Real-time Continuous Gesture Recognition for Natural Multimodal Interaction

by

Ying Yin

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Author .....
Department of Electrical Engineering and Computer Science
May 19, 2014

Certified by .....
Randall Davis
Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Real-time Continuous Gesture Recognition for Natural Multimodal Interaction

by

Ying Yin

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

I have developed a real-time continuous gesture recognition system capable of dealing with two important problems that have previously been neglected: (a) smoothly handling two different kinds of gestures: those characterized by distinct paths and those characterized by distinct hand poses; and (b) determining how and when the system should respond to gestures. The novel approaches in this thesis include: a probabilistic recognition framework based on a flattened hierarchical hidden Markov model (HHMM) that unifies the recognition of path and pose gestures; and a method of using information from the hidden states in the HMM to identify different gesture phases (the pre-stroke, the nucleus and the post-stroke phases), allowing the system to respond appropriately to both gestures that require a discrete response and those needing a continuous response.

The system is extensible: new gestures can be added by recording 3-6 repetitions of the gesture; the system will train an HMM model for the gesture and integrate it into the existing HMM, in a process that takes only a few minutes. Our evaluation shows that even using only a small number of training examples (e.g. 6), the system can achieve an average F_1 score of 0.805 for two forms of gestures.

To evaluate the performance of my system I collected a new dataset (YANG dataset) that includes both path and pose gestures, offering a combination currently lacking in the community and providing the challenge of recognizing different types of gestures mixed together. I also developed a novel hybrid evaluation metric that is more relevant to real-time interaction with different gesture flows.

Thesis Supervisor: Randall Davis
Title: Professor

Acknowledgments

I would like to dedicate this work to my advisor, Prof. Randall Davis. Words cannot express my gratitude for all the trust, advice and learning opportunities he has provided for the past 6 years. His critiques helped me to be more precise in my research, and his questions guided me to think more carefully about my approaches. The comments and suggestions he gave me on this thesis are invaluable.

I would like to thank my thesis committee members Antonio Torralba and Bill Freeman. They have provided me helpful suggestions and insightful questions. They pushed me to work harder on my research, and answered my questions promptly.

I also want to express my gratitude to my friendly and supportive group-mates: Andrew Sabisch, Jeremy Scott, and Yale Song. They have given me tips and suggestions throughout the course of my study and research. More importantly, their companionship makes my time at lab enjoyable. I also want to thank Aaron Adler, Chih-yu Chao, Tom Ouyang, and Sonya Cates, who have graduated earlier from our group, but have given me useful advice when I was still new.

Thank you to Nira for all of her help and for making tedious things such as reimbursement, travel arrangement, and purchase much more easier. Thank you to Ron for helping me with many workshop related tasks.

I feel fortunate to have a group of supporting friends who make my life here so wonderful. Thank Dave McCoy for his strong support and belief in me. His drive and passion in creating something new has also touched and motivated me. Thank Shen Shen for bringing me laughter through her ingenuity.

Thank MIT for a wonderful 6-year experience. I will never forget this place and the brilliant people here.

Finally, I want to thank my parents for their strong and loving support. Even though they are thousands of miles away, their unwaving love is the fuel for my determination. I am deeply indebted to them for everything they taught me. Today, they are as happy as I am.

Contents

List of Figures

List of Tables

Gesture is a critical link running through the evolution of perception, conceptualization, and language.

David Armstrong, William Stokoe, and Sherman Wilcox,
Gesture and the nature of language

1

Introduction

Imagine how nice it would be, the next time you make a presentation, if you did not need to stand close to your laptop, or use a remote control with its limited functionality. What if you could present your work as naturally as having a conversation with your audience. You swipe your hand left and right to change slides. When you point to the slide with your hand, the display shows a cursor following wherever you point. When you are showing a video, you use a palm forward hand pose ("stop" gesture) to pause the movie, then move left and right to fast forward or rewind the video. You can also say "faster" or "slower" to change the video speed. When you need to jump to a particular slide, you make a circle gesture to show all the slides, and say "show this slide" while pointing at that slide. You can also make a dismiss gesture to pause the slide show (making the screen black) to take the distracting slides off the screen and get the full

attention of the audience.

This scenario shows an application of a multimodal interface to a real-world problem, with different categories of gestures playing an important part in the scenario. The system I developed makes the scenario real (Figure 1-1 and 1-2[1]). It provides real-time continuous gesture recognition and interaction, addressing problems that have previously been neglected, such as handling different types of gestures, and determining how and when the system should respond.

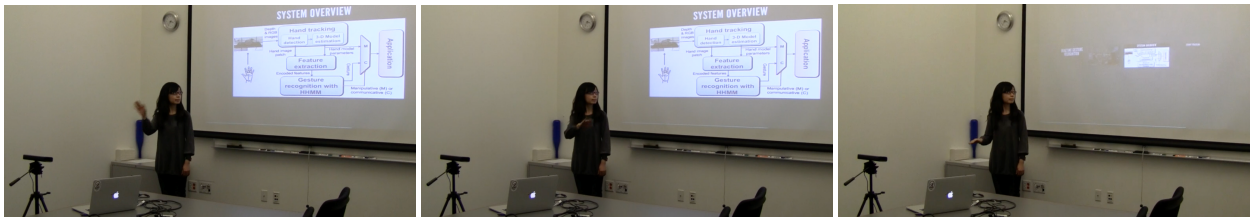


Figure 1-1: Real-time demonstration of the gesture controlled presentation application. The sequence shows using a circle gesture to bring up all the slides.

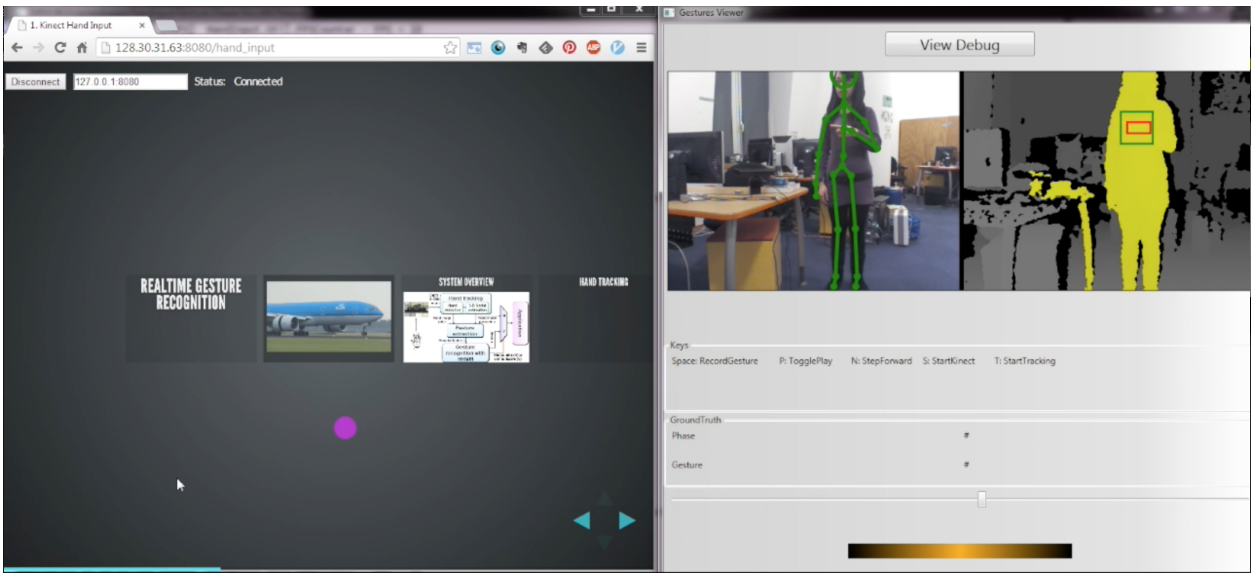


Figure 1-2: Two views of the gesture controlled presentation application. The left view is the browser based presentation framework. The purple circle indicates that the user is pointing at the slide. The right view is the debug interface showing the RGB image, the depth mapped image, the skeleton tracking, and the bounding box of the gesturing hand.

[1]A live demo video can be found at `https://www.youtube.com/watch?v=09BXfN2vk1E`

Recent trends in user interfaces have brought on a new wave of interaction techniques that depart from the traditional mouse and the keyboard, including multi-touch interfaces (e.g., the iPhone, the iPad and the Microsoft Surface) as well as camera-based systems (e.g., the Microsoft Kinect and the Leap Motion Controller). Most of these devices gained instant popularity among consumers, and the common trait among them is that they make interacting with computation more natural and effortless. Many of them allow users to use their hands and/or body gestures to directly manipulate virtual objects. This feels more natural because this is how we interact with our environment everyday.

There is also a trend in wearable human-computer interfaces (e.g., Google Glass, Samsung's Galaxy Gear smartwatches, Pebble smartwatches) that have potential for gesture input as well. Google Glass has a camera that can be used to recognize hand motion and hand shapes, while the accelerometers and gyroscopes in the smartwatches can be used to measure hand motion.

There is considerable potential and demand for natural interaction, and gesture is an important part of it. We are starting to see more gestural interfaces, but many of them still require that the hands function as a mouse with a limited number of other gestures. Our goal is to break this old paradigm of "point, click, drag" interaction. Our hands are much more versatile, and hence, offer the chance to design a gesture recognition system for natural human computer interaction (HCI) starting from the user interaction perspective. This means asking: what different types of gestures do people use; when should the system respond; how should the model be defined and trained; and how should we combine gesture and speech? These are the questions addressed in this thesis. Based on my findings, I developed a real-time continuous gesture recognition and interaction system that handles different types of gestures seamlessly, and responds to gestures appropriately.

1.1 Background

To design a natural gesture input interface, it is important to understand the nature of human gestures. This section gives some background on gesture production and taxonomy, and introduces several important concepts and terms central to the final system design.

1.1.1 Definition of Gestures

For a natural interface, it is important for the system to distinguish gestures from non-gestures (e.g., unconscious habitual movements like scratching one's head) because this will avoid restricting how people place or move their hands when they are not doing any purposive gestures.

Webster's Dictionary defines gestures as "... a movement usually of the body or limbs that expresses or emphasizes an idea, sentiment, or attitude." This definition is particularly related to the communicative aspect of the human hand and body movements. However, in HCI, the notion of gestures is somewhat different. In their review of the visual interpretation of hand gestures for HCI, Pavlović et al. [47] state that in a computer controlled environment one wants to use the human hand to perform tasks that mimic the natural use of the hand both as a manipulator and as used in human-machine communication. They describe this in part by having both *manipulative* and *communicative* gestures in their gesture taxonomy. We adopt this distinction.

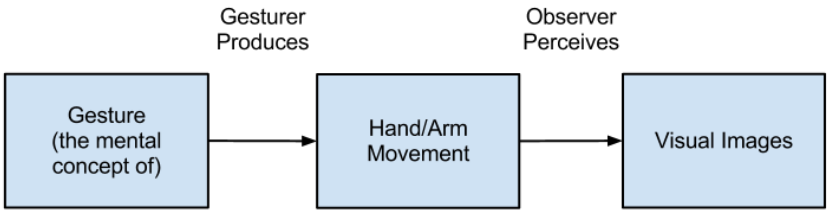


Figure 1-3: Production and perception of gestures. Hand gestures originate as a mental concept, are expressed through arm and hand motion, and are perceived as visual images [47].

Pavlović et al. [47] also gives a model (Figure. 1-3) for the production and perception of gestures, based on the model used in the field of spoken language recognition. According to their model, gestures originate as a gesturer's mental concept, possibly in conjunction with speech. They are expressed through the motion of arms and hands, while observers perceive gestures as streams of visual images that they interpret using their knowledge about those gestures. In HCI, the observer is the computer and the knowledge it possesses is the training data.

1.1.2 Gesture Taxonomy for Natural Interaction

Several gesture taxonomies have been suggested in the literature. Some of them deal with psychological aspects of gestures [29, 39], while others are inspired by an HCI perspective [47, 50, 74]. The taxonomy that seems most appropriate for natural HCI and human-centric design is developed by Wobbrock et al. [74]. Their study was based on eliciting natural behavior from non-technical users when interacting with a computing system. As their study focused on tabletop gestures, I further generalized their taxonomy to encompass interaction for both vertical and horizontal interfaces.

Wobbrock et al. [74] classified gestures along four orthogonal *dimensions*: *form*, *flow*, *binding*, and *nature*. Within each dimension, there are multiple *categories*, shown in Figure 1-4. The *form* dimension is particularly relevant for gesture recognition because it concerns the visual characteristics of gestures. The *flow* and the *binding* dimensions are relevant for the application layer because they are related to how the user interface (UI) should respond to gesture events. The *nature* dimension is very similar to the taxonomies mentioned in other related work, and hence, will be explained further below. However, the *form* and the *flow* dimensions are the focus of this thesis. Since these four dimensions have specific meanings in this taxonomy, I will refer them in italic text in the thesis to make the distinction.

Wobbrock's Nature Dimension

Along the *nature* dimension, Wobbrock et al. divide gestures into four categories: symbolic, physical, metaphorical and abstract. These categories have some overlap with Pavlović's classification, but Pavlović's is more comprehensive.

The hierarchical categorization along the *nature* dimension in Figure 1-4 is the one I summarized based on Pavlović's taxonomy. Gestures are divided into manipulative and communicative. Manipulative gestures are used to act on objects (e.g. moving an virtual object around, click a button), while communicative gestures have an inherent purpose for communication [47].

People perform communicative gestures via acts or symbols. Gestures via acts are those directly related to the interpretation of the movement itself. Such movements are classified

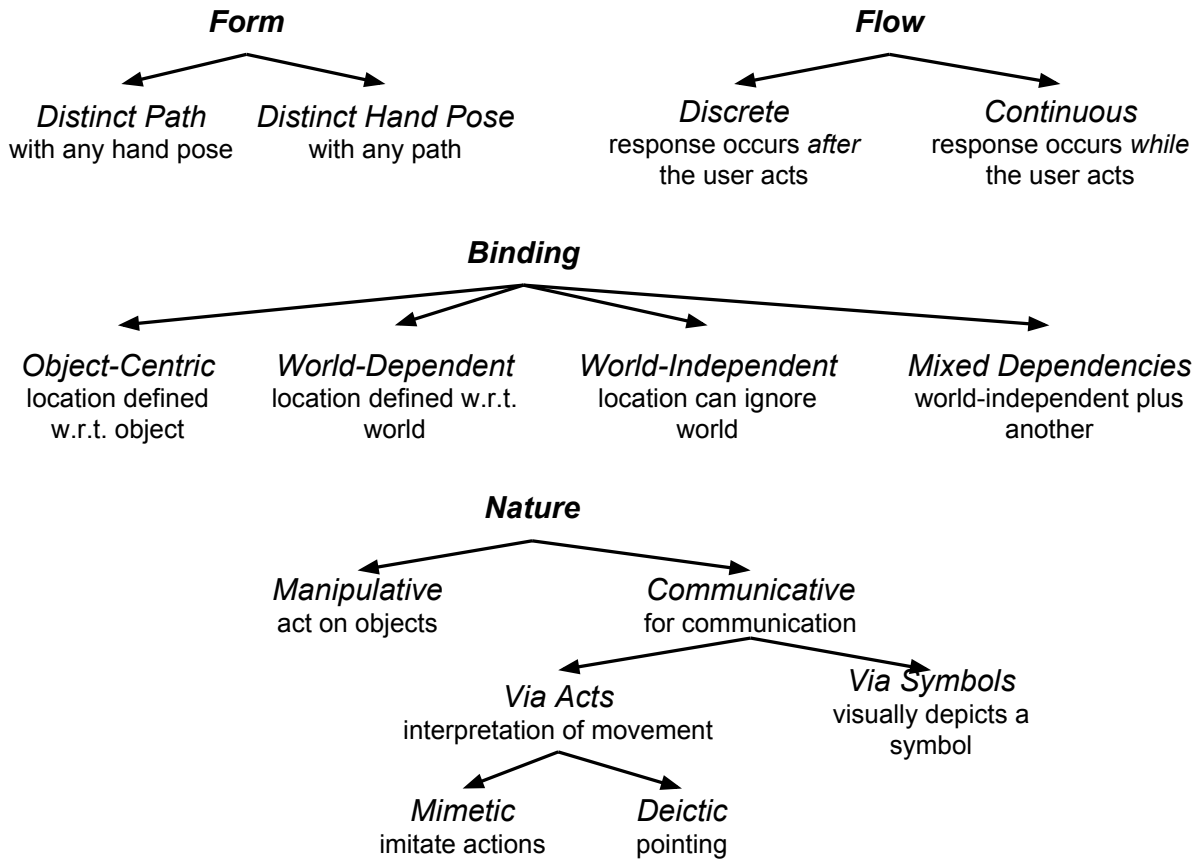


Figure 1-4: Gesture taxonomy along four dimensions. The abbreviation "w.r.t" means "with respect to."

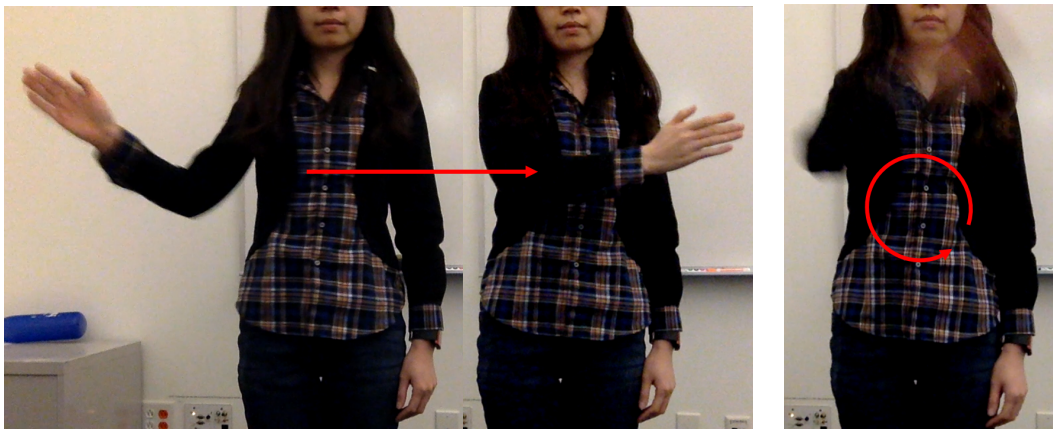
as either mimetic (which imitate some actions) or deictic (pointing acts that select objects by location). Gestures via symbols are those that have a linguistic role, and are often represented by different static hand postures. An example is forming the O.K. pose for "accept".

