continuous grabbing press F2 and to stop it press F4. With AutoInit activated, as soon as a face is detected in the image, the tracker will start. At each time step a frame is grabbed, segmented and finally the pose of the object is estimated.

Load Sequence

To load a prerecorded sequence from the demo program, you can select Load Sequence in the Files menu and click on the "ParamSeq.cfg" representing the sequence you want to load (for example \sequences\ExempleStereo\ParamSeq.cfg). This file contains all the calibration information for the sequence. As explained in the next chapter, this process can be automated by modifying the ParamWatsonUser.cfg to point on a specific ParamSeq.cfg.

Output console

The output console gives you some information about the pose estimate (rotation and translation) of the object as well as the results from the head nods detector. The top frame shows the absolute pose of the object. The translation is displayed in millimeters and the rotation is displayed in degrees. The variance gives an idea of the accuracy of the pose. The middle frame represents the displacement between the previous frame and the current frame (velocity of the object). The third frame shows the approximate center of the object (also in millimeters). The last frame shows the results from the head nods and headshakes detectors. The numbers below each button represent the confidence of each detector.

Position, Orientation and Coordinate system

The referential coordinate system is set on the left camera for stereo cameras. It is a right-handed coordinate system where the Z axis point behind the camera, the Y axis point below the camera and the X axis point on the left side (when looking at the camera).

The position returned by the tracker represents the distance between the center of the object and the center of the left camera. The orientation returned by the tracker represents the rotation between the first tracked frame and the current frame. When using the Auto-initialization option, the tracker will start only if it finds a frontal face. Since the first tracked frame is a frontal view, each following frame will be relative to the frontal view.

To compute the absolute orientation of the object, you must apply the rotation [rx, ry, rz] to the initial orientation (frontal view: [0,0,-1]). The rotation notation used by Watson is based on a rotation axis and a rotation around this axis. The norm of the vector a=[rx, ry, rz] represents the amount of rotation in radian. The normalized vector represents the axis of rotation. You can change the notation to a rotation matrix by applying this equation $[R] = [I] + \sin(angle)[\sim axis] + (1-\cos(angle))[\sim axis]^2$ (see [4] for more details). Finally, the absolute orientation can be computed by applying the rotation matrix to the frontal view: orientation = [R]*[0,0,-1].

Parameter console

The parameter console gives a visual interface for most of the parameters of the tracker. You can find a description of those parameters in the following chapter. Also, the complete list of parameters can be found in the file `ParamWatson.cfg`.

Main shortcut keys

F2 - Start continuous grabbing/tracking (also on the toolbar);

F3 – Start continuous grabbing/tracking and record images on disk(also on the toolbar);

F4 - Stop grabbing/tracking (also on the toolbar);

F5 - Show current intensity image;

F6 - Show current depth image;

F7 – Show keyframe intensity image;

F8 – Show keyframe depth image;

CTRL+ 1 - Switch to No Display mode (no OpenGL display);

CTRL+ 2 - Switch to 2D mode;

CTRL+ 3 - Switch to 3D mode;

CTRL+ 0 - Switch between 3D modes: Frontal view or Top View;

CTRL+ A – Activate/deactivate the autoinitialization;

CTRL+P – Show/hide the Parameter console;

CTRL+O – Show/hide the Output console;

CTRL+L – Load a new sequence;

CTRL+R – Reload the current sequence;

Parameter Files

The demo program gives you a visual interface to test different settings of Watson library. It gives to the user the flexibility to change most tracking parameters.

Files description

The tracking parameters are kept in 3 different files: ParamWatsonUser.cfg, ParamWatson.cfg and ParamSeq.cfg (or ParamSeqDirect.cfg). ParamWatson.cfg contains the default parameters for the tracker as well as some parameters for the display. You should not modify this file directly since your changes will lost next time you update Watson. Instead, you should enter the parameters you want to modify inside ParamWatsonUser.cfg since this file is never updated by the Installer. ParamSeq.cfg contains all the parameters relative to the grabbing. The parameter files are separated by sections:

- [SECTION_WATSON]: This is the main section of the parameter files. It sets some high-level parameters and specifies the path of other parameter files like ParamSeqDirect.cfg.
- [SECTION_NETWORK]: Set the networking options (client and server) of the demo program.
- [SECTION_HEAD_NODS]: This section sets parameters related to the HMMs (Hidden Markov Model) and SVMs (Support Vector Machines) trained for head nods and head shakes detection.
- [SECTION_MAP_BUILDER]: This section sets the parameters for the keyframes acquisition process. You can set how those keyframes will be acquired (tessellation or clustering) and the gap between each acquired keyframe.
- [SECTION_TRACKER_DIRECTOR]: This section specifies which tracker is activated, sets some main tracking parameters (MATCH_FUNCTION: and UPDATE_POSE:) and let you print some debug information like poses, velocity and center of mass.

- [SECTION_TRACKER_ICP]: Detailed parameters for the default tracker (ICP). Those parameters should be only changed by "advanced" users.
- [SECTION_INIT_TRACKER]: Set some parameters for the tracking initialization and reinitialization.
- [SECTION_SIMPLE_TRACKER]: This section sets parameters for the image segmentation. The setting of those parameters will influence the tracking initialization since only segmented pixels will be used for initialization.
- [SECTION_3D_MODEL]: This section sets parameters for the ellipse matching algorithm used during monocular tracking.
- [SECTION_RECORDER]: Set the default values for the recording option.
- [SECTION_OPEN_GL]: Set the display options of the demo program.
- [SECTION_SEQUENCE]: This section, found usually in ParamSeq.cfg or ParamSeqDirect.cfg, specifies the parameters related to the grabbing/stereo process. Some parameters like SIZE_ROI are used for tracking/segmentation purpose.
- [SECTION_FILES_GRABBER]: This section is used for pre-recorded sequences. It gives all the details about the file format and the camera used to record that sequence. You will usually find this file in ParamSeq.cfg.