Form and Flow Dimensions

Although the classification in the *nature* dimension gives us a good understanding of gestures, it is less useful for designing the gesture recognition system than the *form* and the *flow* dimensions are.

I distinguish two categories in the *form* dimension: *path* and *pose*. The path category contains gestures characterized by distinct paths without any distinct hand poses. For example, a Swipe Left gesture is characterized by a right to left motion, while a Circle

gesture is characterized by a circular motion of the hand (Figure 1-5). In doing these, users typically hold their hands in some natural, relaxed, but unpredictable pose.



(a) Swipe Left (b) Circle

Figure 1-5: Examples of path gestures.

Pose gestures are characterized by distinct hand poses without any distinct paths. This category of gestures is usually associated with direct manipulation of some virtual objects on the interface. For example, a user may use a Point hand pose and move around to point at different things on a display.

In the *flow* dimension, a gesture's flow is *discrete* if the gesture is performed, delimited, recognized, and responded to as an atomic event [74]. For example, if the Wave gesture is regarded as a discrete flow gesture, the system should respond once, at the last repetition of the left-right motion. Flow is *continuous* if ongoing recognition is required and the system should respond frame by frame, as for example during a "point" gesture, where we want to show the cursor on the screen continuously moving according to the hand position.

The *form* dimension informs us what different features we need to consider to differentiate gestures. The *flow* dimension informs us how the system should respond to gesture events. As a result, I focus on these two dimensions in this thesis.

While for some gestures, their categorization along certain dimension is obvious; for others, it is not, and the actual categorization can be decided by the application developer or the user. Figure 1-6 shows an example of gesture categorization along the two dimensions.

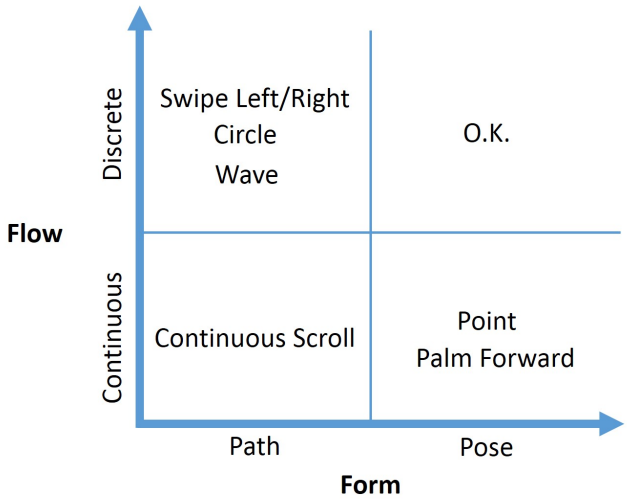


Figure 1-6: A categorization of gestures along the *flow* and the *form* dimensions.

1.1.3 Temporal Modeling of Gestures

Making gesture interaction feel natural requires a system that responds at the correct moment, meaning that we have to consider the temporal characteristics of a gesture. We set a foundation for doing this by taking account of the three *phases* that make up a gesture:

- pre-stroke,
- nucleus (peak [39]), and
- post-stroke [47].

Pre-strokes and post-strokes are movement from and to the rest position. The nucleus of a gesture, as Kendon [29] observes, has some "definite form and enhanced dynamic qualities". Every gesture must have a nucleus, which is the content-carrying part of the gesture. Based on this theory, the lack of a nucleus phase can be used to distinguish unintentional movements from gestures .

Even though the end of the post-stroke phase can be more easily detected by finding the start of the rest position, I want to do more than this. Since the nucleus phase is the meaningful part of the gesture, for a discrete *flow* gesture, the system should respond immediately at the end of the nucleus, instead of at the end of the post-stroke. To make the system more responsive, I address this challenging problem of distinguishing the start

and end of the nucleus phase from the pre-stroke and post-stroke phases. This also allows the system to respond to continuous *flow* gestures immediately at the start of the nucleus phase.

1.2 System Overview and Thesis Outline

The gesture interaction system consists of four modules: hand tracking, feature extraction, gesture recognition, and application user interface (Figure 1-7). At each time frame, the hand tracking module takes raw data from the sensor and estimates the location of the gesturing hand. The feature extraction module computes feature descriptors from the localized hand and sends the encoded features to the gesture recognition module. The gesture recognition module estimates the current most likely gesture label and gesture phase information based on the input stream of feature vectors. The gesture information, together with smoothed hand position information are sent to the application level.

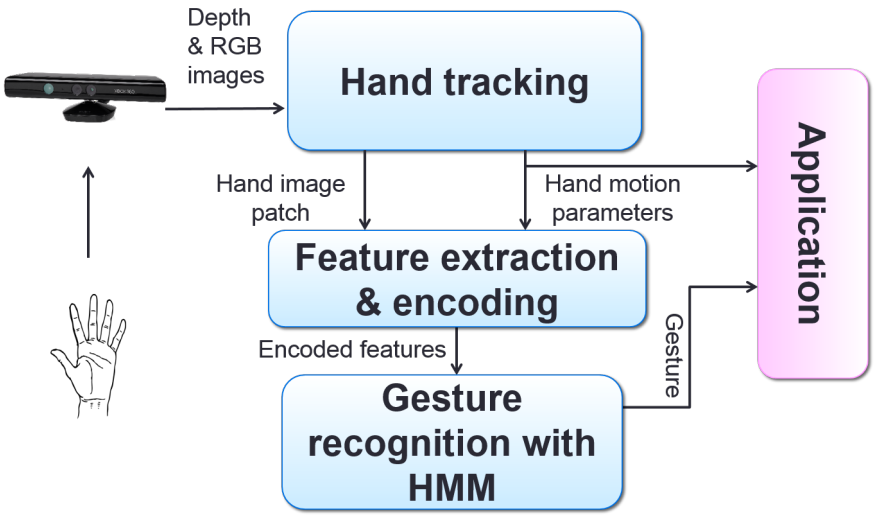


Figure 1-7: System overview.

In each module, I have developed new techniques to improve the gesture recognition accuracy and user experience, and improved upon existing methods. The main focus and contributions of this work are in gesture recognition.

1.3 Contributions

The main contributions of this work include:

- Hand tracking
  - I improved hand tracking by replying in part on gesture salience: I define gesture salience to be proportional to the amount of motion and the closeness of the motion to the observer. Based on this, I compute a probability map for the gesturing hand locations in a frame. Compared with using the hand joint position from the Kinect SDK, using our hand tracking method gives a 2.7% absolute increase in the recognition F_1 score.
- Feature extraction
  - I use histogram of oriented gradients (HOG) as a hand shape descriptor and apply principal component analysis (PCA) to reduce its dimensionality. I then use it as part of the feature vector input to the hidden Markov models (HMMs) based recognition module. This novel approach allows the system to handle path and pose gestures in a unified way.
- Gesture recognition
  - I developed a probabilistic framework based on HMMs for real-time continuous (i.e., unsegmented) gesture recognition that unifies recognition of two *forms* of gestures (path and pose). I use different HMM topologies for different forms of gestures and combine them into one flattened hierarchical HMM for simultaneous recognition and segmentation. With user dependent training and testing, the system achieves an average F_1 score of 0.805 on the YANG dataset.
  - I used embedded training and hidden state information to detect different gesture phases – the pre-stroke, the nucleus, and the post-stroke phases – allowing the system to respond more appropriately and promptly to gestures that require a discrete response and those needing a continuous response. With user independent

training and testing on the ChAirGest dataset, this method achieves a temporal segmentation rate of 0.923 for identifying the start and the end of nucleus phases.

  - I collected a new dataset (YANG dataset) that includes 4 path and discrete flow gestures and 3 pose and continuous flow gestures from 10 users, a combination currently lacking in the community, to evaluate system performance.
  - I developed a hybrid evaluation metric that is more relevant to real-time interaction with different gesture *flows*.
  - I used gesture phase information to do gesture spotting, filtering out non-gestures with no nucleus phases.

- User interaction techniques
  - I identified two main ways of combining gesture and speech for natural interaction: using gestures to augment speech, and using speech to augment gestures, and demonstrated the combination in an interactive system.

2

Related Work

A gesture input pipeline can be broken down into several sequential modules: sensors, hand tracking, feature extraction, gesture recognition, and user interfaces. I discuss related work in each module.

2.1 Sensors

The first step in the pipeline is having sensors capture hand movements and configurations. Early attempts to solve this problem resulted in mechanical devices that measure hand joint angles and spatial positions directly. This group is best represented by the glove-based approaches using devices such as CyberGloves [20] and Powergloves [28]. However, wearing gloves makes gesturing more cumbersome, leading to many efforts to make the gloves more light-weight (e.g., by using Bluetooth wireless data transmission as in the CyberGove II). To

further reduce the bulkiness of the gloves, people have used colored markers on the fingers [40] or colored gloves with no electronics [70] and using RGB cameras and computer vision techniques to interpret gestures. However, by requiring the user to wear something extra hinders the acceptance of such devices as "everyday" natural interaction interfaces.

The most non-obtrusive way to capture the hand is bare-hand tracking. Shin et al. [56] use stereo RGB cameras to extract the hand from background based on skin colors. One limitation of RGB cameras is that they are very sensitive to lighting condition. This prompted researchers to look into other types of cameras. Oka et al. [45] use thermal imaging for hand segmentation under complex background and changing light, relying on the condition that a hand's temperature is almost always distinct from the background. Larson et al. [35] improved on this method by adding touch detection. They detect finger contacts by using heat transferred from a user's hand to the surface for touch-based gestures. However, in order to detect the heat trace, users have to drag their fingers a bit instead of just touching. This may be a small departure from what users would expect as "natural" based on their experience in the physical world.

Thermal imaging measures radiation emitted by objects in the far-infrared (F-IR) spectrum. There are other well-known "IR-imaging" techniques used in the HCI community which use devices operating in the near-infrared (N-IR) spectrum. N-IR is employed in some fairly recent interactive tabletop surfaces and depth cameras [27, 81]. A number of projects have used IR for tabletop interaction by detecting multi-touch gestures using an under-surface mounted camera and an illumination source, e.g., Microsoft's Surface. Recently, multi-touch phones and tablets have become more and more ubiquitous. These devices are based on capacitive touch sensitive screens. Touch-based devices are becoming more sophisticated, but are still limited to gestures in 2D space with one or multiple fingers.

Depth sensing input devices such as the Kinect sensor and the Leap Motion Controller open up the possibility of gesturing in 3D space. The Kinect sensor contains a depth sensor, a color camera, and a four-microphone array that provide full-body 3D skeleton tracking, face recognition, and voice recognition capabilities [81]. The first generation of the sensor uses structured light for depth sensing. The color stream has a resolution of 640×480px at 30Hz or 1280×960px at 12Hz. The depth sensing video stream has a resolution of 640×480px

at 30Hz[1]. With the default range, the Kinect sensor (version one) has a depth sensing range limit of about 0.8m - 4m[2]. The random error of its depth measurements increases quadratically with increasing distance from the sensor, and ranges from a few millimeters at 0.5m distance to about 4cm at the maximum range of 5m [31]. It can track 20 body joints including hands. The newer version of Kinect uses a time-of-flight camera for depth sensing, and a higher resolution color camera (1920×1080px at 30Hz)[3]. It can track 25 body joints (including thumbs)[4].

In stead of full-body tracking, the Leap Motion Controller has a smaller observation area and specializes in hand tracking with higher resolution. It uses two monochromatic IR cameras and three infrared LEDs, and observes a roughly hemispherical area, to a distance of about 1m [5]. It can track all 10 fingers up to 1/100th of a millimeter[5]. However, as the algorithm is optimized for detecting finger-shaped objects, detecting other hand shapes such as "palm up" (with finger closed) or "fist" can be challenging with the Leap Motion Controller.

More recently, a new generation of smart watches is leading the way in wearable computing interfaces. These smart watches are equipped with motion sensors such as accelerometers, gyroscopes and magnetometers, which are basic components of an inertial measurement unit (IMU). These sensors can give relatively accurate motion and orientation information of users' hands at a high frame rate (e.g., the Xsens MTw IMU has a frame rate of 50Hz [54]), and can thus be used for gesture input. While data from camera-based sensors is prone to occlusion, and its quality of accuracy is highly dependent on the position of the user relative to the sensor, IMU sensors are occlusion free and position independent. In addition, data from IMUs can be used with less complex processing compared with data from camera-based sensors [54].The disadvantage of an IMU is that it cannot capture hand shape information.

There is a plethora of new sensors, and they have both pros and cons for hand tracking and gesture recognition. It is possible to combine different sensors so that they can complement each other. Sensor technology will continue to advance, and as a result it is important to

[1] http://msdn.microsoft.com/en-us/library/jj131033.aspx
[2] http://msdn.microsoft.com/en-us/library/hh973078.aspx
[3] http://www.develop-online.net/news/next-xbox-leak-reveals-kinect-2-specs/0114096
[4] http://en.wikipedia.org/wiki/Kinect#Kinect_for_Xbox_One
[5] https://www.leapmotion.com/product

make gesture recognition system flexible and generalizable to different feature input.

I classify sensors into two categories according to their placement: environmental and wearable. An environmental sensor is installed at a fixed position (e.g., the Kinect senor and the Leap Motion Controller); while a wearable sensor is worn by the user (e.g., smart watches). I evaluated my system with sensors from both of these categories: a Kinect sensor and an IMU sensor. The Kinect sensor's microphone array is useful for multimodal interaction, while its depth sensor and color camera can be used for bare hand tracking and hand pose recognition, integral parts of my system. With the IMU, I demonstrate that the framework is generalizable and the two sensors can be used together.

2.2 Hand Tracking

The next step in the pipeline is tracking the hand(s), i.e., localizing and segmenting hands from the rest of the input. This step is substantive for camera based sensors, but trivial for sensors worn on hands or wrists.

Common hand tracking methods are based on either skin detection [56] or motion detection [15]. Skin detection in the HSV [12] or the YCrCb color space can be relatively robust and less sensitive to lighting conditions, compared to the RGB color space. However materials with skin-like colors (such as clothes) can produce false positives. Skin from other parts of the body can also interfere with hand tracking. Shin et at. [56] filter out false positives by considering the skin blob closest to the camera. However this can fail when the hand is close to the body, generally resulting in the face being detected instead. Methods based on motion detection would assume the background is relatively static and there is no motion from other parts of the body.

Marcos-Ramiro et al. [38] developed a method of computing hand likelihood maps based on RGB videos. They mention that, given a frontal, static camera pointing to the gesturer, hands are usually the closest part to the camera, and also move with the highest speed. These characteristics translate to more movement in the image where the gesturing hand(s) is. As they only used non-stereo RGB images, they could not compute the closeness of the movement.

Hand tracking can be considered as a feature localization/detection problem. Feature detectors usually select spatio-temporal locations and scales in video by maximizing specific salience functions [68]. One example is space-time interest points (STIPs) detector introduced by Laptev [33]. He used it for human action recognition from movies [34]. STIPs are points in the video sequence where the image values have significant local variations in both space and time. His method extends the Harris interest point detector in the spatial domain [25] to the spatial-time domain. For each cuboid of interest point, both histograms of oriented gradients (HOG) and histograms of optic flow (HOF) are computed as feature descriptors. Although promising results were demonstrated using STIPS, Wang et al. [68] find that the simple dense sampling method (i.e., computing feature descriptors at regular positions and scales in space and time) outperforms STIPs and other interest point detectors for action recognition in realistic video settings. They suggest that this indicates the limitations of current interest point detectors.

Another approach to hand tracking is based on 3D body pose estimation and searching for the hand region near the wrist [59]. Skeleton tracking provided by the Kinect Software Development Kit (SDK) gives a relatively robust 3D body pose estimation. The tracking is based on a body part detector trained from a random forest of a huge number of synthetically-generated body poses [58]. One of its major strengths is that it does not require an initialization pose. However, the hand joint tracking from the Kinect SDK fails when the hands are close to the body or are moving fast, especially in the seated mode [78].

With a coarse initialization of a hand's configuration, Sudderth et al. [64] use a graphical model of hand kinematics and nonparametric belief propagation (NBP) to refine the tracking. However, the process is relatively slow (1 min per NBP iteration o a Pentium IV workstation and the process requires multiple iterations) and cannot be used in real-time yet.

My hand tracking method is salience based and is similar to Marcos-Ramiro et al.'s, but I combine both RGB images and depth images to compute gesture salience. I use depth data to compute the amount of motion, which is computationally less expensive than the optical flow method they used. By combining skin, motion and closeness together to compute a probability map for the gesturing hand location, my method better filters out false positive regions with skin-like colors and makes fewer restrictions on motion. I compared my method

with the dense sampling method using the same dataset and recognition method, and my method gives 11.6% absolute improvement in frame-based classification F_1 score. Compared with the hand joint tracking from Kinect SDK, my method is also shown to be more robust when the user is seated and the hands are close to the body or are moving fast.

2.3 Feature Extraction

The extraction of low level image features depends on the hand model in use, and the complexity of the hand model is, in turn, application dependent. For some applications, a very coarse and simple model may be sufficient. The simplest model is treating the hand as a blob and tracking only the 2D/3D location of the blob. For example, Sharma et al. [55] use 2D position and time difference parameters to track the hands as blobs, which is sufficient for them to distinguish whether the hands are doing a point, a contour, or a circle gesture. The PrimeSense NITE middleware for OpenNI's natural interaction framework also tracks the hand as a point. It requires the user to do a "focus" gesture ("click" or "wave") in order to start hand tracking [49]. The gestures it supports are "click" (pushing hand forward), "wave", "swipe left", "swipe right", and "raise hand".

However, to make a more generalizable system for natural interaction, a more sophisticated model is required. One step forward is adding fingertip locations in the hand model as in [45, 26, 35]. Tracking fingertips is usually sufficient for manipulating objects on the 2D surface. To support a richer set of gestures involving 3D manipulation, we may need a more sophisticated model. For instance, Wang et al. [70] uses a 26 DOF 3D skeletal model in their real-time hand-tracking system with a color glove. Oikonomidis et al. [44] also used a parametric 3D model with 26 DOF and 27 parameters. They treat hand pose estimation from markerless visual observation as an optimization problem and achieved a 15Hz frame rate with an average error of 5mm (distance between corresponding finger joints in the ground truth and the in the estimated model) .

Another approach is using an appearance-based model. This means that the model parameters are not directly derived from the 3D spatial description of the hand. The hand poses are modeled by relating the appearance of any pose to the appearance of the set of

predefined, template poses [47]. In their markless hand-tracking system, Wang et al. [69] use efficient queries of a database of gestures and desktop-specific hand silhouette samples for pinch/click gesture detection.

In this work, I use a simplified 3D skeletal hand model for horizontal display interaction where hands are close to the display surface and can directly manipulate virtual objects. For communicative gestures, we need to know just the meaning of the gesture and do not require exact spatial parameters. Hence appearance-based models is more suitable which also require less computation, allowing us to achieve a real-time frame rate of 30Hz.

2.4 Gesture Recognition

Many previous efforts on gesture recognition have focused on a single category of gestures (either path gestures or pose gestures), though here are some work that addressed the problem of handling multiple gesture categories in one system.

2.4.1 Pose Gestures

One group of prior work focuses on classifying a set of predefined static hand poses frame by frame. Freeman and Roth [23] use histogram of local orientations, a precursor of HOG [16], for hand pose recognition. Recognition is based on selecting the feature vector in the training set that is closest to the test feature vector. Suryanarayan et al. [65] use a volumetric shape descriptor computed from depth data as the feature vector, and use Support Vector Machine (SVM) for classification. I use HOG as a hand pose feature descriptor but incorporate it in a unified HMM-based framework for both pose and path gestures.

2.4.2 Path Gestures

Another group of prior work focus on recognizing only path gestures. Most of these efforts used a hidden Markov model (HMM) and its variants to recognize such gestures [63, 55].

Starner and Pentland used HMMs to recognize the part of American Sign Language that uses path gestures [63]. They collected about 500 sentences of a specific grammar ("personal

pronoun, verb, noun, adjective, (the same) personal pronoun") with a total lexicon of forty words. No intentional pauses were placed between signs within a sentence, but the sentences themselves were distinct. Because finger spelling was excluded and there were few ambiguities in the vocabulary based on individual finger motion, each gesture word would have a distinct path. The feature vector they used includes 8 elements: each hand's x and y position, angle of axis of least inertia, and eccentricity of bounding ellipse. They use Gaussian distributions to model the emission probabilities. When training the HMMs, they used Viterbi training (see Appendix D for more details on HMMs) to estimate the initial means and variances of the output probabilities (after initially dividing the evidence equally among the words in the sentence). The initial estimates are fed into a Baum-Welch re-estimator, whose estimates are refined in embedded training. The techniques they use are very similar to the ones used in speech recognition (not surprisingly given the close parallel between sign language and speech). In fact, they were able to use Entropic's Hidden Markov Model TookKit (HTK)[6] directly for all the modeling and training tasks.

Gestures for HCI usually are less structured than sign language and do not follow a grammar, hence to learn context-dependent transition and emission parameters will require $O(N^2)$ training examples to cover all possible pairs of gestures for N gestures. But our user study indicate that people prefer to give around 5 examples per gesture (i.e. linear growth with the number of gestures). To prevent quadratic growth of required training examples, I do not embed gesture HMMs in a sentence to learn context-dependent models for inter-gesture transition and assume the transitions between gestures have equal probability. However, I do embed gesture *phase* HMMs in a full gesture sequence to identify different gesture phases which is necessary for real-time interaction. In this way, we learn context-dependent models within a gesture, i.e., different gestures may have different pre-stroke and post-stroke phases depending on their starting and ending points.

Sharma et al. [55] considered natural gestures that do have grammar constraints. They examined hand gestures made in a natural domain, weather narration, and identified three deictic gestures: *point*, *area*, and *contour*. They also considered the pre-stroke and post-stroke gesture phases, and used 5 states for each of the pre-stroke, post-stroke and point

[6] `http://htk.eng.cam.ac.uk/`

HMMs and 6 states for each of the contour and rest HMMs. The feature vector they use is motion based, and includes relative distance between the hand and the face, angle of the arm, angular velocity, radial velocity and tangential velocity. Hand poses are not considered. Without considering speech input, they obtained 69.52% recognition accuarcy for the three gestures. Although the deictic gesture is an important category in HCI, it is still too limited to support a broader range of applications.

More recently, discriminative models such as conditional random fields (CRF) and its variants, such as hidden CRF [71] and latent dynamic CRF (LDCRF) [41], have been applied to gesture recognition with improved recognition results. Morency et al. [41] formulated the LDCRF model that is able to perform sequence labeling and segmentation simultaneously. Song et al. [61] use LDCRF to distinguish path gestures where certain gestures share the same paths but different hand poses. They use the HOG feature descriptor for hand poses and use SVM to classify a pre-determined set of hand poses. The result of the classification is used as part of the final feature vector. They considered hand poses for gestures with distinct paths, but did not handle arbitrary movement, as I am doing.

As discriminative classifiers model the posterior $p(y|x)$ directly, or learn a direct map from inputs x to the class labels y, it is generally believed that discriminative classifiers are preferred to generative ones (such as HMMs) for classification problems. However, one limitation of CRF-based models is that training for these models is much more expensive computationally and converges much slower than those of HMMs [32] because of the larger parameter space to search. Discriminative models can also require more training data to reach lower asymptotic error rates [43]. I applied LDCRF for gesture classification and gesture phase segmentation. Compared with my concatenated HMM-based method, LDCRF gives better gesture phase segmentation result, but lower gesture classification result. The LDCRF model also takes much longer time to train (18hr on the ChAirGest dataset) compared to the HMM-based method (which takes 7min). As my user study shows that people prefer to use a few examples to define their own gestures when necessary (see Section 3.2.4), I decided to use the HMM-based model, which is fast to train and requires fewer training examples.

2.4.3 Multiple Categories of Gestures

Some work has addressed the problem of handling multiple gesture categories in one system. Keskin et al. [30] propose a unified framework to allow concurrent usage of hand gestures, shapes and hand skeletons. Hand gestures are modeled with mixture of HMMs using spectral clustering. Hand shape classification and hand skeleton estimation are based on classifying depth image pixels using randomized decision forests. Hand gesture classification is active all the time. The framework estimates a set of posteriors for the hand shape label at each frame, and continuously uses these posteriors and the velocity vector as observations to spot and classify known gestures. They distinguish gestures with pure motion and pure hand shape by thresholding the magnitude of the velocity vector. However, they did not mention handling gestures that combine distinct hand poses with arbitrary movement. For this category of gesture, it will be hard to manually set a velocity threshold to distinguish them from gestures with distinct paths.

Oka et al. [45] developed a system that allows both direct manipulation and symbolic gestures. These two categories are in the *nature* dimension. Based on the tracking result, the system first differentiates the gesture as either manipulative or symbolic according to the extension of the thumb. They define gestures with an extended thumb as direct manipulation and those with a bent thumb as symbolic gestures. For direct manipulation, the system selects operating modes such as rotate, move or re-size based on the fingertips configuration; for symbolic gestures, it uses HMMs for classification. The use of thumb to distinguish manipulative and communicative gestures seems arbitrary and unnatural. My system does not require an arbitrary hand pose to indicate gesture categories. It handles the path and pose gestures seamlessly under one recognition framework. I believe that this can help to reduce users' cognitive load for using the interface, and make the interaction more natural.

2.4.4 Gesture Spotting

A common approach to distinguish non-gestures from gestures is to use one or two non-gesture HMMs to provide likelihood thresholds for outlier rejection [75]. Peng et al. [48] argue that using one or two HMMs cannot effectively reject non-gesture outliers that resemble

portions of gestures. In addition to a general non-gesture model, they train several non-gesture HMMs by automatically identifying and manually specifying non-gesture models from the training data. This approach is suitable when the set of the input data is limited, but in real life the set of possible non-gestures is limitless and it is not clear how this approach will scale.

I use gesture phases to distinguish gestures from non-gestures. As explained in Section 1.1.3, every gesture must have a nucleus phase. My system identifies different gesture phases, and any hand movement that does not contain a nucleus phase will be treated as a non-gesture. I combine this method with a thresholding method to improve the overall performance of gesture spotting.

2.4.5 Online Recognition

Another group of work focus on online gesture recognition which is important for real-time interactive systems. Song et al. [61] extend LDCRF with multi-layered filtering and a temporal sliding window to perform online sequence labeling and segmentation simultaneously. Their approach incurs one to four seconds delay in their experiments. For real-time interaction, 0.1s is about the limit for having the user feel that the system is reacting instantaneously. We may relax this response time a bit, but 1.0s is about the limit for the user's flow of thought to stay uninterrupted [13]. Song et al. focus only on communicative gestures and assume no non-gestures in the input data.

Françoise et al. [22] use hierarchical HMMs to model musical gestures using motion data from the Wii remote controller, and use fixed-lag smoothing for real-time recognition and segmentation.

My method is similar to Françoise et al.'s, but I consider path and pose gestures in a single framework. My system incurs a 0.3s delay, which is shorter than that of Song et al.'s method.

2.4.6 Commercial Systems

There are several commercially available gesture recognition systems that are worth noting. The gesture interaction provided by the Kinect-based games is one of the popular ones. Based on the observation of the interaction available in Kinect games, it seems that the system is looking for only one gesture (wave) or body pose (the exit pose[7]) at a time, and the rest of the time, it is just tracking the hand and the body, and turns the hand into a mouse.

Many commercial gesture recognition systems use if-then (hand-coded) rules based on heuristics. For example, the Leap Motion plugin[8] for the Reveal.js[9] HTML5 presentation framework uses the number of fingers detected and the changes in the x and y coordinates between consecutive frames to detect swipe and point gestures. While if-then rules could be easy to define for a small number of simple gestures, they may be hard to define for more complex gestures [4]. For example, it may be hard to define a circle gesture using if-then rules because a sequence of several coordinate locations (instead of just two) are needed, and the rules can conflict with other rules (e.g., the rules for swipe left/right gestures). My system uses a probabilistic model and learns the gesture parameters automatically based on observations (training data), which provides a more general and scalable way to define gestures.

2.5 Multimodal User Interfaces

Bolt's pioneering work in the "Put That There" system [10] demonstrated the potential for voice and gestural interaction. In that system, the hand position and orientation were tracked by the Polhemus tracker, i.e., the hand was essentially transformed to a point on the screen. The actual hand posture did not matter, even if it was not in a pointing shape. The speech also followed a rigid and limited command-like grammar. Even though this is early work, it provides some insight about the advantages of multi-modal interaction. As Bolt summarized in the paper, using pointing gesture allows the use of pronouns in the speech,

[7] `http://support.xbox.com/en-US/xbox-360/kinect/how-to-use-the-kinect-hub-and-guide`
[8] `https://github.com/hakimel/reveal.js/blob/master/plugin/leap/leap.js`
[9] `http://lab.hakim.se/reveal-js`

with the corresponding gain in naturalness and economy of expression [10].

Since then, several multi-modal interaction prototypes have been developed that moved beyond Bolt's "Put That There" system. Cohen et al. [14] developed the QuickSet prototype, a collaborative, multi-modal system running on a hand-held PC using pen and voice as input. They used a multi-modal integration strategy that allows speech and pen gesture to compensate for each other, yielding a more robust system. Rauschert et al. [53] developed a system called Dialogue-Assisted Visual Environment for Geoinformation (DAVE_G) that uses free hand gestures and speech as input. They recognized that gestures are more useful for expressing spatial relations and locations. Gestures in DAVE_G include pointing, indicating an area and outlining contours. Speech and gesture are fused for commands that need spatial information provided by the gesture.

In this thesis, I identify two ways of combining speech and gestures that are particularly effective for HCI. In addition to using deictic gestures to provide spatial information as a complement to speech as in [53], I also use speech as a complement to manipulative gestures. Based on the user study [77] I conducted, I observe that manipulative gestures are at times accompanied by adjectives and adverbs that refine the actions. I demonstrate these two combinations in a gesture controlled presentation framework I developed.

3

Datasets

This chapter describes the datasets used to perform the experiments in this dissertation.

3.1 Related Work

Many public datasets for evaluating gesture recognition contain only one category of gesture [37, 54, 60]. One dataset that contains different categories of gestures is the Chalearn Gesture Dataset (CGD 2011) [24]. It contains both static postures and dynamic gestures. In this dataset, a static posture is one in which a single posture is held at the same position for a certain duration. If we consider the pre-stroke and the post-stroke phases as well, the static posture also has a distinct path, and hence, they could be handled by the same method as the dynamic gestures. This dataset does not contain gestures with distinct hand poses but arbitrary movement.

3.2 YANG Dataset

As we have been unable to find gesture datasets that include both gestures with distinct paths and gestures with distinct hand poses. To evaluate our method, I collected a new dataset named YANG (Yet Another Natural Gesture dataset) which has a vocabulary of 7 one-hand/arm gestures including both path and posture gestures. They are chosen for possible use in gesture controlled presentation and also to span over different potential difficulties (see the comments in Table 3.1). Figure shows an example of each gesture.

#	Name of gesture	Form	Comment
1	Swipe Left	distinct path	simple path
2	Swipe Right	distinct path	simple path
3	Circle	distinct path	complex path
4	Horizontal Wave	distinct path	has arbitrary repetitions
5	Point	distinct hand pose	arbitrary path
6	Palm Up	distinct hand pose	arbitrary path
7	Grab	distinct hand pose	arbitrary path

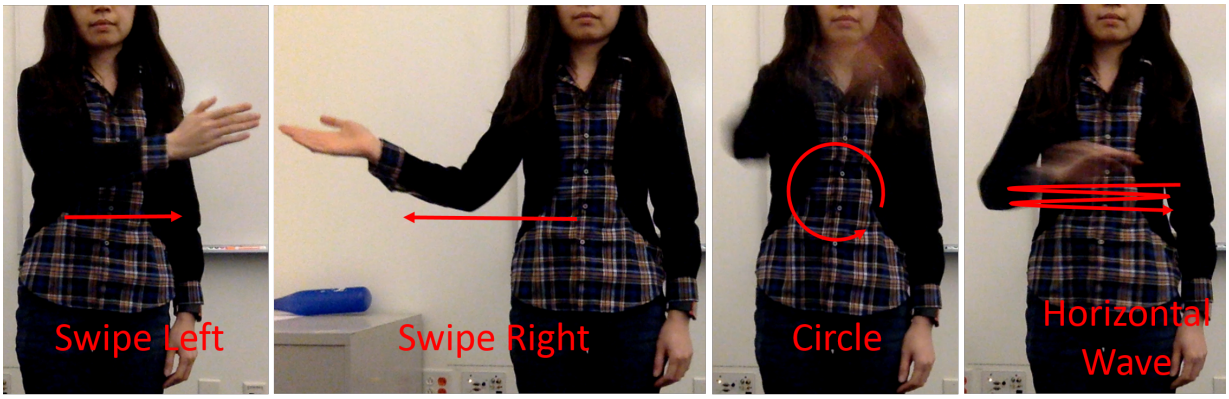
Table 3.1: YANG dataset gesture vocabulary.

This dataset presents various features of interest. Table 3.2 lists both the challenging and easy aspects of the dataset.

3.2.1 Recording Setup and Procedure

The dataset contains data from 10 participants each performing the gestures in 4 sessions. All the participants are university students. The participants were shown a video demonstration of each gesture[1] at the beginning. In each session, the participant stands at about 1.5m from the Kinect for Windows sensor (version one), and performs each gesture 3 times according to the text prompts on a screen indicating the name of the gesture to perform. The order of the gestures is random and the time between each gesture is random (between 2s and 6s). The first 2 sessions have "Rest" prompts between each gesture, telling participants to go to the rest position (hands relaxing at the side of the body), and the second 2 sessions do not have "Rest" prompts so participants can choose to rest or not between consecutive

[1]Video demonstration of the gestures: `https://www.youtube.com/watch?v=VDDX0TkenTY`



(a) Path gestures

(b) Pose gestures with arbitrary movement

Figure 3-1: YANG dataset gesture vocabulary.

gestures. This too distinguishes the dataset from previous ones [54, 24] where gestures are always delimited by rest positions.

Unlike Ruffieux et al. [54], we do not show video demonstration every time the participants perform a gesture because we want a realistic scenario. In real practice, it is unlikely that a user will follow a video demonstration every time he/she does a gesture. The result is that there will be more variations among the gestures.

To motivate movement for gestures with distinct hand poses that require a continuous response, the text prompt asks participants to draw random letters in the air with the specified hand pose.

The full corpus contains $10P \times 4S \times 7G \times 3R = 840$ gesture occurrences where P = participants, S = sessions, G = unique gestures, R = repetitions per gesture. There are approximately 96 minutes of continuous recording.

Challenging
Within each sequence:
Different forms of gestures: path and pose gestures
Pose gestures have arbitrary movement
Resolution for hands is low
Continuous gestures with or without a resting pose
Many gesture instances are present
Non-gestures may be present
Between sequences:
High variabilities across participants
Variations in clothing, skin color, lighting
Easy
Fixed camera
Near frontal view acquisition
Within a sequence the same user
Gestures performed by arms and hands
Camera framing full body
Several data sources: skeletal model, user mask, depth, and RGB
Several instances of each gesture for training
Single person present in the visual field
One hand/arm gestures

Table 3.2: Challenging and easy aspects of the dataset.

3.2.2 Data Formats

The data from the Kinect sensor is recorded in a raw binary format at 30 frame per second (FPS). It includes RGB, depth and skeleton data. Both the RGB and the depth data have a resolution of 640 × 480px. The skeleton data contains joint positions for 20 joint types. Figure 3-2[2] visualizes these joint types

During the recording process, the name and the start time (for pre-stroke) for each gesture are recorded according to the prompts. Due the human reaction time, the prompt time may not exactly be the true start time of the pre-stroke. I wrote a program to improve the start pre-stroke and stop post-stroke time labeling based on the initial recorded timings by automatically detecting the rest positions. There are no ground truth time labels for the start and the stop of nucleus phases.

[2]Image from `http://msdn.microsoft.com/en-us/library/jj131025.aspx`

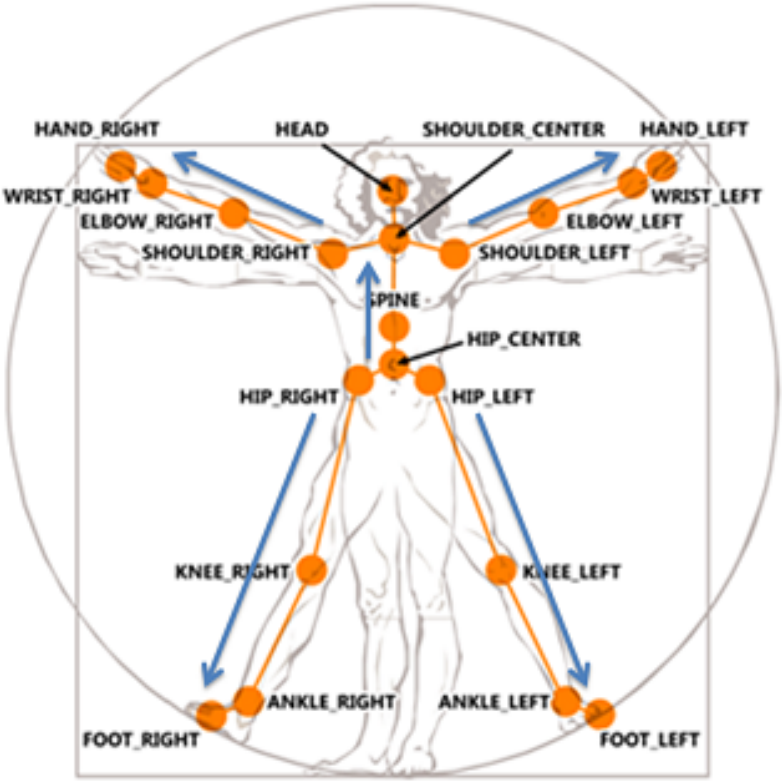


Figure 3-2: Skeleton joint positions.