Network parameters

The main way to communicate with Watson is via network. All the parameters related to networking are usually set in the section [SECTION_ NETWORK] of ParamWatsonUser.cfg. Watson supports 2 mode of communication: UDP (datagram) or TCP (socket). Also, Watson can be used as a client, a server or both. In the client mode, Watson can send information about the tracking results as well as the grabbed images. In the server mode, Watson receives the images from the network instead of grabbing them from camera or files.

Client mode

When setting up Watson in the client mode, you have to specify three kind of information:

- Which **information** do you want to be sent via network?
- In which **format** do you want the information?
- Which **host** will receive the information?

Currently, Watson can open up to 2 connections. This feature makes it possible to send the results of the head nod detector to one computer while sending the results of the head pose tracker to another computer. The parameter CONNECT_SOCKET: activate/deactivate the connection number 1 and the parameter CONNECT_SOCKET2: activate/deactivate the connection number 2.

Information parameters

The parameter TYPE_INFO_SENT: (or TYPE_INFO_SENT2:) specify which type of information will be sent to the connected computer. After the TYPE_INFO_SENT: tag, you should enumerate all the information tag you want. Each tag must be separated by a space and the line must end by the tag END. Here is a list of information tags available:

- INFO_LINKS: The details of each transformation computed during the tracking will be sent (see Section 5 for more details about the format).
- INFO_PREVIOUS_LINKS_ONLY (can't be used with INFO_LINKS): Equivalent to the velocity. This tag will send the transformation between each consecutive frame (see Section 5 for more details about the format).
- INFO_POSES: Send the absolute pose of the head for each frame.
- INFO_SCREEN_COORDS: Send the estimated projection of the "nose" on the screen. This option can be useful for moving the mouse cursor with your head. The screen is supposed to be parallel to the camera. The parameter SCREEN_POSITION: should be set adequately.
- INFO_CENTERS: The estimated center of mass of the object (in millimeters).
- INFO_HEAD_NODS: Send the results form the head nods detector.
- INFO_FRAME: Send Intensity image, depth image and frame info.
- INFO_INTENSITY: Send Intensity image only.
- INFO_DEPTH: Send Depth image only.

Also, the parameter `SEND_MESSAGE_DURING_TRACKING_ONLY:` can be set to `TRUE` or `FALSE` depending if you want to always receive network message (FALSE) or receive network messages only when the tracking is working (TRUE).

Format parameters

Each message sent via network is in ASCII format (at the exception of the images). A header is sent before each message to specify which information will follow. To define the format of those headers, you can use of those parameters:

- MESSAGE_PREFIX: Prefix used by every message (including images). This can be used to identify the information coming from Watson.
- MESSAGE_LINKS_SUFFIX: This parameter specifies which text should follow the MESSAGE_PREFIX: when a transformation is sent via network.

- MESSAGE_POSES_SUFFIX: This parameter specifies which text should follow the MESSAGE_PREFIX: when a pose is sent.
- MESSAGE_NODS_SUFFIX: This parameter specifies which text should follow the MESSAGE_PREFIX: when a head nods and head shakes detection results are sent via network.

Host parameters

Three parameters should be set to specify the address of your host and the type of connection:

- SOCKET_TYPE: (or SOCKET_TYPE2:) Can be TCP (for socket connection) or UDP (for datagram or connection-less).
- PORT_CONNECTION: (or PORT_CONNECTION2:) This specify the port for connection. Should be the same as your server (listener).
- NAME_HOST: (or NAME_HOST2:) This specify the name of your host. The name can be an IP address (xxx.xxx.xxx.xxx) or a machine name (registered on the DNS server).

Server mode

Watson can receive stereo images from a TCP/IP connection. To activate this option, you must set the parameter CONNECT_SERVER: TRUE. The parameters PORT_SERVER: and NAME_SERVER: set the name of the client that will connect to Watson. Please refer to section 5 for more details on the image format.

Remote commands

Starting with version 1.4, you can now remotely start and stop Watson. To do this, you must connect to Watson (Client or Server mode) and send one of the following commands:

- REINIT: Used to start or restart the tracker. This command is equivalent to the keyboard shortcut F2.
- RECORD: Used to start or restart the tracker and record images on the hard disk. This command is equivalent to the keyboard shortcut F3.
- STOP: Used to stop the tracker. This command is equivalent to the keyboard shortcut F4.

Each command name can be personalized using a prefix common to every commands and a suffix specific to each command. The parameter COMMAND_PREFIX: sets the common prefix to every command. By default, this parameter is an empty string. The parameters COMMAND_REINIT_SUFFIX, COMMAND_RECORD_SUFFIX and COMMAND_STOP_SUFFIX specify the suffix string for each command.

Chapter

Network Interface

Watson demo program gives you a bi-directional network interface to send images, change tracking parameters or gather tracking results.

Server vs. Client

Watson demo program can receive and send information at the same time. Usually, the information received would be tracking parameters, action commands like "Start Tracker" or stereo images grabbed by another program. The information sent by Watson will usually be tracking results like the head position and orientation, its velocity or the head nods and shakes detection results. The current supported formats for network communication are UDP (datagram) or TCP/IP sockets.

Client Protocol

When Watson acting as a client, the demo program will connect to a TCP/IP server (or connectionless, UDP) and start sending information via the socket. If the connection to the server is not initiated at the beginning, it will try to reconnect everytime a image is grabbed. The name of the server, the type of the connection (UDP or TCP/IP) and the type of information sent are all set in ParamWatson.cfg (please refer to chapter 4 for more details). Two type of information can be sent: tracking results and stereo images.

Stereo images transfer

It is possible to use Watson to grab stereo images and send images to a remote system (which could be another instance of Watson) via network. Please refer to the section on Server Protocol for more details about the stereo images format.