3.2.3 Qualitative Observations

We find that there is considerable variations in the way participants perform each gesture even though they were given the same demonstration video. Major variations are observed in speed, the magnitude of motion, the paths and hand poses.

For example, some participants do the swipe left and right gestures in a rather straight horizontal line, while others have a more curved path. Some participants do swipe left and right with a palm up pose while others have less distinct hand poses (their hands are more relaxed). Some participants start the circle gesture at the bottom, others start at the top. Some participants do the "circle" gesture clockwise while others do it anti-clockwise. Figure 3-3 shows an illustration of such differences. However, within each participant, the intra-class variation of gestures is smaller, although still present.

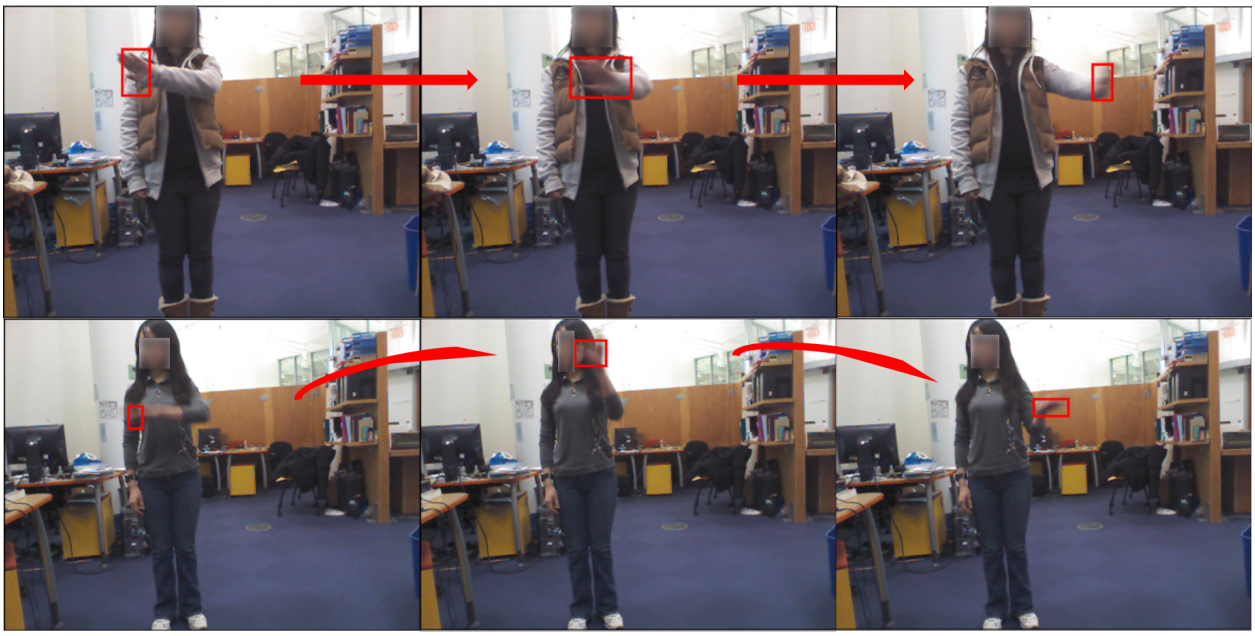


Figure 3-3: Differences between participants for the same swipe right gesture. The first row shows that a user does the Swipe Right gesture in a straight horizontal path with a Palm Up pose; the second row shows that another user does the same gesture in a curve path with less distinct poses.

3.2.4 User Preferences

We did a survey with the participants on questions that can influence gesture interface design. Below are the aggregated results:

- User differences: As an example to show user differences, we asked the participants how they would prefer to do the Circle gesture. 54% of them prefer doing the Circle gesture in clockwise direction, 15% in anti-clockwise direction, and 31% do not care.

- Predefined gestures versus user defined gestures: 90% of the participants prefer to be able to define their own gestures if necessary while 10% of them prefer to follow prefined gestures completely. No one prefers to use a system without any predefined gestures either.

- How to define gestures: 80% prefer defining a gesture by performing it themselves; no one prefers to define gestures solely via rules written in terms of positions and directions of movement of the hands. However 20% prefer to be able to do both.

- Number of repetitions per gesture for training: 50% are willing to give a maximum of 4 – 6 examples, 40% are willing ot give 1 – 3 examples, and 10% are willing to give more than 13 examples. So the average maximum is about 5 repetitions.

- Number of gestures for an application: 80% think 6–10 gestures are appropriate and easy to remember for a given application, while 20% prefers 1 – 5 gestures, giving an average of 7 gestures.

- Intuitiveness of the gesture vocabulary for PowerPoint presentation: the average score is 4 out of 5 where 5 is very intuitive.

3.2.5 Implications for a Gesture Interaction Interface

Based on the observation of the large variation in gesture execution among users and small variations within users, and the fact that a majority of participants prefer defining their own gestures if they do not like the predefined gestures, I suggest that it is more important to optimize user dependent recognition and user adaptation. As no one prefers to define their own gesture at the very beginning, it also means that having a reasonable predefined gesture set and basic user independent model for recognition will be useful too.

Recognition methods based on examples will allow users to train models of their own gestures easily. We also need to develop methods that require relatively few training examples and fast training speed.

3.3 ChAirGest Dataset

ChAirGest dataset [54] is a publicly available dataset[3] for the ongoing open challenge on Multimodal Mid-Air Gesture Recognition for Close HCI[4]. Although the dataset only has path gestures, it has other interesting features which are relevant for evaluating the methods in this thesis:

[3] `https://project.eia-fr.ch/chairgest/Pages/Download.aspx`
[4] `https://project.eia-fr.ch/chairgest/Pages/Scoreboard.aspx`

- It has data from both a Kinect sensor and IMU sensors, allowing me to evaluate the generalizability of my methods for different sensor input and compare recognition performance for different combinations of sensor input.

- It has ground truth labeling of pre-stroke, nucleus and post-stroke phases. Few datasets have gesture phase labeling. This dataset allows me to evaluate my gesture phase segmentation method.

- It represents the scenario where a person sits in front of a desk working on a computer. Because of the absence of the full body and the presence of the distracting foreground (the desk), the Kinect skeleton tracking is less robust. This allows me to evaluate my salience based hand tracking method in the case where the skeleton tracking fails.

- It contains three different rest poses selected according to common user positions when sitting in front of a computer: "hands on the table" when working/typing, "elbows on the table, hands joined under chin" when thinking and "hands on the belly" when watching a movie. These variations, which closely mimic the reality, present challenges to hand tracking, as well as to gesture phase detection as the pre-stroke and the post-stroke are affected the rest positions.

- It contains non-gestures mainly as transitions between rest poses, and hence, can be used to evaluate gesture spotting.

3.3.1 Gestures

The dataset contains a vocabulary of 10 one-hand/arm gestures focusing on close HCI (Table 3.3). The vocabulary has been chosen to present a range of difficulties, including variations in paths, hand rotations, and hand poses. Some gestures have overlapping paths but different hand poses, and this promotes algorithms using fusion from multiple sensors.

The dataset contains 10 participants, each doing 4 recording sessions with 2 different lighting conditions (dark and normal). In a recording session, the participant performs once each gesture class for each resting posture. The full corpus contains $10P \times (2L \times [2S \times 10G \times 3R]) = 1200$ gesture occurrences, where S = subject, L = lighting condition, S = session, G

#	Name of gesture
1	Shake Hand
2	Wave Hello
3	Swipe Right
4	Swipe Left
5	Circle Palm Rotation
6	Circle Palm Down
7	Take From Screen
8	Push To Screen
9	Palm Down Rotation
10	Palm Up Rotation

Table 3.3: ChAirGest gesture vocabulary.

= unique gesture, and R = resting posture. Only three fourths of the corpus (3 recording sessions from each participant) is publicly available and the remaining is used for judging. Hence, in the actual dataset I use, there are 900 gesture occurrences.

3.3.2 Recording Setup

The participant sits on a chair in front of a desk as if working on a computer. He/she wears 4 Xsens IMUs attached under his/her clothes on his/her shoulder, arm, wrist and hand. A Kinect for Windows records the scene from the top of a computer screen with a 30°downward angle.

3.3.3 Data Formats

The Kinect binary format contains the RGB and the depth streams along with the 3D skeleton representation at 30Hz acquired using the official SDK. Each Xsens IMU provides information in: linear acceleration, angular acceleration, magnetometer, Euler orientation and orientation quaternion at 50Hz.

3.3.4 Performance Metric

The challenge uses a combination of existing event-based metrics and a novel time-based metric to evaluate performance.

The two event-based metrics are precision and recall, which are combined to compute an F_1 score[5]. Precision is to the number of correctly detected events divided by the number of returned events and recall is to the number of correctly detected events divided by the number of events in the ground truth [54]. Let $t_{gt\_start\_nucleus}$ and $t_{gt\_stop\_nucleus}$ be the ground truth start time and stop time of a gesture nucleus respectively, and let $t_{alg\_start\_nucleus}$ and $t_{alg\_stop\_nucleus}$ be the corresponding timings returned by a recognition algorithm for the same gesture. A detected gesture event is correct only if the label of the gesture is correct and the timings satisfy the following condition

$$|t_{gt\_start\_nucleus} - t_{alg\_start\_nucleus}| < 0.5 \times (t_{gt\_stop\_nucleus} - t_{gt\_start\_nucleus}) \quad \&\&$$
$$|t_{gt\_stop\_nucleus} - t_{alg\_stop\_nucleus}| < 0.5 \times (t_{gt\_stop\_nucleus} - t_{gt\_start\_nucleus})$$

Figure 3-4 shows an example comparing the results from two algorithms. Even though both of them have correct labels for the detected gesture nucleus, the result from Algorithm 1 is not considered correct because the time discrepancy is larger than the allowed tolerance, which is half of the ground truth duration of the gesture nucleus.

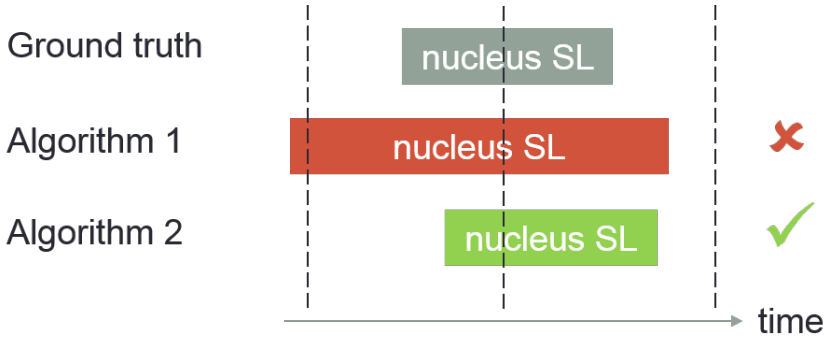


Figure 3-4: Even though the gesture nucleus label detected by Algorithm 1 is correct (SL stands for "swipe left"), the start time difference from the ground truth is too large (larger than half of the ground truth nucleus duration), and hence, it is not considered as a correct detection. The detection from Algorithm 2 is considered correct.

The time-based metric is used to measure the gesture spotting performance and the accuracy of temporal segmentation. The metric is named *Accurate Temporal Segmentation Rate* (ATSR) and represents a measure of the performance in terms of accurately detecting

[5]See Appendix A for details

the start and stop timings of all correctly detected gesture nuclei. The ATSR is computed as follows: for each correctly detected gesture occurrence, the *Absolute Temporal Segmentation Error* (ATSE) is computed according to Equation 3.1; the ATSR metric is computed for a particular sequence with n correctly detected gestures according to Equation 3.2.

$$ATSE = \frac{|t_{gt\_start\_nucleus} - t_{alg\_start\_nucleus}| + |t_{gt\_stop\_nucleus} - t_{alg\_stop\_nucleus}|}{t_{gt\_stop\_nucleus} - t_{gt\_start\_nucleus}} \tag{3.1}$$

$$ATSR = 1 - \frac{1}{n}\sum_{i=1}^{n} ATSE(i) \tag{3.2}$$

The final single metric used by the challenge is the combination of F_1 score and ATSR shown in Equation 3.4, which is of the same form as the F_2 metric and correspondingly weighs F_1 more than ATSR. This is because the recognition of gestures remains more important.

$$\text{Perf} = (1 + 2^2) \times \frac{ATSR \times F_1}{2^2 \times ATSR + F_1} \tag{3.3}$$

$$= 5 \times \frac{ATSR \times F_1}{4 \times ATSR + F_1} \tag{3.4}$$

3.3.5 Evaluation Protocol

All evaluations based on the ChAirGest dataset reported in this thesis use user independent training and testing. The results are the average of 3-fold cross-validation.

4

Hybrid Performance Metric

It is important to have metrics that can accurately evaluate and compare performance of different algorithms for a given task domain, and important to recognize that a metric that is good for one task, is not necessarily appropriate for another task.

4.1 Existing Methods for Error Scoring

Gesture recognition can be considered a sub-domain of human activity recognition. Two basic units of comparison are typically used in this field – frames or events:

Scoring Frames. A *frame* is a fixed-length, fixed-rate unit of time. It is often the smallest unit of measure defined by the system [72], and in such cases approximates continuous time. For example, in our case, a frame is a data frame consisting of RGB data, depth data and skeleton data from the Kinect sensor at 30 FPS. If there is ground truth for each frame, each

frame can be assigned to one of: true positive (TP), true negative (TN), false positive (FP) or false negative (FN). Commonly recommended frame-based metrics include: true positive rate (TPR = $\frac{TP}{TP+FN}$), false positive rate (FPR = $\frac{FP}{TN+FP}$), precision ($\frac{TP}{TP+FP}$).

Scoring Events. An *event* is a variable duration sequence of frames within a continuous time-series. It has a start time and a stop time. Given a test sequence of g known events, $E = [e_1, e_2, \ldots, e_g]$, a recognition outputs h return events, $R = [r_1, r_2, \ldots, r_h]$. There is not necessarily a one-to-one relation between E and R. A comparison can instead be made using alternative means such as dynamic time warping (DTW) [9] or edit distances [24]. An event can then be scored as either correctly detected (C), falsely inserted (I), or deleted (D) [72]. Event scores can be summarized by precision ($\frac{C}{h}$), and recall ($\frac{C}{g}$).

4.2 Shortcomings of Conventional Performance Characterization

Either frame-based or event-based metrics alone may not be adequate for evaluating a real-time continuous gesture recognition system handling different types of gestures. We illustrate this using examples from related work.

Both the ChaLearn Gesture Challenge 2012 [24] and the Multimodal Gesture Recognition Challenge 2013 [19] use the Levenshtein edit distance[1], $L(R, E)$, between the ordered list of recognized events (R) and the ground truth events (E) to evaluate performance. However, such event-based metrics that ignore the timing offset errors are inadequate to identify true positives in sequences. For example, in Figure 4-1, both Algorithm 1 and Algorithm 2 would have the same score using their metric. However, Algorithm 1 is in fact worse because the recognized event B is mistakenly considered as a true positive in the minimum edit distance calculation, and the number of mistakes should be 3. Consider a real-time application, if the user does gesture A, it cannot be considered correct if the system classifies it as gesture B.

Song et al. [61] and Morency et al. [41] used frame-based metrics to evaluate their continuous gesture recognition systems. Frame-based metrics consider timing inherently, but there are artifacts, such as fragmenting and merge errors [72] in the results that cannot be captured

[1] `http://en.wikipedia.org/wiki/Levenshtein_distance`

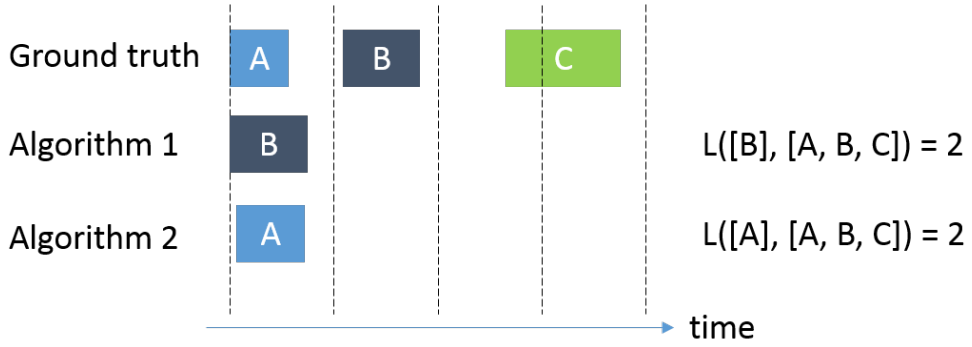


Figure 4-1: Considering events as an ordered list without timing information does not give a fair comparison for recognition performance. Applying the edit distance score used in the challenges, both algorithms have 2 mistakes. However if we consider the timing of the recognition, Algorithm 1 would have 3 mistakes.

by this type of metrics. For example, in Figure 4-2, Algorithm 1 has a higher frame-based TPR. However, depending on the application, Algorithm 2 can have a better performance. Suppose we cast this example into a concrete scenario of a gesture-controlled presentation application, if the user does a "swipe left" gesture, using Algorithm 1, the system would respond three times and change the slides three times; while using Algorithm 2, the system would respond one time which is the desired outcome. The same argument can also be made for merge errors (see Figure 4-3). This scenario shows that frame-based evaluation is less relevant for gestures requiring discrete responses.

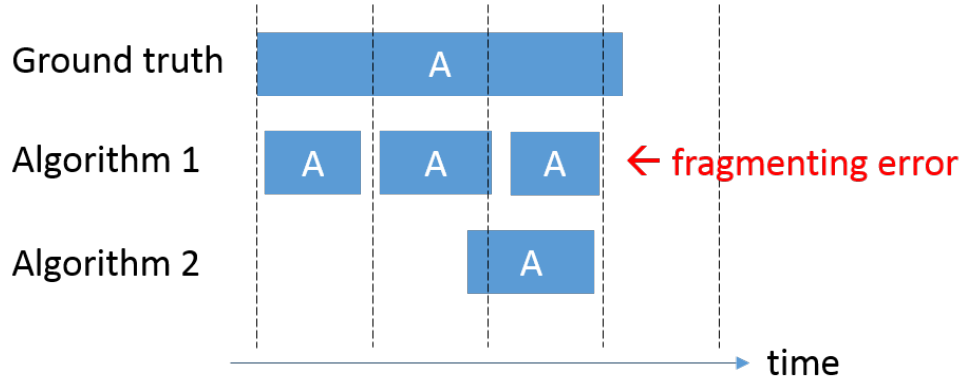


Figure 4-2: Even though Algorithm 1 has a higher true positive rate, it has more fragmenting errors. For certain applications, Algorithm 2 would have a better performance.

There are situations where frame-based metrics are more relevant than event-based met-

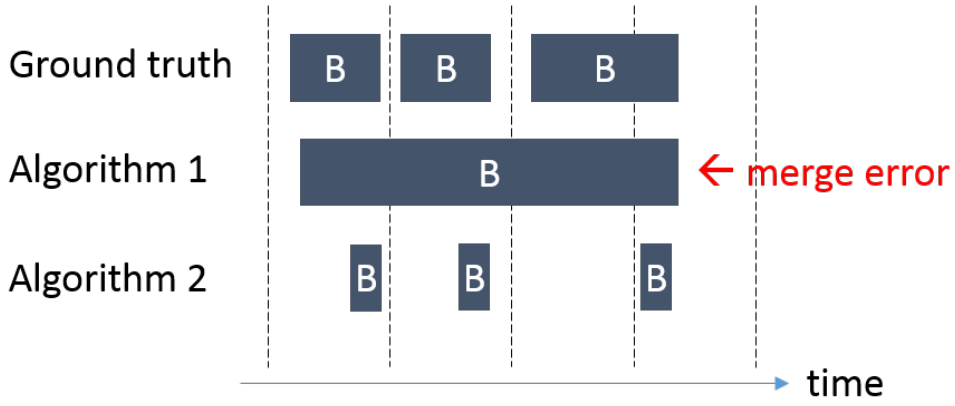


Figure 4-3: Even though Algorithm 1 has a higher true positive rate, it has more merge errors. For certain applications, Algorithm 2 would have a better performance.

rics as well. Consider the same recognition results in Figure 4-2, but this time gesture A is the "point" gesture requiring continuous frame-by-frame response from the system (e.g., the system shows a point cursor moving around according to where the user points at). In this case, Algorithm 1, having a higher frame-based TPR, would have better performance.

4.3 Hybrid Performance Metrics

The examples in the previous section demonstrate the requirement of a hybrid performance evaluation system. I believe that all three types of information – frames, events and timings – are relevant for a real-time activity/gesture recognition system, and should be included in the metric. More importantly, as different categories of activities/gestures require different kinds of responses from the system, it is necessary to identify which metric is appropriate for which category of activities/gestures: the event-based metric is appropriate for discrete *flow* gestures and the frame-based metric is appropriate for continuous *flow* gestures.

There are previous works that consider hybrid metrics. Ruffieux et al. combined a time-based metric with an event-based metric (see Section 3.3.4). They included timing offset errors explicitly in the metrics, but they did not consider frame-based metric. Ward et al. [72] proposed a comprehensive scheme to combine both frame and event scoring, but they did not consider how the different types of metrics are relevant for different categories of activities.

The following section explains the details of the hybrid performance metric I propose.

4.3.1 Metric for Discrete Flow Gestures

For discrete *flow* (DF) gestures, the system responds at the end of the nucleus phase, therefore the evaluation should be event-based. Let $T_{\text{gt\_start\_pre}}$ be the ground truth start time of the pre-stroke phase and $T_{\text{gt\_stop\_post}}$ be the ground truth stop time of the post-stroke phase. A recognized event is considered a TP if the time of response (T_{response}) occurs between $T_{\text{gt\_start\_pre}}$ and $T_{\text{gt\_stop\_post}} + 0.5 \times (T_{\text{gt\_stop\_post}} - T_{\text{gt\_start\_pre}})$. We allow some margin for error because there can be small ground truth timing errors[2]. Once a TP event is detected, the corresponding ground truth event is not considered for further matching, so that multiple responses for the same gesture (fragmenting errors) will be penalized. We then can compute event-based precision, recall and F_1 score for DF gestures:

$$\text{precision}^{\text{DF}} = \frac{\text{\# TP DF events}}{\text{\# recognized DF events}} \tag{4.1}$$

$$\text{recall}^{\text{DF}} = \frac{\text{\# TP DF events}}{\text{\# ground truth DF events}} \tag{4.2}$$

$$F_1^{\text{DF}} = 2 \cdot \frac{\text{precision}^{\text{DF}} \cdot \text{recall}^{\text{DF}}}{\text{precision}^{\text{DF}} + \text{recall}^{\text{DF}}} \tag{4.3}$$

For discrete *flow* gestures, we also define a Responsiveness Score (RS) as the time difference in seconds between the moment when the system responds and the moment when the hand goes to a rest position or changes gesture. Let N_{TP} be the number of true positives, then

$$RS = \frac{\sum_{i=1}^{N_{TP}} T_{\text{gt\_stop\_post}} - T_{\text{response}}}{N_{TP}} \tag{4.4}$$

A positive score means the responses are before the end of the post-strokes, hence higher scores are better.

[2]Pre-stroke and post-stroke timings are used because there may not be ground truth timings for nucleus phases, such as in the YANG dataset. Manual labeling of the start and stop timings of nucleus phases may be too time consuming.

4.3.2 Metric for Continuous Flow Gestures

For continuous *flow* (CF) gestures, the system responds frame by frame, so it is more appropriate to use frame-based evaluation. For all the frames that are recognized as continuous gestures, we can compute the number of TPs by comparing them with the corresponding frames in the ground truth. Then, we can compute frame-based precision, recall and F_1 score for all the frames corresponding to CF gestures:

$$\text{precision}^{\text{CF}} = \frac{\#\text{ TP CF frames}}{\#\text{ recognized CF frames}} \tag{4.5}$$

$$\text{recall}^{\text{CF}} = \frac{\#\text{ TP CF frames}}{\#\text{ ground truth CF frames}} \tag{4.6}$$

$$F_1^{\text{CF}} = 2 \cdot \frac{\text{precision}^{\text{CF}} \cdot \text{recall}^{\text{CF}}}{\text{precision}^{\text{CF}} + \text{recall}^{\text{CF}}} \tag{4.7}$$

The average of the two F_1 scores can give an overall indication of the performance of the system.

The retinal image produced by the hand of a gesticulating speaker is never the same from moment to moment, yet the brain must consistently categorize it as a hand.

Semir Zeki, *The visual image in mind and brain*

5

Hand Tracking

The difficulty of hand tracking (localizing and segmenting hands) varies with the complexity of the input scene, which in turn depends on the setup of the interaction interface. Gesture input is particularly useful for large displays for which keyboard and mouse input become cumbersome. Depending on the orientation of the display (horizontal or vertical), the sensor setup can be different, and hence, the characteristics of the input data can be different. As a result, I developed different approaches for hand tracking using data from the Kinect sensor under different system and sensor setup.

5.1 Hand Tracking for Horizontal Display

Horizontal displays are also called tabletop displays, and are useful for collaborative work and touch-based interaction.

5.1.1 System Setup

Our custom tabletop structure includes four 1280 × 1024 pixel projectors (Dell 5100MP) that provide a 2560 × 2048 pixel resolution display. The display is projected onto a flat white surface digitizer (GTCO Calcomp DrawingBoard V), which uses a stylus as an input device. The digitizer is tilted 10 degrees down in front, and is placed at 41in (104cm) above the floor, following the FAA's design standard to accommodate the 5^{th} through 95^{th} percentiles of population. The projected displays are aligned to produce a single seamless large display area using the ScalableDesktop Classic software developed by Scalable Display Technologies[1]. The graphics card used is AMD Radeon™TM HD 6870 and the operating system used is Ubuntu 11.10.

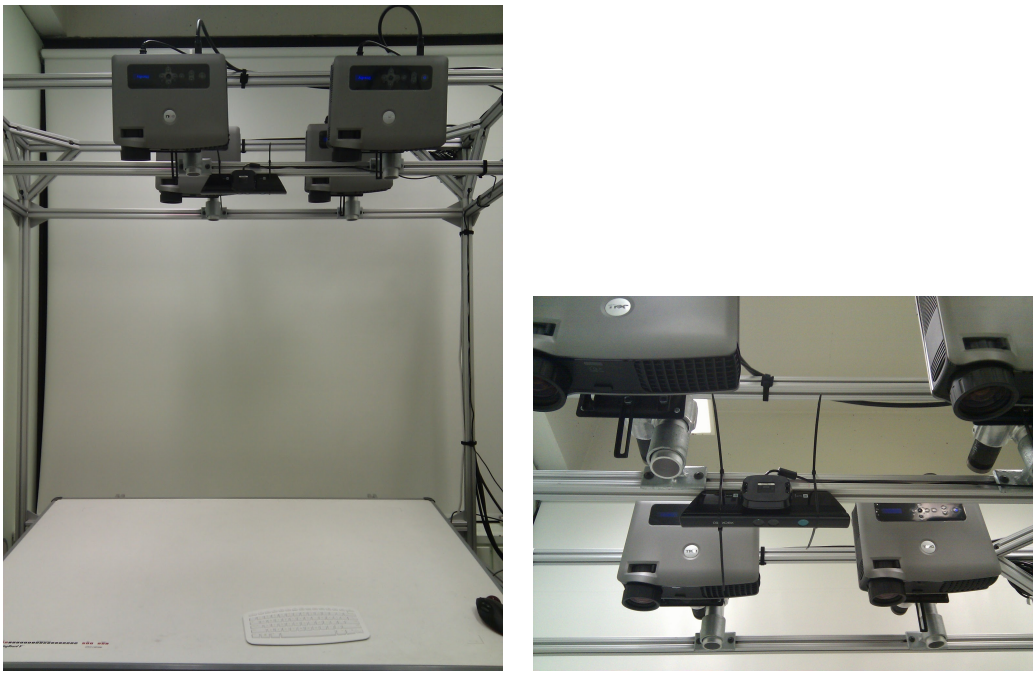


Figure 5-1: System setup for a horizontal display.

One Kinect sensor is placed above the center of the tabletop at the same level as the projectors. Figure 5-1 shows the setup. I use the depth data for hand tracking because it is less sensitive to lighting conditions. This is particularly relevant for our projection system. The Dell 5100MP projector uses a spinning color wheel to modulate the image. This produces a visible artifact on the screen, referred to as the "rainbow effect", with colors

[1] `http://www.scalabledisplay.com/products/software/scalabledesktopclassic`

separating out in distinct red, green, and blue. At any given instant in time, the image on the screen is either red, or green, or blue, and the technology relies upon people's eyes not being able to detect the rapid changes from one to the other. However, when seen through a RGB camera, the effect is very obvious, and this would greatly affect hand segmentation if I were to use the RGB images.

The depth sensing video stream has a resolution of 640×480 pixels with 11-bit depth value. The depth value increases as the distance of the object from the sensor increases. The tabletop surface is about 1.2m away from the Kinect sensor which allows us to have a relatively good depth resolution. I use the open source OpenNI framework[2] and its Kinect driver[3] to get both the depth and RGB data streams.

5.1.2 Kinect Calibration

In order to develop an interactive interface, it is necessary to accurately map a point in the depth image to a point on the display. I do this by projecting a checkerboard image on the tabletop display, and placing some wooden blocks at the corners of the checkerboard image to create the depth differences so that the depth sensor can capture these corners (see Figure 5-2). I manually labeled 16 pairs of corresponding points on the display and the depth image. Then I apply undistortion to the depth image and planar homography to find the mapping.

Planar homography is a projective mapping from one plane to another. In this case, I map points on the plane of the depth image to the points on the plane of the display. To evaluate the result of the calibration, I obtain a new set of manually labeled corresponding points on the display and the depth image. I then transform the coordinates of the points on the depth image to the coordinates on the display using the calibration result, and find the Euclean distance (error) between the transformed coordinates and the labeled coordinates of the points on the display. The average errors in the x-axis and y-axis are 4.3px and 8.9px respectively, which are 0.21cm in x, and 0.4cm in y in physical distance. The average width of the index fingertip is about 1.4cm, so the error is less than 30% of the fingertip width.

[2]https://github.com/OpenNI/OpenNI
[3]https://github.com/avin2/SensorKinect

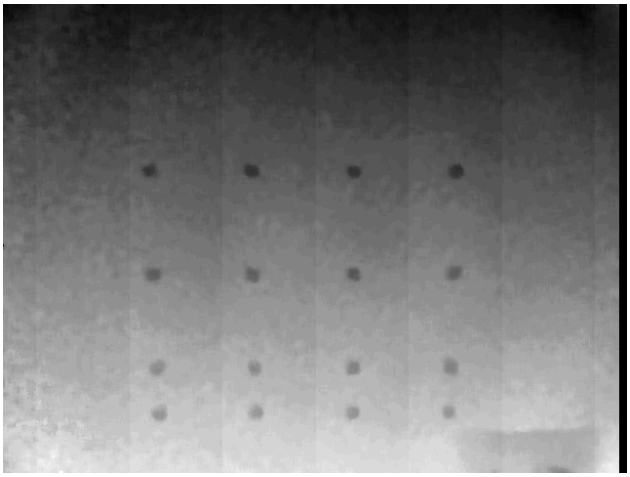


Figure 5-2: Kinect calibration. The darker squares are the wooden blocks. The intensity of the gray level image is proportional to the distance from the Kinect sensor after applying histogram equalization, so closer things look darker.

5.1.3 Hand and Fingertip Tracking

As the Kinect sensor is looking down at the tabletop, only the upper limbs of the users are in the field of view of the sensor, so skeleton tracking from the Kinect SDK does not work. Hence, I developed a hand tracking process from scratch, which consists of 4 steps:

1. Background subtraction
2. Upper limb and hand segmentation
3. Fingertips tracking
4. Kalman Filtering

The following subsections explain these steps in details. Many of the computer vision methods I use are based on the OpenCV[4] library and its Java interface JavaCV[5].

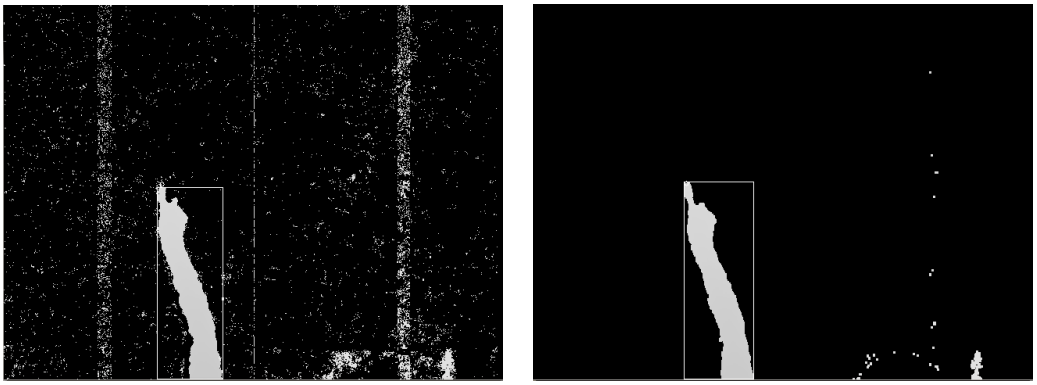
Background Subtraction

While the background - i.e., the tabletop - is relatively static, there is still noise from the depth sensor. I use the averaging background method, which learns a model of the background in terms of the average distance and the average difference in distance to each pixel

[4]http://opencv.willowgarage.com/wiki/
[5]http://code.google.com/p/javacv/

in the depth image. The average values are based on the initial 30 frames with no hands in the scene; the average frame-to-frame absolute difference for my system is 1.3mm. For the subsequent frames, any value that is 6 times larger than this (i.e., at least 7.8mm above the surface) is considered foreground.

To clean up the background subtracted depth image, I use 1 iteration of morphological opening to clear out small pixel noise. Morphological opening is a combination of erosion and dilation. The kernel of erosion is a *local minimum* operator, while that of dilation is a *local maximum* operator. Figure 5-3 shows the difference after using morphological opening.



(a) Without using morphological opening.

(b) Using morphological opening to clear out small pixel noise.

Figure 5-3: Background subtraction.

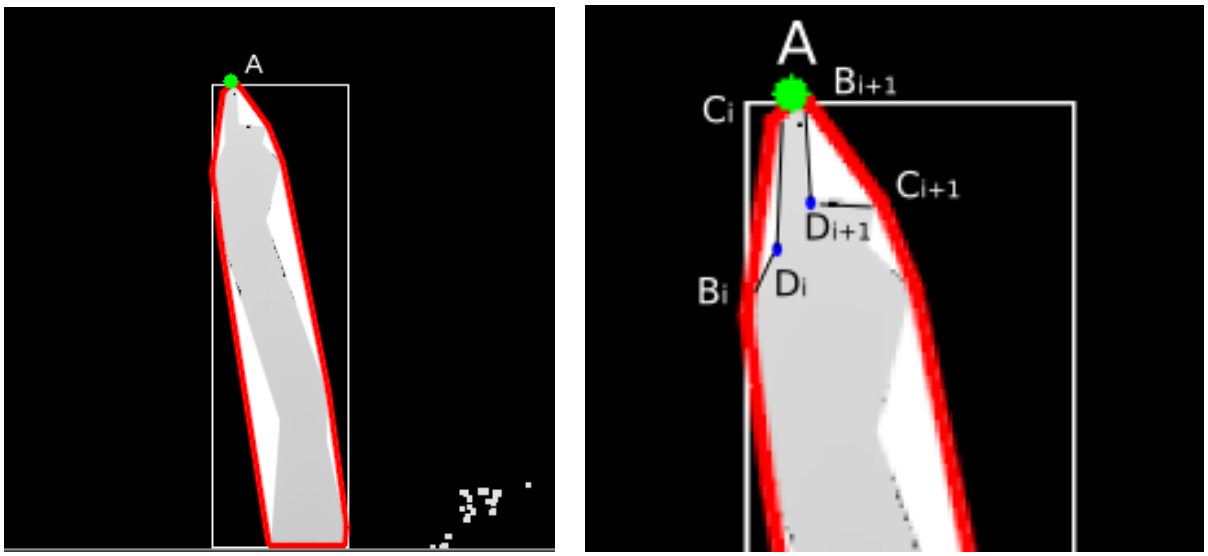
Upper Limb and Hand Segmentation

With the cleaned up, background subtracted depth image, we find connected components by finding all contours with perimeters greater than a threshold value of 300mm. These components are considered to be the upper limbs. The threshold value is roughly the lower bound of the perimeter of a hand. We use this lower bound because the perimeter of the upper limb changes depending on the position of the hand relative to the table. The smallest perimeter occurs when the hand is at the edge of the table.