Tracking results transfer

After processing each new frame, Watson can optionally send the tracking results via a TCP/IP socket (or UDP datagram). As described in Chapter 4, Watson can send 3 types of tracking results: poses, links, and head nods detection results. All the information sent on the socket will be in ASCII format.

Links format

A link represents the relative pose between two frames. During tracking, Watson computes two kinds of links: link between 2 consecutive frames and link between the current frame and a keyframe. As described in chapter 4, Watson can send all the links or only the consecutive links (also called previous link). Each link is sent using the following format:

```
[LinkTag] [I1] [I2] [var] [tx] [ty] [tz] [rx] [ry] [rz]
```

where

- `[LinkTag]`: Tag sent at the beginning of each link message. This tag can be customized in the parameter file (see chapter 4).
- `[I1]` : Index of the previous frame
- `[I2]` : Index of the current frame
- `[var]`: Variance of the link
- `[tx]`, `[ty]`, `[tz]`: Translation between the previous frame and the current frame (in mm)
- `[rx]`, `[ry]`, `[rz]`: Rotation between the previous frame and the current frame (in rad)

See Chapter 3 for more details on the position and orientation format.

Poses format

The pose represents the position and orientation of the object in a given frame. The pose information sent via network has the following format:

```
[PoseTag] [index] [variance] [tx] [ty] [tz] [rx] [ry] [rz]
```

where

- `[Posetag]`: Tag sent at the beginning of each pose message. This tag can be customized in the parameter file (see chapter 4).
- `[index]` : Integer uniquely describing the frame
- `[variance]`: Variance of the pose
- `[tx]`, `[ty]`, `[tz]`: Position of the object relative to the camera (in mm)
- `[rx]`, `[ry]`, `[rz]`: Orientation of the object relative to the frontal view (in rad)

See Chapter 3 for more details on the position and orientation format.

Nods format

The pose information sent via network has the following format:

```
[NodsTag] [Index] [State] [LogNod] [LogShake]
```

where

- `[Nodstag]`: Tag sent at the beginning of each Nods message. This tag can be customized in the parameter file (see chapter 4).
- `[index]` : Integer uniquely describing the frame
- `[State]`: State of the head nods and head shakes detector. Three possible states:
  - `0` : No head nods of head shakes detected
  - `1` : A head nod has been detected
  - `-1` : A head shake has been detected
- `[LogNod]`: Log likelihood of the HMM trained to detect head nods.
- `[LogShake]`: Log likelihood of the HMM trained to detect head shakes.

Server Protocol

Stereo images transfer

It is possible with Watson to grab stereo images on a remote system and send the images via network. Each stereo images received by Watson will be automatically processed when the transfer is completed.

Frame Header

Each stereo image sent must have the following header (ASCII standard):

```
    F [FrameTag] [FrameIndex] [Focal] [CX] [CY]
```

where

- `[FrameTag]` : describe the type of information following the header. Each item following the header is represented by one capital letter. The order of each letters is not important. Here are the different items possible:
  - `I` : Intensity image for the referential camera
  - `Z` : Depth image for the referential camera
  - `R` : Intensity image of the referential camera
  - `O` : Region of interest of the tracked object

    - P : Pose of the object relative to the camera
- [FrameIndex] : Integer uniquely describing the frame
- [Focal] : Focal length of the referential camera (in pixel)
- [CX] : Center of the image along the X axis (in pixel)
- [CY] : Center of the image along the Y axis (in pixel)

After the header, each item described [FrameTag] must be sent via the network. The order they are sent is not important but the frame will not be processed until all items are received.

Image format

Each image sent have the following format (ASCII standard):

```
    [ImageType] [Width] [Height] [BufferSize]
```

where

- [ImageType]: describe the type of image. Here the different items possible:
    - I : Intensity image for the referential camera;
    - IC : Compressed intensity image for the referential camera (JPEG);
    - Z : Depth image for the referential camera;
    - ZC : Compressed depth image for the referential camera (ZIP);
- [Width] : Width of the image
- [Height] : Height of the image
- [BufferSize] : Size of the (compressed, if specified) buffer (in byte)

Region of interest format

Each region of interest sent have the following format (ASCII standard):

```
ROI [offsetX] [offsetY] [width] [height] [nearZ] [farZ]
```

where

- [offsetX]: Horizontal offset of the region of interest
- [offsetY]: Vertical offset of the region of interest
- [width] : Width of the region of interest
- [height] : Height of the region of interest
- [nearZ] : Near boundary of the region of interest along the Z axis

- `[farZ]` : Far boundary of the region of interest along the Z axis

Pose format

Each pose sent have the following format (ASCII standard):

```
POSE [variance] [tx] [ty] [tz] [rx] [ry] [rz]
```

where

- `[variance]`: Variance of the pose
- `[tx]`, `[ty]`, `[tz]`: Position of the object relative to the camera (in mm)
- `[rx]`, `[ry]`, `[rz]`: Orientation of the object relative to the frontal view (in rad)

See Chapter 3 for more details on the position and orientation format.

Chapter

Programming Interface

Watson offers a C++ interface for the head tracking library and the head gesture recognition library. Using this interface, Watson can be used with different type of stereo cameras.

Sample programs

Watson comes with three sample programs:

- **SimpleSocket:** Shows how to connect to Watson via TCP/IP or UDP and how to receive tracking results. The TCP/IP example also shows how to start/stop Watson demo program remotely;
- **SimpleWatson:** This program shows how to grab images, track the head and detect head gestures using Watson DLL interface;
- **WatsonFromFile:** This program shows how to use Watson DLL interface to read intensity and depth images from disk, insert them into Watson grabbing sequence and track the head pose. This example can be extended to read images from a custom stereo camera.

All three samples program can be found in the directory \Watson\samples\. To compile them, you will need Microsoft Visual C++ .NET 2003 (msvc 7.1). For SimpleWatson and WatsonFromFile, the working directory should be set to ..\..\Sequences\Exemple.