Each upper limb is approximated with a convex hull and a bounding box. The hand region is at either end of the bounding box depending on the position of the arm relative to the table.

Fingertips Tracking

The estimation of the hand model is based on geometric properties of the hand. I first compute the convexity defects (shown by the white areas in Figure 5-4(a)) from the convex hull of the upper limb. These convexity defects offer a means of characterizing the hand shape [11]. As Figure 5-4(a) shows, an extended finger has one convexity defect on each side, and the two adjacent sides of the defects form a small acute angle (e.g., point A in Figure 5-4(a)). We iterate through two adjacent convexity defects each time, for example, $\triangle B_i C_i D_i$ and $\triangle B_{i+1} C_{i+1} D_{i+1}$ in the closeup view in Figure 5-4(b). If the angle between the two sides, $C_i D_i$ and $B_{i+1} D_{i+1}$ is smaller than a threshold value (45 °), we mark the middle point of $C_i B_{i+1}$ as potential fingertips. The distance between the `depth_points` (the point on the defect that is farthest from the edge of the hull [11], e.g., D_i, D_{i+1}) of the adjacent convexity defects also has to be greater than the finger width threshold value (14mm).



(a) (b) Closeup view.

Figure 5-4: The white triangular areas are convexity defects and the red outline is the convex hull.

We further refine the fingertip position by searching in the direction of the finger for a sharp change in the gradient of the depth value (i.e., when the gradient is greater than 0.05). Figure 5-5 shows the result of fingertip tracking even with multiple fingers in the image[6].

[6] A video of this demo can be found at: `https://www.youtube.com/watch?v=_LZ4RJYBgZ4`

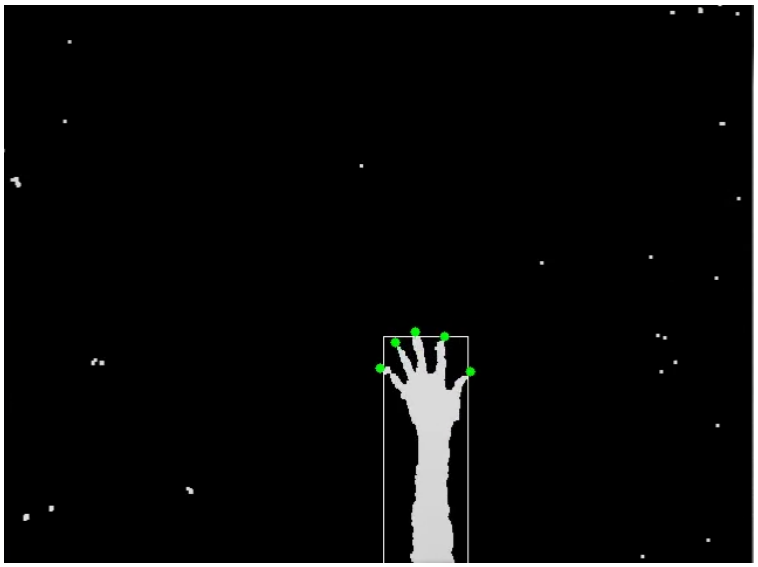


Figure 5-5: Fingertip tracking results: green dots are detected fingertips.

Kalman Filtering

Like [45] and [26], I use a Kalman filter to further improve tracking accuracy. The dynamic state of the fingertip can be summarized by a 4-dimensional vector, $\underline{x}_t$, of two position variables, x and y, and two velocities, v_x and v_y.

$$\underline{x}_t = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_t$$

The measurement is a 2-dimensional vector, $\underline{z}_t$, of the measured x and y coordinates of the fingertip.

Assuming no external control, the a priori estimate $\underline{x}_t^-$ of the state is given by:

$$\underline{x}_t^- = F\underline{x}_{t-1} + \underline{w}_t$$

F is the 4-by-4 *transfer matrix* characterizing the dynamics of the system with the following values:

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

assuming constant instantaneous velocities at time t, and that time is measured in frames. $\underline{w}_t$ is the *process noise* associated with random events or forces that directly affect the actual state of the system. I assume that the components of $\underline{w}_t$ have Gaussian distribution $N(\underline{0}, \Sigma_t)$ for some 4-by-4 covariance matrix Σ_t which has the following values:

$$\Sigma_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

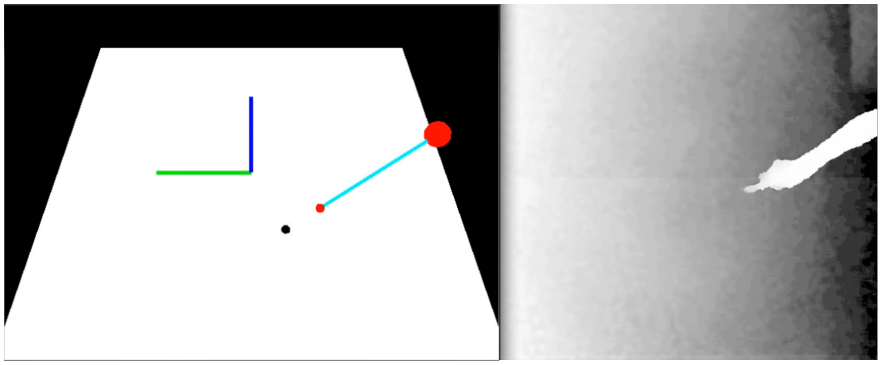
The larger variance values in the last 2 values in the diagonal indicate greater uncertainty in the velocities, as they can have big changes instantaneously.

5.1.4 3D Hand Model and Touch Detection

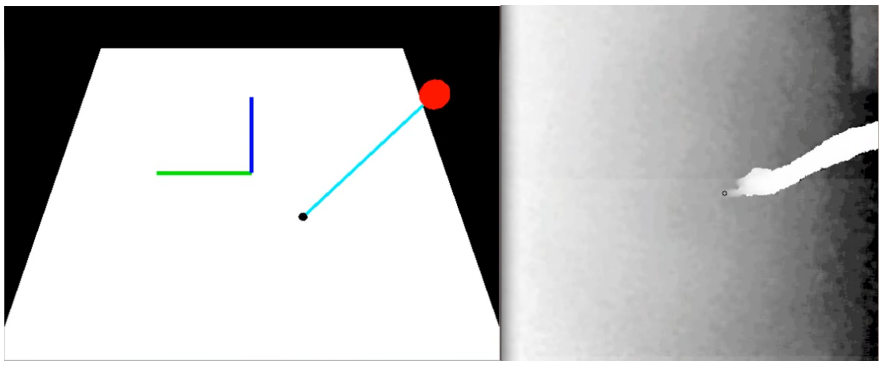
I use a simplified 3D skeletal hand model, which includes hand position, orientation and fingertip positions, to enable touch-based direct manipulation.

I use a line segment in 3D space to model the upper limb by computing the arm joint position and an average of fingertip positions from the depth data. The tabletop is modeled as a plane. In Figure 5-6(a), the image on the left shows the 3D model: the big red sphere is the arm joint and the small red sphere is the average fingertips position.

The black dot in Figure 5-6(a) is the intersection between the line extended from the upper limb segment and the tabletop plane. If the intersection is the same as the average fingertip position, the finger(s) is in contact with the table (Figure 5-6(b)). As the tabletop plane is computed from averaged depth data, this touch detection method is more robust



(a) Fingertip is not in contact with the tabletop.



(b) Finger is in contact with the tabletop.

Figure 5-6: On the left are images showing 3D model of the upper limb and the tabletop surface; on the right are are corresponding depth mapped images. The vertical blue line is the z axis and the green horizontal line is the y axis.

than comparing the depth value of the fingertip with the depth of the tabletop surface at one point.

5.1.5 Evaluation

Figure 5-7 shows an example of the fingertip tracking result on an actual display[7]. The red dot is the detected fingertip position. I evaluate the accuracy of our fingertip tracking method by comparing the detected fingertip position with the manually labeled position in a sequence of video frames. In this evaluation, only one extended finger was used. My method finds the fingertips with an average error (Euclidean distance) of 5.3px, about 10mm in physical distance on the projected display. The error rate also informs us the size of the virtual objects we should have in our applications, i.e., they need to be at least 10mm in

[7]A video of this demo can be found at: `https://www.youtube.com/watch?v=8tr0ZZ-4KMc`

radius, in order to increase the accuracy of the manipulative interaction. This result is comparable with the accuracy in [26], but my system has no restriction on the angle of the fingers with respect to the surface.

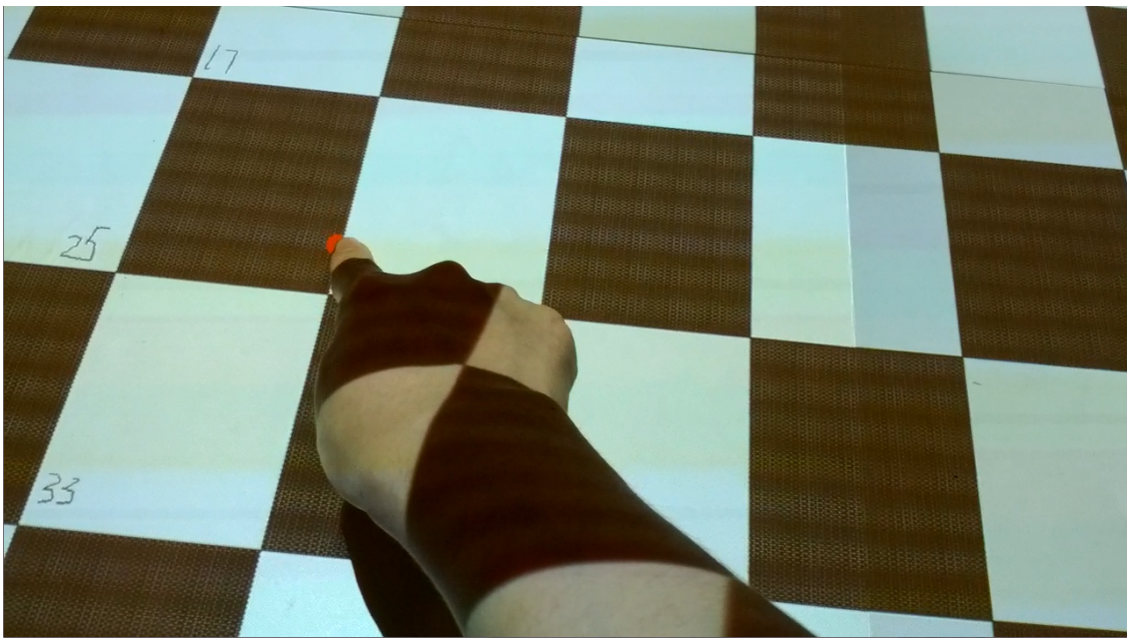


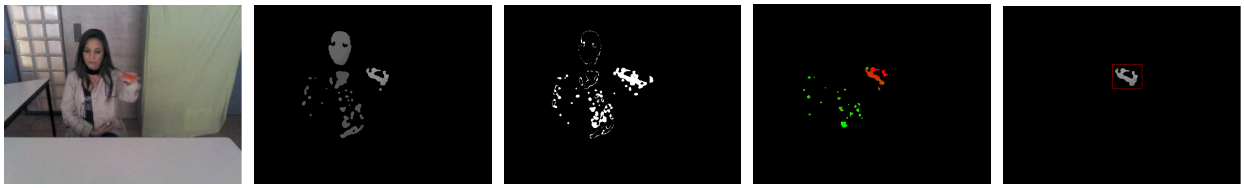
Figure 5-7: Tracking result displayed on the tabletop surface. The red dot is the detected fingertip position.

5.2 Hand Tracking for Seated Interaction with Vertical Display

As most people interact with computers while sitting in front of a desk, it is important to consider gesture input in this setting as well. In a setup with a vertical display, the Kinect sensor is usually placed near the display facing the user. In this case, the background scene is more complex compared with the tabletop setup. Because of the absence of the full body and the presence of the distracting foreground (the desk), the Kinect skeleton tracking is less robust in the seated mode, and in particular, hand joint tracking fails when the hands are close to the body or are moving fast. To improve hand tracking for seated interaction, I developed a salience based hand tracking method.

5.2.1 Gesture Salience Detection

Similar to Marcos-Ramiro et al. [38], we define gesture *salience* as a function of both the closeness of the motion to the observer (e.g., the sensor) and the magnitude of the motion, i.e., the closer the motion and the larger the magnitude of the motion, the more salient the motion is and the more likely it is from the gesturing hand. There are 4 steps in our method (Figure 5-8).



(a) (b) (c) (d) (e)

Figure 5-8: Gesture salience detection steps: (a) RGB image under low lighting condition; (b) depth map D_t filtered by skin and user mask, $M_t^{S \wedge U}$. False detection of skin is due to the similar colors between clothes and skin; (c) motion mask, $M_{t \vee t-1}^{M}$, indicating moved pixels for time t and $t-1$; (d) salience map with red color indicating high probability of the salience; (e) final gesture salience bounding box, B_t. (Best viewed in color. Based on data from the ChAirGest corpus.)

Skin Segmentation

We use an off-the-shelf simple skin color detection method to compute a binary skin mask at time t, M_t^S, based on the RGB image converted to the YCrCb color space. We also find the user mask, M_t^U obtained from the Kinect SDK based on the depth image. We align the two to find their intersection $M_t^{S \wedge U}$, which indicates the user's skin region.

Motion Detection

The depth data is first clipped to a maximum value $\text{depth}_{\text{max}}$ of 2m and then scaled to a single byte value between 0 (depth of 2m) and 255 (depth of 0m).

We compute the motion mask for the current depth frame at t based on 3 frames at t, $t-1$ and $t-2$. We first filter each depth frame by the user and skin mask $M_t^{S \wedge U}$, and then smooth it through a median filter to obtain D_t (Figure 5-8(b)). Equation (5.1) computes the binary mask, $M_{t \vee t-1}^{M}$, indicating pixels whose depth values have changed from time $t-1$

to t (Figure 5-8(c)). $D_{t \vee t-1}$ is the absolute difference between D_t and D_{t-1}, and T is the threshold operator that filters out small changes in depth value (with a threshold of 15mm). To obtain the motion mask, M_t^M for time t, we use $M_{t-1 \vee t-2}^M$, the motion mask for $t-1$ and $t-2$ as well (see Equation (5.2), AND and XOR are indicated by $\wedge$ and $\oplus$).

$$M_{t \vee t-1}^M = T(D_{t \vee t-1}) \tag{5.1}$$

$$M_t^M = M_{t \vee t-1}^M \oplus (M_{t \vee t-1}^M \wedge M_{t-1 \vee t-2}^M) \tag{5.2}$$

Salience Map

We compute histograms of depth values in both D_t and $D_{t \vee t-1}$ and then apply histogram equalization[8] to obtain cumulative distributions H_t and $H_{t \vee t-1}$. H_t represents the probability of salience given a depth value, while $H_{t \vee t-1}$ represents the probability of salience given a depth difference value. The lower the depth value or the higher the depth difference value, the higher the salience probability. We use histogram equalization to reduce the effect of outliers, so that a single large depth value will not suppress the salience probabilities of other depth values. The salience map (Figure 5-8(d)) can then be computed for each pixel (x, y):

$$S_t(x, y) = H_t(D_t(x, y)) \times H_{t \vee t-1}(D_{t \vee t-1}(x, y)) \times M_t^M$$

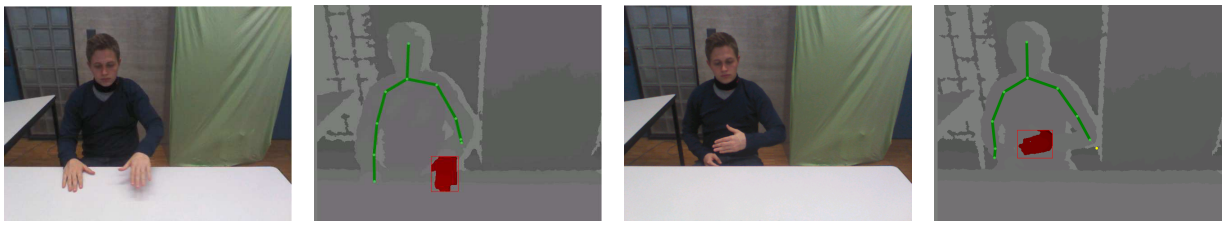
The multiplication of the binary motion mask M_t^M allows us to consider only the motion due to the user at t.

Salience Location

The final step of locating the most salient region in a frame is finding the contour, C_t, from the salience map S_t that has a perimeter greater than the smallest possible hand perimeter and with the highest average salience for all the pixels inside the contour.

When motion is slow, the motion mask usually indicates the edge of the moving object. As a result, the center of C_t may not be the center of the moving object (in this case, the user's hand). Hence, I use 2 iterations of camshift [12] on the depth image D_t with a

[8] `http://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf`



(a) (b) (c) (d)

Figure 5-9: Comparison of hand tracking results. Our method (red region) gives more reliable result on hand tracking compared to the off-the-shelf Kinect software (green line). (Best viewed in color. Based on data from the ChAirGest corpus.)

starting search location at the center of C_t to refine the final bounding box, B_t, of gesture salience (Figure 5-8(e)). Camshift is similar to the mean-shift algorithm which finds the local maximum in a distribution. The difference is that camshift uses an adaptive search window whereas mean-shift uses a fix search window. In this case, the higher a pixel value in D_t, the closer it is to the sensor, and hence the more salient. The camshift algorithm is used to find the region with the highest salience.

5.2.2 Evaluation

The red regions in Figure 5-9 show examples of my hand tracking result, which demonstrates that my salience detection method is more reliable than hand joint locations from the Kinect SDK. Using the ChAirGest dataset and the same recognition method, Table 5.1 gives a quantitative comparison between the two hand tracking methods. The same motion features from the IMU attached to the hand are used in both cases. It shows that using my salience detection method to extract hand position features gives a 2.7% absolute increase in gesture recognition F_1 score compared to using the hand joint position from the Kinect SDK.