Software architecture

Watson comes with two dynamics libraries:

- **Watson.dll:** This dynamic library contains all the functions related to grabbing and tracking. This is the main library for interfacing with the 3D object tracker. The internal structure of this library is described in the following subsections.

- **NodsShakes.dll:** This dynamic library contains specific functions for head nods and head shakes detection. Tracking results from Watson can directly be used in this library.

C++ Classes Overview

Main classes

The following classes are the main classes needed to interact with Watson:

Name	Inherit from	Description
CWatson		Main interface for the head pose tracker.
CNodsShakes		Main interface for the head gesture recognizer.
CSequence	list<CFrame>	Sequence of stereo images.
CFrame	CIPLImage3D	Stereo image with associated pose, velocity and 3D mesh.
CIPLImage3D		Stereo image with mask and region of interest (ROI).
Transformation		Rigid transformation (Rotation + Translation).
CIPLROI3D	vipiRoi	3D region of interest.
vipImage		Generic 2D image class (see following section)
vipiRoi		2D region of interest
CMesh		3D mesh.
CFaceMatch		Results from the face detector.

vipImage Image Library

This library implements a generic image wrapper for different color mode and storage types. It is based on the Image processing module of Intel Integrated Performance Primitives (IPP) library. The complete library of vipImage can be downloaded on SourceForge.net.

Name	Type	Typical use
vipImage8uC1	unsigned char	Grayscale images, mask images.
vipImage8uC3	unsigned char	Color images
vipImage8uC4	unsigned char	Color images with extra space for Alpha channel
vipImage16sC1	unsigned short	Disparity image

vipImage32fC1	Float	Depth image, X coordinates and Y coordinates.

Parameter Classes

The following classes contain the thresholds and parameters needed to track and recognize head gestures:

Name	Associated class	Description
CParamWatson	CWatson	High level parameters for the head pose tracker.
CParamNodsShakes	CNodsShakes	Parameters for the head gesture recognizer.
CParamSeq	CGrabSequence	Grabbing parameters for the stereo camera and model of the head (ROI).
CParamDirector	CTrackerDirector	Parameters for the online selection of keyframes and merging of the tracking results.
CParamInit	CInitTracker	Initialization criteria for the head tracker.
CParamMap	CMapBuilder	Parameters for the insertion of new keyframes (view-based appearance model).
CParam3DModel	C3DModel	Parameters for the ellipsoid matching algorithm.
CParamTrackerICP	CTrackerICP	Parameters for the core differential tracker.
CRecordParam	CGrabSequence	Record parameters for saving offline sequences.
CParamSimple	CTrackerSimple	Parameters for the face detection and segmentation.

Detailed Interface

The following subsections list and describe the member functions of the most important classes.

CWatson

Grabbing images

- **GrabNewFrame()**: `Utility function that automatically calls AcquireImages, GetImages and InsertImages.`
- **AcquireImages()**: `Acquires the images from internal Grabber.`
- **GetImages()**: `Returns images acquired by the internal Grabber.`
- **InsertImages()**: `Crops the image (if necessary), compute ROI, compute depth and insert frame inside the ImageSequence (calls InsertFrame).`

- **InsertFrame():** Inserts frame (intensity and depth) inside the ImageSequence.

Stereo images can be grabbed automatically using the internal Grabber or inserted manually using one of the Insert function. If you decide to use the internal Grabber (which supports VidereDesign cameras and pre-recorded sequences), you should use the utility function **GrabNewFrame()**. The functions **AcquireImages()**, **GetImages()** and **InsertImages()** can be used if you want to multi-thread the processes of grabbing and stereo. If you decide to insert manually your images (i.e. because you are using a different camera/stereo algorithm), you should use **InsertImages()** or **InsertFrame()**. **InsertImages()** takes as input the left and right images and compute the stereo internally. To work properly, you will need a valid license of Small Vision System (SVS). If you already computed the stereo, then you should use **InsertFrame()** to insert the depth image with its associated intensity image.

Tracking

- **ProcessNewFrame():** Segments face, detects face (if activated) and tracks head.
- **SetMode():** Set tracking state of Watson (see description below).
- **SetAutoDetection():** Activates the face detection for automatic initialization of the head tracker.
- **SetAutoReinit():** Set if the tracker should automatically reinitialize when the user move too fast or not enough valid pixels are present.
- **SetRoi():**

Results

- **GetCurrentFrame():** Returns current frame (with associated pose and velocity).
- **GetFrameSeq():**
- **GetLinkList():**
- **Reset():** Cleans the image sequence and the model (if autoClean == true), and resets the tracker.

Keyframes

- **CleanMap():** Erases all the keyframes from the view-based appearance model.
- **SetAutoClean():** Set if the view-based appearance model should be erased every time the tracker is reinitialized.
- **GetMapSeq():**
- **LoadMap():**
- **SaveMap():**

Recording

- **StartRecording()**
- **StopRecording()**
- **LoadSequence()**
- **SaveSequence()**
- **ReloadSequence()**

Face detector

- **GetNbFaceMatches()**
- **GetListFaceMatches()**
- **GetCommonMask()**
- **DrawBoxes**
- **getCountDown()**

Parameters

- **GetParamWatson()**
- **GetParamInit()**

- `GetParamSimpleTracker()`
- `GetParamRecorder()`
- `GetParamMap()`
- `GetParamDirector()`
- `GetParamICP()`

CFrame

Images

- `GetIntensityImage()`
- `GetIntensityRightImage()`
- `GetColorImage()`
- `GetDepthImage()`
- `GetXImage()`
- `GetYImage()`
- `GetMask()`
- `GetValidDepth()`

Calibration

- `BackProject()`
- `GetFocalLength()`
- `GetImageCenterX()`
- `GetImageCenterY()`
- `GetDeltaX()`
- `GetDeltaY()`

Pose

- `GetPose()`
- `GetVelocity()`
- `GetRoi3D()`
- `GetCenterX()`
- `GetCenterY()`
- `GetCenterZ()`

Pose

- `GetFrameIndex()`
- `GetTimeStamp()`
- `isKeyframe()`

Transformation

- `GetEulerAngle()`
- `GetRotationMatrix()`
- `GetTranslation()`
- `GetPtrTransformationMatrix()`
- `GetVariance()`
- `ApplyTransform()`

CNodsShakes

Detection

- `InsertLink()`
- `Reset()`

- `Enable()`
- `IsEnabled()`

Results

- `GetCurrentState()`
- `GetLLNods()`
- `GetLLShakes()`
- `GetCurrentTimeStamp()`

Chapter

Troubleshooting

In this chapter, we describe solutions to common problems/mistakes happening when installing Watson.

"Can't open frame grabber"

Problem

When starting Watson, a message saying "Can't open frame grabber" is displayed in the DOS prompt and no intensity image (F5) or depth image (F6).

Solution

This error message usually signifies that the stereo camera has not been installed properly.

- Check if the stereo camera is connected☺. You should be able to see a red light from the front of the stereo camera.
- For Videre Design cameras, SVS must be installed before plugging the camera. If you have a Mega-D stereo camera, check the Device manager to be sure that the camera is recognized as a PixelLink™ imaging module.
- Be sure that you are using the appropriate svsgrab.dll file. If you have a Mega-D you should run setup_megad.bat and if you have a DCS, you should execute the file setup_dcs.bat.

"Can't start continuous capture"

Problem

When starting Watson, a message saying "Can't start continuous capture" is displayed in the DOS prompt and no intensity image (F5) or depth image (F6).

Solution

This error message usually signifies that the camera is not responding.

- Unplug and replug the camera. When a program stop during the grabbing process, the camera must be reset.

Bad stereo images or blank stereo image

Problem

The stereo images (F6) looks noisy (or you get a blank image) but you get an intensity image (F5).

Solution

This happens usually if you are using the wrong calibration.

- If you change the lens on your stereo camera or if you get a new camera, you should always recalibrate the stereo camera. Please refer to SVS documentation for more information on how to calibrate your camera.
- When calibrating the camera, be sure to use SVS42d.exe. Some older versions of SVS may also work.
- Be sure that you modified the parameters file ParamSeqDirect.cfg so it uses your new calibration file. To do so, open the file, search the field CONFIG_FILENAME: and modify its value to be the name of your calibration file (calib-xxxxx.ini).

Tracker doesn't initialize

Problem

The images are grabbed properly but the head tracker never starts.

Solution

When in Auto-init mode, the head pose tracker initialize after it detected a face.

- Check that the auto-initialization is turned on. In the demo program, you can press CTRL+A to toggle the auto-init option. In the parameter file, you can set the option AUTO_INIT: TRUE.
- The Adaboost-based face detector uses parameter files placed in the directory \Watson\Classifier. If you receive the error message "Cannot open file classifier.txt to read." During the startup, this means that Watson could not find these files.
- The face detector checks for faces at different scales. You can increase the parameter NUMBER_SCALE: 4 to a larger value so that closer faces are detected.
- When a face is detected, Watson checks that the face is inside a certain depth range. You can modify this range of valid detection using the parameters MIN_DEPTH_MASK and MAX_DEPTH_MASK.

- Finally, Watson will initialize only after a face has been detected for a certain time. You can reduce the number of frame detected using the parameter NB_DETECT_BEFORE_INIT.

Bad head tracking results

Problem

The head is detected but doesn't seem to be tracked properly.

Solution

- Try to increase the gain of the camera. Sometime when the images are too dark, the intensity gradient computed during the tracking become too noisy. Also, some internal parameters for key-frame selection depend on the intensity of the image.
- Be sure that you are using the right calibration file. Watson comes with a default calibration file (calib.ini) which should be replaced by the appropriate calibration file that you created using SVS. The quality of the tracking will improve dramatically if you use the right calibration file for your camera.

Watson crashes during grabbing/tracking

Problem

Watson crashes sometime on Pentium 4 HT.

Solution

- Turn off the hyper-thread option in your BIOS.

References

[1] Louis-Philippe Morency and Trevor Darrell, From Conversational Tooltips to Grounded Discourse: Head Pose Tracking in Interactive Dialog Systems, International Conference on Multimedia Interfaces, College State, PA, 2004

[2] Morency, L.-P., Rahimi, A. and Trevor Darrell, Adaptive View-based Appearance Model, Proceedings of IEEE conference on Computer Vision and Pattern Recognition, 2003

[3] Morency, L.-P., and Trevor Darrell, Stereo Tracking using ICP and Normal Flow, Proceedings of International Conference on Pattern Recognition, 2002

[4] http://www.euclideanspace.com/maths/geometry/rotations/conversions/angleToMatrix/index.htm

Bibliography

[1] M. Assan and K. Groebel. Video-based sign language recognition using hidden markov models. In *Int'l Gest Wksp: Gest. and Sign Lang.*, 1997.

[2] AT&T. *Natural Voices.* http://www.naturalvoices.att.com.

[3] Ruth Aylett and Marc Cavazza. Intelligent virtual environments - state-of-the-art report. In *Eurographics*, 2001.

[4] S. Basu, I. Essa, and A. Pentland. Motion regularization for model-based head tracking. In *In Intl. Conf. on Pattern Recognition (ICPR '96)*, 1996.

[5] S. Basu, I.A. Essa, and A.P. Pentland. Motion regularization for model-based head tracking. In *Proceedings. International Conference on Pattern Recognition*, 1996.

[6] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Patt. Anal. Machine Intell.*, 14(2):239–256, February 1992.

[7] T. Bickmore. Towards the design of multimodal interfaces for handheld conversational characters. In *Preoceedings of the CHI Extended Abstracts on Human factors in Computing Systems Conference*, pages 788–789, 2002.