	Hand position from salience detection	Hand position from Kinect skeleton
F_1 Score	**0.897 (0.01)**	0.870 (0.02)

Table 5.1: Comparison of the average 3-fold cross validation results for different hand tracking methods using the ChAirGest dataset. Values in parentheses are standard deviations.

It is precisely this variation – and the apparent success of our visual system and brain in achieving recognition in the face of it – that makes the problem of pattern recognition so interesting.

Irving Biederman, *Higher-level vision*

6

Hand Features and Representations

This chapter discusses different hand feature representations and encoding methods I use to produce feature vectors for input to the recognition module. One feature vector is computed for each input frame streamed from the sensor(s) to form a sequence of feature vectors.

As features can have different numeric ranges, to avoid features with greater numeric ranges dominating those with smaller numeric ranges, it is important to scale the feature vectors before feeding them into the recognition module [73]. Hence, after computing the feature vectors, the last step is always standardizing all the features to have mean 0 and standard deviation 1 using all the training data. During testing, the data are standardized using the means and the standard deviations from training.

6.1 Hand Motion Features

Motion features are important for representing path gestures.

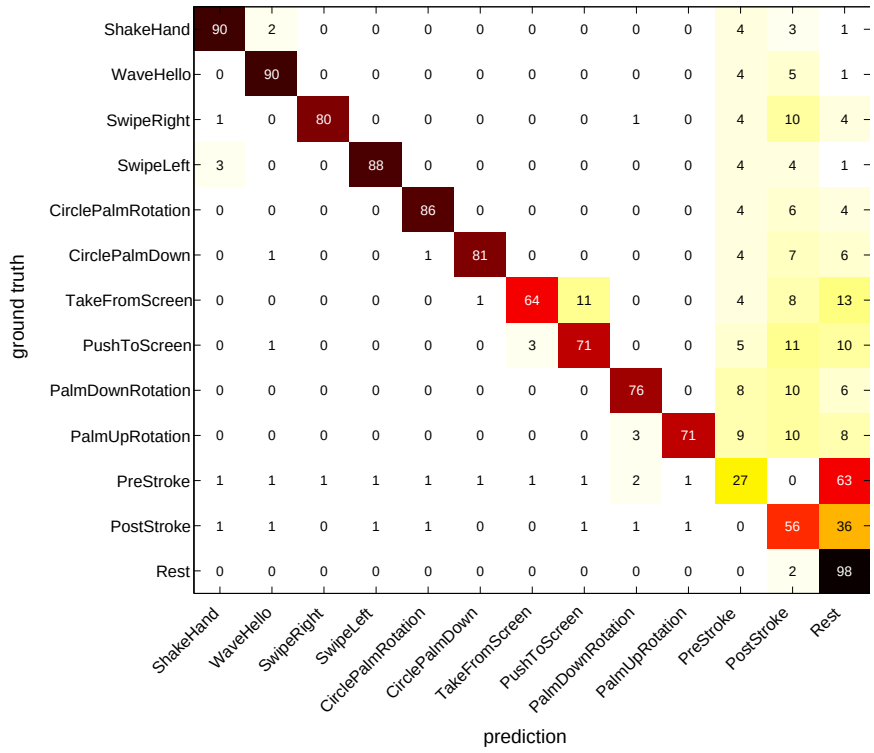
It is relatively easy to obtain motion features from IMUs if they are present. From the ChAirGest dataset, I use linear acceleration (x, y, z), angular velocity (x, y, z) and Euler orientation (yaw, pitch, roll) from the IMU on the hand to form a 9-dimensional feature vector $\underline{x}_t^{\text{imu}}$ at every time frame t.

As IMUs cannot provide hand position information, I use hand position information from the hand tracking described in the previous section. With the Kinect sensor data from the ChAirGest dataset and using the gesture salience based hand tracking method, I extract the position of a gesturing hand in (x, y, z) coordinates relative to the shoulder center joint, forming a 3-dimensional vector $\underline{x}_t^{\text{kinect}}$ (shoulder center joint position from the Kinect SDK is relatively accurate under most situations). Combining the two, we have a 12-dimensional feature vector $\underline{x}_t = [\underline{x}_t^{\text{kinect}}, \underline{x}_t^{\text{imu}}]$. The evaluation in Table 6.1 shows that adding position information can improve recognition accuracy further for path gestures.

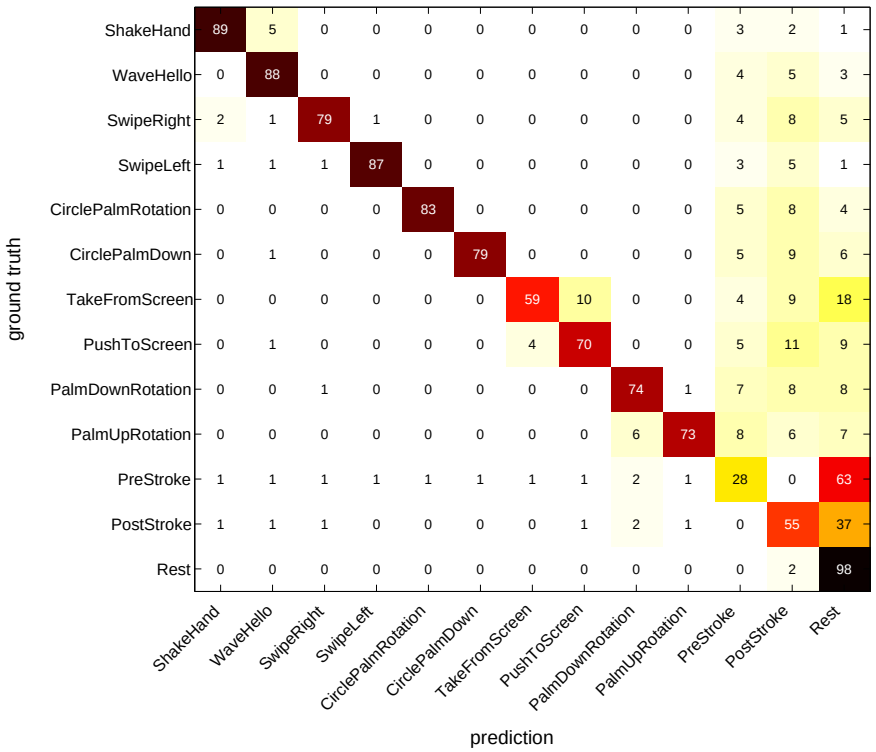
	Hand position from salience detection & IMU, $[\underline{x}_t^{\text{kinect}}, \underline{x}_t^{\text{imu}}]$	IMU only, $\underline{x}_t^{\text{imu}}$
F1 Score	**0.897 (0.03)**	0.863 (0.02)
ATSR Score	**0.907 (0.02)**	0.918 (0.02)
Final Score	**0.898 (0.02)**	0.874 (0.02)

Table 6.1: Comparison of the average 3-fold cross validation results for different motion feature vectors using the ChAirGest dataset. Values in parentheses are standard deviations.

Figure 6-1 shows a comparison between the confusion matrices based on results using different motion features. The per frame classification accuracy increases for most gestures when hand position features are added. For example, Wave Hello and Shake Hand have very similar motion trajectory. One difference between them is that the hand position for Shake Hand is lower. When hand position features are used, the confusion between Shake Hand and Wave Hello decreases.



(a) Using hand position features and IMU features.



(b) Using IMU features only.

Figure 6-1: Per frame classification confusion matrices. The numbers are percentages. The darker the color the higher the percentage.

It is important to check the distribution of the feature values in order to model them accurately. Figure 6-2(a) shows the histograms of the standardized (x, y, z) coordinates of relative position, velocity and acceleration from one user's data in the YANG dataset. The peak values corresponds to the rest position because it is the most frequent pose in the recording. Figure 6-2(b) shows the histograms of the same data excluding those from the rest position. The distribution roughly follows Gaussian or mixture of Gaussians distributions which means that it will be reasonable to model them using these probability distributions in the gesture model.

6.2 Hand Pose Features

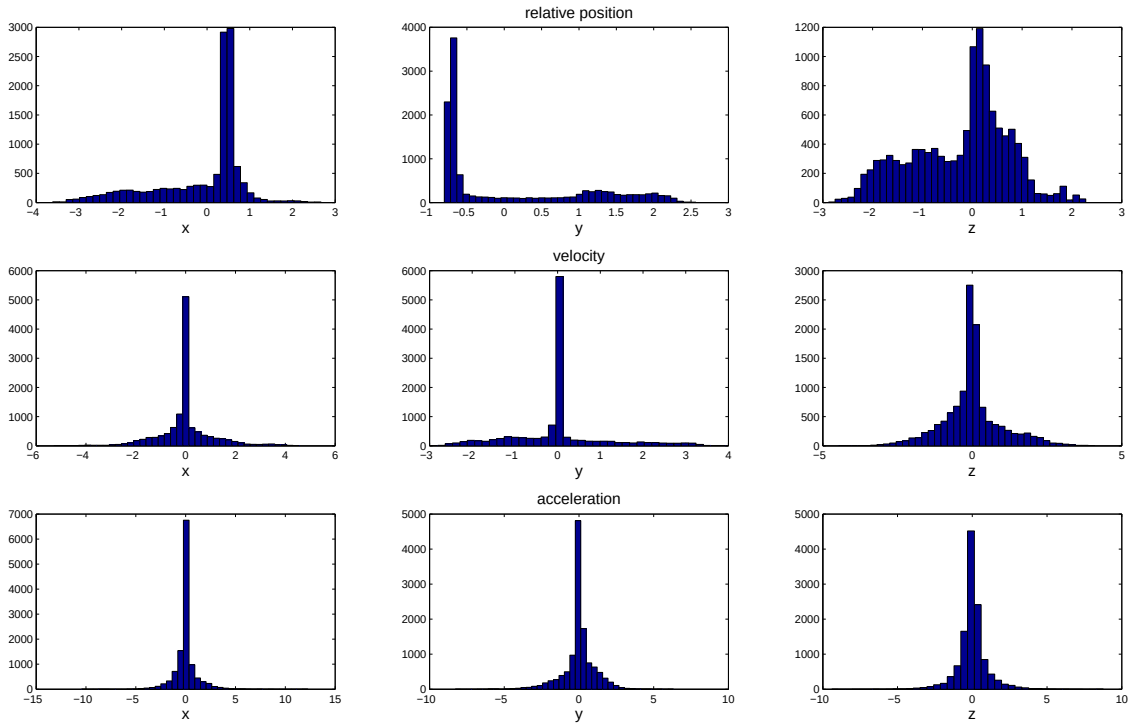
If no IMU is present, we can still compute velocity and acceleration from relative positions, but we will lose the information on hand orientation, which is important to distinguish gestures with the same path but different hand poses, e.g., gestures in the ChAirGest dataset. In this case, I use the HOG feature descriptor derived from the Kinect data to represent hand poses. The HOG feature descriptor can be computed from either color, depth or both images. Figure 6-3 shows sequences of hand image patches extracted using the salience based hand tracking algorithm.

6.2.1 Histogram of Oriented Gradients (HOG)

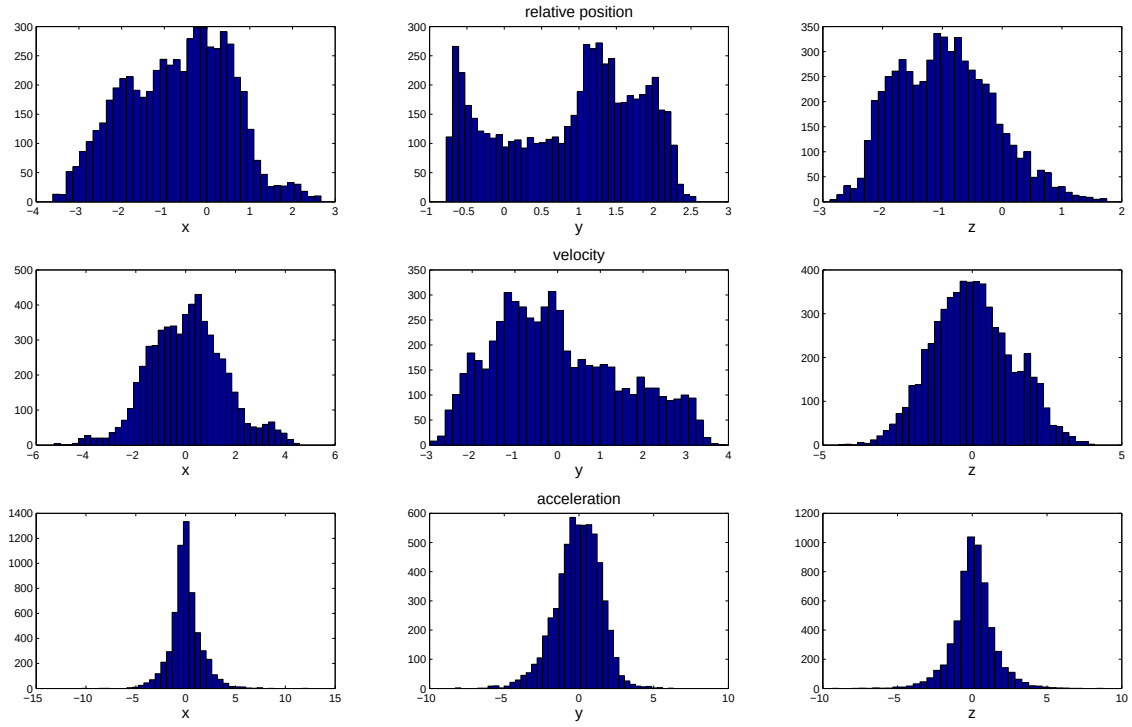
For each pixel, the magnitude (r) and the orientation (θ) of the gradient are

$$r = \sqrt{dx^2 + dy^2}$$
$$\theta = \arccos(dx/r)$$

where dx and dy are computed using a centered $[-1, 0, 1]$ derivative mask. Each pixel contributes a weighted vote (the magnitude of the gradient r) to an edge orientation histogram based on the orientation, θ, of the gradient element centered on it, and the votes are accumulated into orientation bins over local spatial regions called *cells*. The orientation bins are evenly spaced over 0°– 180°("unsigned" gradient). To reduce aliasing, votes are interpolated



(a) With rest positions.



(b) Without rest positions.

Figure 6-2: Histograms of motion features.

(a) Gray images converted from color images with only skin-colored pixels.

(b) Corresponding depth-mapped images.

Figure 6-3: 64×64px raw image patches of hands from the ChAirGest dataset.

bilinearly between the neighboring bin centers in both orientation and position [16].

Fine orientation and spatial binning turns out to be essential for good performance. My evaluation shows that using 4×4px cells (`cell_size` $= 4$) and 9 orientation bins gives the best result.

Finally, cell values are normalized using blocks of 2×2 cells (Figure 6-4). If an image I has dimensions $m \times n$, the size of the computed feature vector H is $(m/\texttt{cell\_size} - 1) \times (n/\texttt{cell\_size} - 1) \times \texttt{num\_bin}$. Figure 6-5 shows a visualization of the HOG descriptors from both the color images and depth-mapped images.

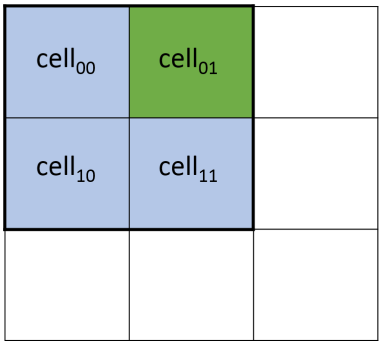
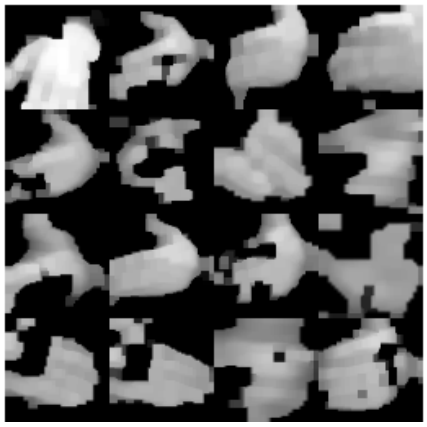
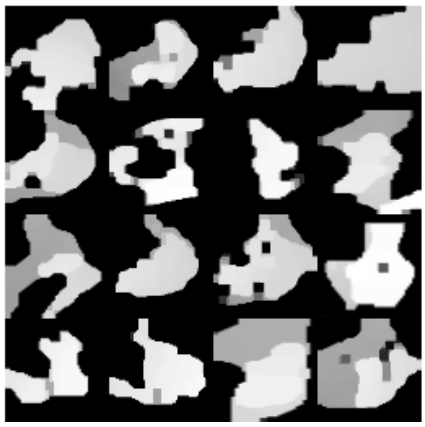


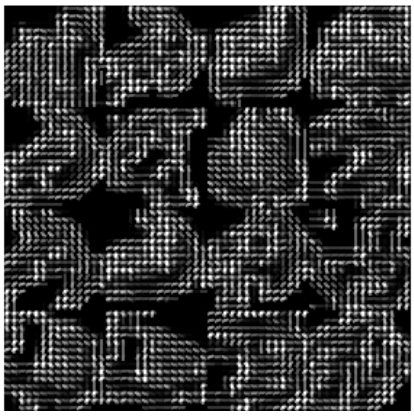
Figure 6-4: Histogram values in cell_{01} is normalized by the sum in cell_{00}, cell_{01}, cell_{10}, and cell_{11}.



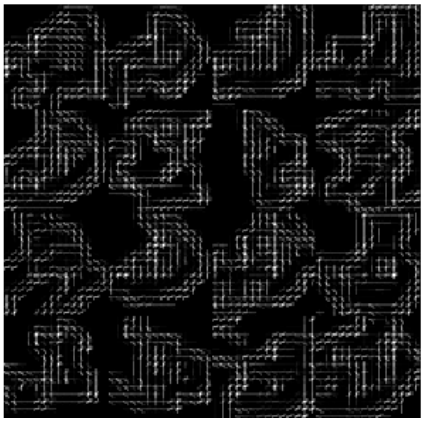
(a) Gray images converted from color images.



(b) Corresponding depth-mapped images.



(c) HOG from color images (converted to gray).



(d) HOG from depth-mapped images.

Figure 6-5: Visualization of HOG descriptors computed from 64×64px image patches.