[8] Tim Bickmore and Justine Cassell. *J. van Kuppevelt, L. Dybkjaer, and N. Bernsen (eds.), Natural, Intelligent and Effective Interaction with Multimodal Dialogue Systems*, chapter Social Dialogue with Embodied Conversational Agents. Kluwer Academic, 2004.

[9] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 232–237, 1998.

[10] M.J. Black and Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion. In *ICCV*, pages 374–381, 1995.

[11] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH99*, pages 187–194, 1999.

[12] M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *CVPR*, 1996.

[13] Breazeal, Hoffman, and A. Lockerd. Teaching and working with robots as a collaboration. In *The Third International Conference on Autonomous Agents and Multi-Agent Systems AAMAS 2004*, pages 1028–1035. ACM Press, July 2004.

[14] C. Breazeal and L. Aryananda. Recognizing affective intent in robot directed speech. *Autonomous Robots*, 12(1):83–104, 2002.

[15] De Carolis, Pelachaud, Poggi, and F. de Rosis. Behavior planning for a reflexive agent. In *Proceedings of IJCAI*, Seattle, September 2001.

[16] J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjalmsson, and H. Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of the CHI'99 Conference*, pages 520–527, Pittsburgh, PA, 1999.

[17] Justine Cassell and Kristinn R. Thorisson. The power of a nod and a glance: Envelope vs. emotional feedback in animated conversational agents. *Applied Artificial Intelligence*, 1999.

[18] M.R.M. Mimica C.H. Morimoto. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98:52–82, 2005.

[19] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. In *Proc. of the IEEE Int. Conf. on Robotics and Authomation*, pages 2724–2728, 1991.

[20] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. Structure from motion causally integrated over time. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(4):523–535, April 2002.

[21] C. Mario Christoudias, Kate Saenko, Louis-Philippe Morency, and Trevor Darrell. Co-adaptation of audio-visual speech and gesture classifiers. In *Proceedings of the International Conference on Multi-modal Interfaces*, November 2006.

[22] H.H. Clark and E.F. Schaefer. Contributing to discourse. *Cognitive Science*, 13:259–294, 1989.

[23] M. Coen. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998.

[24] C. J. Cohen, G. J. Beach, and G. Foulk. A basic hand gesture control system for PC applications. In *Proceedings. 30th Applied Imagery Pattern Recognition Workshop (AIPR'01)*, pages 74–79, 2001.

[25] R. Alex Colburn, Michael F. Cohen, and Steven M. Ducker. The role or eye gaze in avatar mediated conversational interfaces. Technical Report MSR-TR-2000-81, Microsoft Research, July 2000.

[26] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *PAMI*, 23(6):681–684, June 2001.

[27] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *PAMI*, 23(6):681–684, June 2001.

[28] J. Davis and S. Vaks. A perceptual user interface for recognizing head gesture acknowledgements. In *ACM Workshop on Perceptual User Interfaces*, pages 15–16, November 2001.

[29] D. DeCarlo and D. Metaxas. Adjusting shape parameters using model-based optical flow residuals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(6):814–823, June 2002.

[30] Videre Design. *MEGA-D Megapixel Digital Stereo Head.* http://www.ai.sri.com/ konolige/svs/, 2000.

[31] Dillman, Becher, and P. Steinhaus. ARMAR II – a learning and cooperative multimodal humanoid robot system. *International Journal of Humanoid Robotics*, 1(1):143–155, 2004.

[32] Dillman, Ehrenmann, Steinhaus, Rogalla, and R. Zoellner. Human friendly programming of humanoid robots–the German Collaborative Research Center. In *The Third IARP Intenational Workshop on Humanoid and Human-Friendly Robotics*, Tsukuba Research Centre, Japan, December 2002.

[33] Shinya Fujie, Yasuhi Ejiri, Kei Nakajima, Yosuke Matsusaka, and Tetsunori Kobayashi. A conversation robot using head gesture recognition as paralinguistic information. In *Proceedings of 13th IEEE International Workshop on Robot and Human Communication, RO-MAN 2004*, pages 159–164, September 2004.

[34] Atsushi Fukayama, Takehiko Ohno, Naoki Mukawa, Minako Sawaki, and Norihiro Hagita. Messages embedded in gaze of interface agents — impression management with agent's gaze. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 41–48, 2002.

[35] A. M. Glenberg, J. L. Schroeder, and D.A. Robertson. Averting the gaze disengages the environment and facilitates remembering. *Memory and cognition*, 26(4):651–658, July 1998.

[36] J. Gratch, A. Okhmatovskaia, F. Lamothe, S. Marsella, M. Morales, R. J. van der Werf, and L.-P. Morency. Virtual rapport. In *6th International Conference on Intelligent Virtual Agents*, Marina del Rey, CA, 2006.

[37] Z. M. Griffin and K. Bock. What the eyes say about speaking. *Psychological Science*, 11(4):274–279, 2000.

[38] A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt. Hidden conditional random fields for phone classification. In *INTERSPEECH*, 2005.

[39] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *PAMI*, 20(10):1025–1039, October 1998.

[40] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *PAMI*, 20(10):1025–1039, October 1998.

[41] Haptek. *Haptek Player*. http://www.haptek.com.

[42] M. Harville, A. Rahimi, T. Darrell, G.G. Gordon, and J. Woodfill. 3D pose tracking with linear depth and brightness constraints. In *ICCV99*, pages 206–213, 1999.

[43] Craig Hennessey, Borna Noureddin, and Peter Lawrence. A single camera eye-gaze tracking system with free head motion. In *ETRA '06: Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 87–94, 2006.

[44] B.K.P. Horn and B.G. Schunck. Determining optical flow. *AI*, 17:185–203, 1981.

[45] P.J. Huber. *Robust statistics.* Addison-Wesley, New York, 1981.

[46] InterSense Inc. *InertiaCube*^{2}. http://www.intersense.com.

[47] R.J.K Jacob. *Eye tracking in advanced interface design*, pages 258–288. Oxford University Press, 1995.

[48] T. Jebara and A. Pentland. Parametrized structure from motion for 3D adaptive feedback tracking of faces. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[49] A. Kapoor and R. Picard. A real-time head nod and shake detector. In *Proceedings from the Workshop on Perspective User Interfaces*, November 2001.

[50] S. Kawato and J. Ohya. Real-time detection of nodding and head-shaking by directly detecting and tracking the between-eyes. In *Proceedings. Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 40–45, 2000.

[51] A. Kendon. Some functions of gaze direction in social interaction. *Acta Psyghologica*, 26:22–63, 1967.

[52] R. Kjeldsen. Head gestures for computer control. In *Proc. Second International Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-time Systems*, pages 62–67, 2001.

[53] S. Kumar and M. Herbert. Discriminative random fields: A framework for contextual interaction in classification. In *ICCV*, 2003.

[54] M. La Cascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of textured-mapped 3D models. *PAMI*, 22(4):322–336, April 2000.

[55] M. LaCascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of textured-mapped 3D models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(4):322–336, April 2000.

[56] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: probabilistic models for segmenting and labelling sequence data. In *ICML*, 2001.

[57] Sooha Park Lee, Jeremy B. Badler, and Norman I. Badler. Eyes alive. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 637–644, 2002.

[58] Lemon, Gruenstein, and Stanley Peters. Collaborative activities and multi-tasking in dialogue systems. *Traitement Automatique des Langues (TAL), special issue on dialogue*, 43(2):131–154, 2002.

[59] S. Lenman, L. Bretzer, and B. Thuresson. Computer vision based hand gesture interfaces for human-computer interaction. Technical Report CID-172, Center for User Oriented IT Design, June 2002.

[60] Michael Li and Ted Selker. Eye pattern analysis in intelligent virtual agents. In *Conference on Intelligent Virutal Agents(IVA02)*, pages 23–35, 2001.

[61] P. Maglio, T. Matlock, C. Campbell, S. Zhai, and B.A. Smith. Gaze and speech in attentive user interface. In *Proc. of the Third Intl Conf. on Multimodal Interfaces*, Beijing, China, 2000.

[62] P. Majaranta and K. J. Raiha. Twenty years of eye typing: systems and design issues. In *ETRA '02: Proceedings of the 2002 symposium on Eye tracking research & applications*, pages 15–22, 2002.

[63] Philip F. McLauchlan. A batch/recursive algorithm for 3D scene reconstruction. *Conf. Computer Vision and Pattern Recognition*, 2:738–743, 2000.

[64] L.-P. Morency and T. Darrell. Stereo tracking using ICP and normal flow constraint. In *Proceedings of International Conference on Pattern Recognition*, 2002.

[65] L.-P. Morency, A. Rahimi, N. Checka, and T. Darrell. Fast stereo-based head tracking for interactive environment. In *Proceedings of the Int. Conference on Automatic Face and Gesture Recognition*, pages 375–380, 2002.

[66] L.-P. Morency, A. Rahimi, and T. Darrell. Adaptive view-based appearance model. In *CVPR*, 2003.

[67] Louis-Philippe Morency and Trevor Darrell. From conversational tooltips to grounded discourse: Head pose tracking in interactive dialog systems. In *Pro-*

ceedings of the International Conference on Multi-modal Interfaces, College State, PA, October 2004.

[68] Louis-Philippe Morency, Patrik Sundberg, and Trevor Darrell. Pose estimation using 3d view-based eigenspaces. In *ICCV Workshop on Analysis and Modeling of Faces and Gestures*, pages 45–52, Nice, France, 2003.

[69] Nakano, Reinstein, Stocky, and Justine Cassell. Towards a model of face-to-face grounding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, July 2003.

[70] D.G. Novick, B. Hansen, and K. Ward. Coordinating turn-taking with gaze. In *Proceedings of the Fourth International Conference on Spoken Language*, volume 3, pages 1888–1891, 1996.

[71] Nuance. *Nuance.* http://www.nuance.com.

[72] A. Oh, H. Fox, M. Van Kleek, A. Adler, K. Gajos, L.-P. Morency, and T. Darrell. Evaluating look-to-talk: A gaze-aware interface in a collaborative environment. In *CHI 2002*, pages 650–651, Minneapolis, USA, April 2002.

[73] Takehiko Ohno and Naoki Mukawa. A free-head, simple calibration, gaze tracking system that enables gaze-based interaction. In *ETRA '04: Proceedings of the 2004 symposium on Eye tracking research & applications*, pages 115–122, 2004.

[74] N. Oliver, A. Pentland, and F. Berard. Lafter: Lips and face real time tracker. In *Computer Vision and Patt. Recog.*, 1997.

[75] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[76] C. Pelachaud and M. Bilvi. Modelling gaze behavior for conversational agents. In *International Working Conference on Intelligent Virtual Agents*, pages 15–17, September 2003.

[77] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, Seattle, WA, June 1994.

[78] J.B. Pierrehumbert. *The phonology and phonetic of English intonation.* Massachusetts Institute of Technology, 1980.

[79] A. Quattoni, M. Collins, and T. Darrell. Conditional random fields for object recognition. In *NIPS*, 2004.

[80] Pernilla Qvarfordt and Shumin Zhai. Conversing with the user based on eye-gaze patterns. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–230, 2005.

[81] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[82] A. Rahimi, L-P. Morency, and T. Darrell. Reducing drift in parametric motion tracking. In *ICCV*, volume 1, pages 315–322, June 2001.

[83] A. Rahimi, L.P. Morency, and T. Darrell. Reducing drift in parametric motion tracking. In *ICCV01*, volume 1, pages 315–322, 2001.

[84] Rich, Sidner, and Neal Lesh. Collagen: Applying collaborative discourse theory to human–computer interaction. *AI Magazine, Special Issue on Intelligent User Interfaces*, 22(4):15–25, 2001.

[85] Daniel C. Richardson and Michael J. Spivey. Representation, space and hollywood squares: looking at things that aren't there anymore. *Cognition*, 76:269–295, 2000.

[86] D.C. Richardson and R. Dale. Looking to understand: The coupling between speakers and listeners eye movements and its relationship to discourse comprehension. In *Proceedings of the 26th Annual Meeting of the Cognitive Science Society*, pages 1143–1148, 2004.

[87] P. Rowe and A. Kelly. Map construction for mosaic-based vehicle position estimation. In *International Conference on Intelligent Autonomous Systems (IAS6)*, July 2000.

[88] A. Schodl, A. Haro, and I. Essa. Head tracking using a textured polygonal model. In *PUI98*, 1998.

[89] Sidner, Kidd, Lee, and Neal Lesh. Where to look: A study of human–robot engagement. In *Proceedings of Intelligent User Interfaces*, Portugal, 2004.

[90] Sidner, Lee, and Neal Lesh. Engagement when looking: Behaviors for robots when collaborating with people. In *Diabruck: Proceedings of the 7th workshop on the Semantic and Pragmatics of Dialogue*, pages 123–130, University of Saarland, 2003. I. Kruiff-Korbayova and C. Kosny (eds.).

[91] C. Sidner, C. Lee, C.D.Kidd, N. Lesh, and C. Rich. Explorations in engagement for humans and robots. *Artificial Intelligence*, 166(1–2):140–164, August 2005.

[92] Candace Sidner, Christopher Lee, Louis-Philippe Morency, and Clifton Forlines. The effect of head-nod recognition in human-robot conversation. In *Proceedings of the 1st Annual Conference on Human-Robot Interaction*, March 2006.

[93] M. Siracusa, L.-P. Morency, K. Wilson, J. Fisher, and T. Darrell. Haptics and biometrics: A multimodal approach for determining speaker location and focus. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, November 2003.

[94] C. Sminchisescu, A. Kanaujia, Z. Li, and D. Metaxas. Conditional models for contextual human motion recognition. In *Int'l Conf. on Computer Vision*, 2005.

[95] M. J. Spivey and J. J. Geng. Oculomotor mechanisms activated by imagery and memory: Eye movements to absent objects. *Psychological Research/Psychologische Forschung*, 65(4):235–241, 2001.

[96] T. Starner and A. Pentland. Real-time asl recognition from video using hidden markov models. In *ISCV*, 1995.

[97] G. Stein and A. Shashua. Direct estimation of motion and extended scene structure from moving stereo rig. In *Proc. of CVPR*, June 1998.

[98] R. Stiefelhagen. Tracking focus of attention in meetings. In *Proceedings of International Conference on Multimodal Interfaces*, 2002.

[99] R. Stiefelhagen, J. Yang, and A. Waibel. Estimating focus of attention based on gaze and sound. In *Workshop on Perceptive User Interfaces (PUI 01)*, Orlando, Florida, 2001.

[100] C. Sutton, K. Rohanimanesh, and A. McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the International Conference on Machine Learning*, 2004.

[101] Y. Takemae, K. Otsuka, and N. Mukaua. Impact of video editing based on participants' gaze in multiparty conversation. In *Extended Abstract of CHI'04*, April 2004.

[102] A. L. Thomaz, M. Berlin, and C. Breazeal. An embodied computational model of social referencing. In *In IEEE International Workshop on Human Robot Interaction (RO-MAN)*, 2005.

[103] A. Torralba, K. Murphy, and W. Freeman. Contextual models for object detection using boosted random fields. In *NIPS*, 2004.

[104] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin. Context-based vision system for place and object recognition. In *IEEE Intl. Conference on Computer Vision (ICCV)*, Nice, France, October 2003.

[105] K. Toyama. Look, Ma - No Hands! Hands-Free cursor control with real-time 3D face tracking. In *PUI98*, 1998.

[106] D. Traum and J. Rickel. Embodied agents for multi-party dialogue in immersive virtual world. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2002)*, pages 766–773, July 2002.

[107] David Traum and Jeff Rickel. Embodied agents for multi-party dialogue in immersive virtual worlds. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 766–773, 2002.

[108] B. M. Velichkovsky and J. P. Hansen. New technological windows in mind: There is more in eyes and brains for human-computer interaction. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, 1996.

[109] Roel Vertegaal, Robert Slagter, Gerrit van der Veer, and Anton Nijholt. Eye gaze patterns in conversations: there is more to conversational agents than meets the eyes. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 301–308, 2001.

[110] P. Viola and M. Jones. Robust real-time face detection. In *ICCV*, page II: 747, 2001.

[111] S. Wang and D. Demirdjian. Inferring body pose from speech content. In *ICMI*, 2005.

[112] S. Wang, A. Quattoni, L. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *In CVPR*, 2006.

[113] Wikipedia. *Wikipedia encyclopedia.* http://en.wikipedia.org/wiki/Dialog_box.

[114] L. Wiskott, J.M. Fellous, N. Kruger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *PAMI*, 19(7):775–779, July 1997.

[115] C.R. Wren, A. Azarbayejani, T.J. Darrell, and A.P. Pentland. Pfinder: Real-time tracking of the human body. *PAMI*, 19(7):780–785, July 1997.

[116] X. Xie, R. Sudhakar, and H. Zhuang. A cascaded scheme for eye tracking and head movement compensation. *IEEE Transactions on Systems, Man and Cybernetics*, 28(4):487–490, July 1998.

[117] L. Young and D. Sheena. Survey of eye movement recording methods. *Behavior Research Methods and Instrumentation*, 7:397–429, 1975.

[118] S. Zhai, C. Morimoto, and S. Ihde. Manual and gaze input cascaded (magic) pointing. In *CHI99*, pages 246–253, 1999.