6.2.2 Compare HOG from Color or Depth Images

Previous work [61] has used the HOG descriptor computed from color images for hand pose representation. With the addition of depth data, it is not obvious which data, depth or color, should be used for computing the HOG descriptor. Using the ChAirGest dataset and the same recognition method, I compared results between these two types of data. Table 6.2 shows that using HOG from both color and depth data gives the highest F_1 score. However, the improvement is small compared with using HOG computed from depth data only. As a result, in the real-time system, I only use the HOG computed from depth data to increase

processing speed.

	motion (no orientation)	**color**	**depth**	**color and depth**
F_1 Score	0.677 (0.04)	0.703 (0.01)	0.720 (0.02)	**0.723** (0.01)
ATSR Score	**0.893** (0.02)	0.870 (0.01)	0.880 (0.01)	0.873 (0.01)
Final Score	0.710 (0.03)	0.732 (0.01)	0.748 (0.01)	**0.749** (0.00)

Table 6.2: Comparison of the average 3-fold cross validation results for features computed from the Kinect sensor data using the ChAirGest dataset. Values in parentheses are standard deviations.

HOG from depth data may give better results than HOG from color data because depth data contains more information about the contour of the hand in one more dimension (see Figure 6-6).

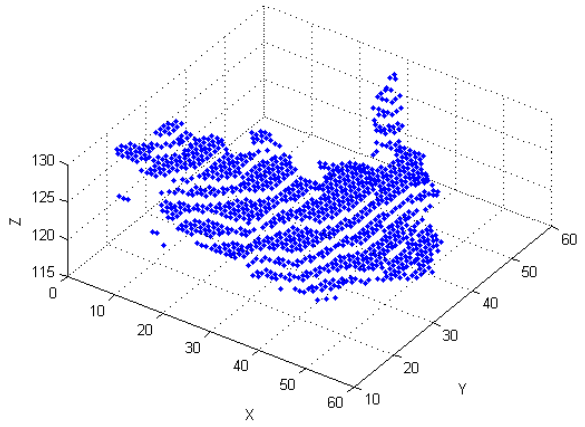


Figure 6-6: View of quantized depth data of a hand in 3D.

The confusion matrices in Figure 6-7 shows that "Take from Screen" and "Push to Screen" have the lowest recognition accuracy, especially when using the Kinect data. I notice that, in this dataset, when the users extend their hands towards the screen, the hands are too close to the sensor (below the too near range), and the depth sensor cannot get accurate readings.

Table 6.2 also indicates that if we use only motion features such as relative position, velocity and acceleration, the result is poor because it cannot distinguish gestures with the

same path but different hand postures such as "Wave Hello" and "Shake Hand", "Circle Palm Rotation" and "Circle Palm Down" (Figure 6-7(b)). However, using motion features only gives the highest temporal gesture nucleus segmentation score (the ATSR score). This means that for segmentation, motion features are probably more useful than the hand pose features.

6.3 Principal Component Analysis

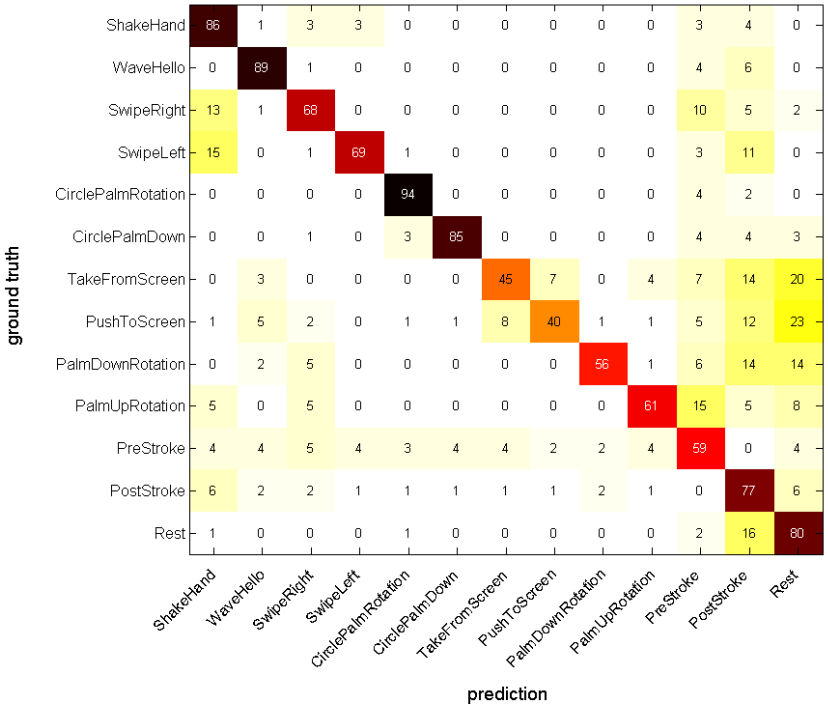
The dimension of the HOG feature descriptor can be large, therefore, I use Principal Component Analysis (PCA) [1] to reduce its dimensionality (see Appendix B for details). PCA can be seen as a feature encoder [52]. Since the principal components are orthogonal to each other, the encoded features are not correlated to each. As a result, the covariance matrix for these features is diagonal.

The number of principal components is determined through cross-validation. For the YANG dataset, I use 15 principal components from the HOG feature descriptor. Figure 8-2 shows the histograms of the 15 variables after projecting the original HOG descriptors onto the principal components and then applying standardization for one user's data from the YANG dataset. The distributions closely follow Gaussian or mixture of Gaussians, which again means we can model them using these probability distributions in the gesture model, same as the motion features.

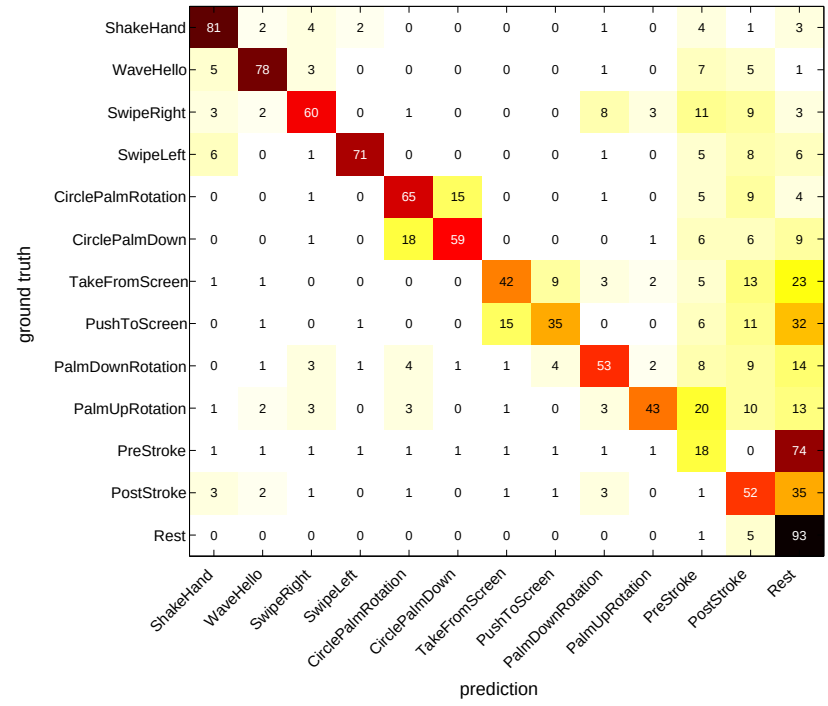
6.4 SVM for Encoding Hand Poses

Similar to Song et al. [61], I attempted to further encode the feature vectors obtained from PCA into hand pose classes. For each pose gesture, there is a class for its hand poses, and for all the path gestures, all of their hand poses are grouped into one class called "Other" (recall that hand pose is irrelevant in path gestures). Figure 6-9 shows hand pose images from two such classes.

I use an SVM with a Radial Basis Function (RBF) as the kernel function to do the



(a) Using HOG computed from both color and depth data.



(b) Using motion features only (relative position, velocity, acceleration)

Figure 6-7: Per frame classification confusion matrices based on result from 3-fold cross validation using the ChAirGest dataset. The numbers are percentages. The darker the color the higher the percentage.

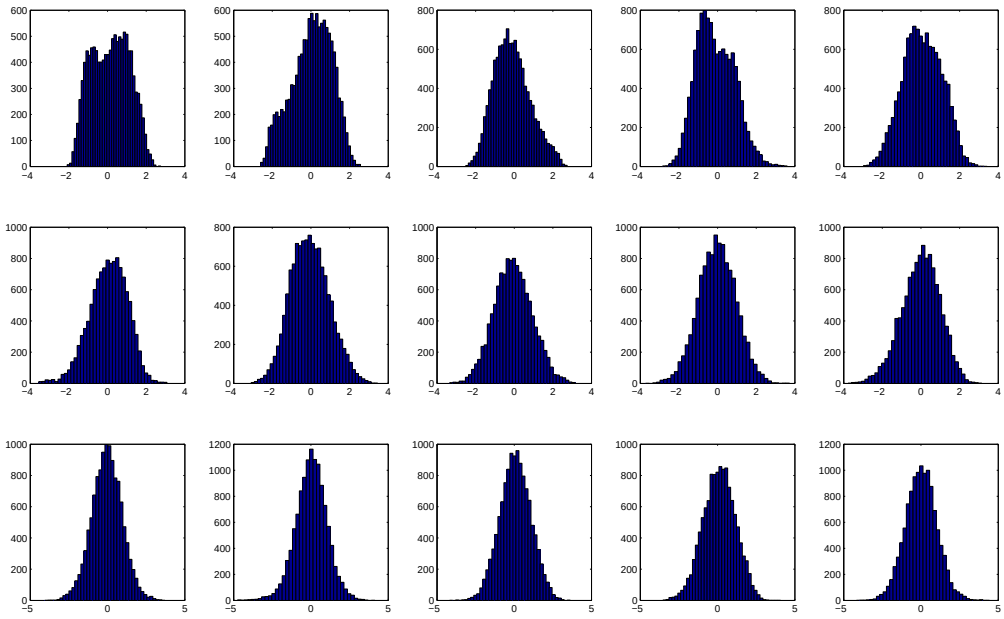


Figure 6-8: Histograms of the 15 components of in the feature vectors computed from apply PCA to the HOG descriptors.

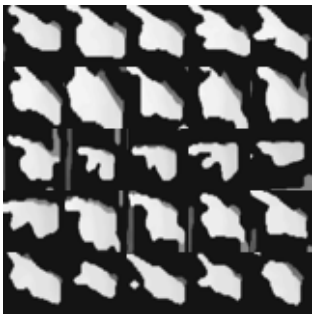
classification[1]. Grid search [73] on the training data was used to find the misclassification costs and γ in the RBF kernel function.

It is important to note that the data is highly unbalanced, for example, there are more instances in the "Other" class than other hand pose classes. If we ignore the fact that the data is unbalanced, the resultant classifier will favor the majority class [8]. To take this into account, I assign different misclassification costs, C_i, (SVM soft-margin constants) to each class i. If n_+ and n_- are the number of examples in a two -class problem, we choose C_+ and C_- such that:

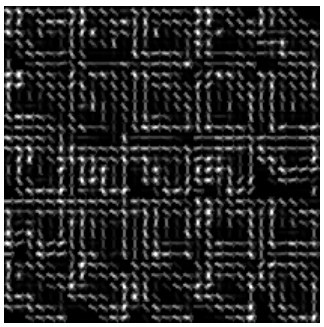
$$\frac{C_+}{C_-} = \frac{n_-}{n_+} \tag{6.1}$$

To generalize this to multi-class problems, if $n_1, n_2, \ldots, n_k$ are the number of examples in a

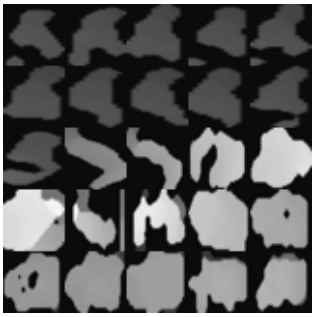
[1] LIBSVM (`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`) are used for SVM related computation.



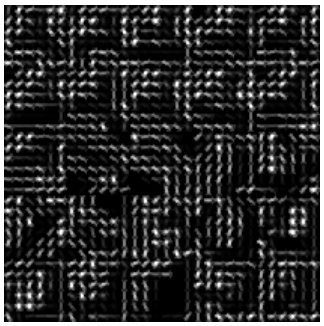
(a) Depth-mapped hand pose images from "POINT" class.



(b) Corresponding HOG descriptor.



(c) Depth-mapped hand pose images from "OTHER" class.



(d) Corresponding HOG descriptor.

Figure 6-9: Examples of hand poses from two classes.

k-class problem, we can choose $C_1, C_2, \ldots, C_k$ such as

$$C_1 : C_2 : \ldots : C_k = \frac{1}{n_1} : \frac{1}{n_2} : \ldots : \frac{1}{n_k} \tag{6.2}$$

Using data from one user in the YANG dataset, I compared hand pose classification results using SVM and my HMM-based gesture recognition algorithm with a mixture of Gaussians (MoG) as the emission probability. There are 5 classes in total: Point, Palm_Up, Grab, Rest, and Other. Table 6.3 shows that using HMM with MoG emission probability gives better result. Figure 6-10 shows a visualization of the classification results for a segment in a sequence. It shows that the HMM-based method has a smoothing effect which helps to improve performance.

	Precision	Recall	F_1
SVM	0.74	0.77	0.75
HMM with MoG emission	**0.81**	**0.82**	**0.81**

Table 6.3: Comparison of hand pose classification results.

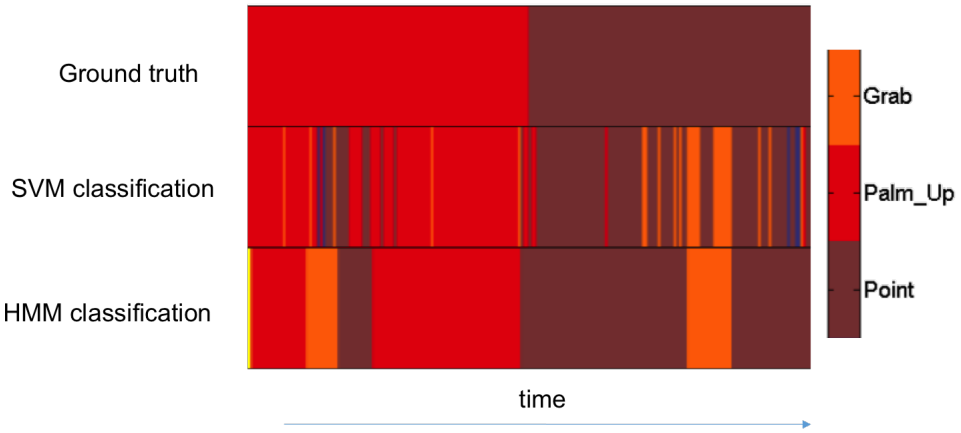


Figure 6-10: Visualization of the classification results comparing two methods. This is a continuous segment of about 200 frames (around 7s) in a sequence with two pose gestures: Palm Up and Point.

An SVM can output probabilities instead of hard classifications. It is plausible to use the probabilities as part of the feature vector (concatenated with the motion features and the HOG features). However, unlike the other features in the feature vector, the probabilities for a particular class from SVM does not follow Gaussian distribution (see Figure 6-11). This makes it hard to incorporate them into an HMM-based model which models compatibility between the output and the hidden state using a conditional probability distribution (CPD).

As a result, it does not seem to be effective to add SVM encoding (hard decisions or soft decisions) into the feature vectors.

6.5 Discussion

Based on the above analysis, the final feature vector is a concatenation of motion features and PCA encoded HOG descriptors. All of the features follow Gaussian or MoG distributions, making it easy to incorporate them in the HMM-based models.

The probability output from an SVM does not follow Gaussian distribution. However, it might be useful to use it as part of a feature vector for a CRF-based model as those models

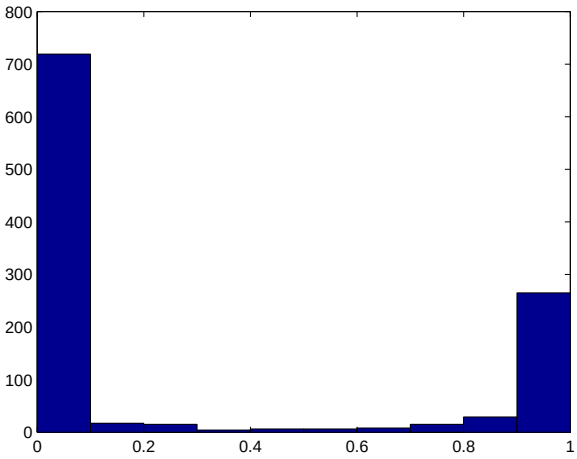


Figure 6-11: Histogram of SVM probability output for one class.

have greater flexibility to incorporate features (e.g. no CPD is required).

It is also useful to apply temporal smoothing on the motion data. In the final real-time gesture recognition system, I apply "box" smoothing (i.e., simple linear smoothing with equal weights) on the relative positions of the gesturing hand with a window size of 15 frames.

Apparently the child recognizes speech sounds as patterns of gestures.

Israel Rosenfield, *The invention of memory*

7

Unified Gesture Recognition Framework

As mentioned in the discussion of related work, most prior work focuses on recognizing one *form* of gesture, either path or pose gestures. We could conceivably just combine the two *forms* by first deciding whether it is a path or pose gesture (e.g., using a threshold or a classifier on speed and/or hand pose features), then apply different recognition methods. However making such decisions too early may not be robust. For example, if the system makes the decision wrongly, it will be hard to correct that later. We could also apply two different methods simultaneously, e.g., compute likelihood from HMMs for path gestures and compute SVM probability scores for pose gestures, but it is not clear how these probabilities can be compared for making final decisions.

As a result, I developed a unified probabilistic framework to handle the two *forms* of gestures so that probabilities are comparable. During online recognition, the system makes soft (probabilistic) decisions rather than hard (categorical) decisions, and propagates prob-

abilities until a response from the system is required according to the *flow* of the currently most likely gesture. This chapter gives details about this unified framework.

7.1 Gesture Modeling using Hierarchical HMM

In Section 1.1.3, I explained that a gesture can be broken down into three phases: *pre-stroke*, *nucleus*, and *post-stroke*. With multiple gestures, the temporal model can be represented by a stochastic state machine as shown by the black circles and arcs (top level) in Figure 7-1. Each gesture phase in turn includes a sequence of hand/arm movements that can also be represented by a stochastic state machine (the blue circles and arcs in the second level in the figure), with each state generating an observation (i.e., the feature vector) according to certain distribution. This generative model is a hierarchical HMM (HHMM).

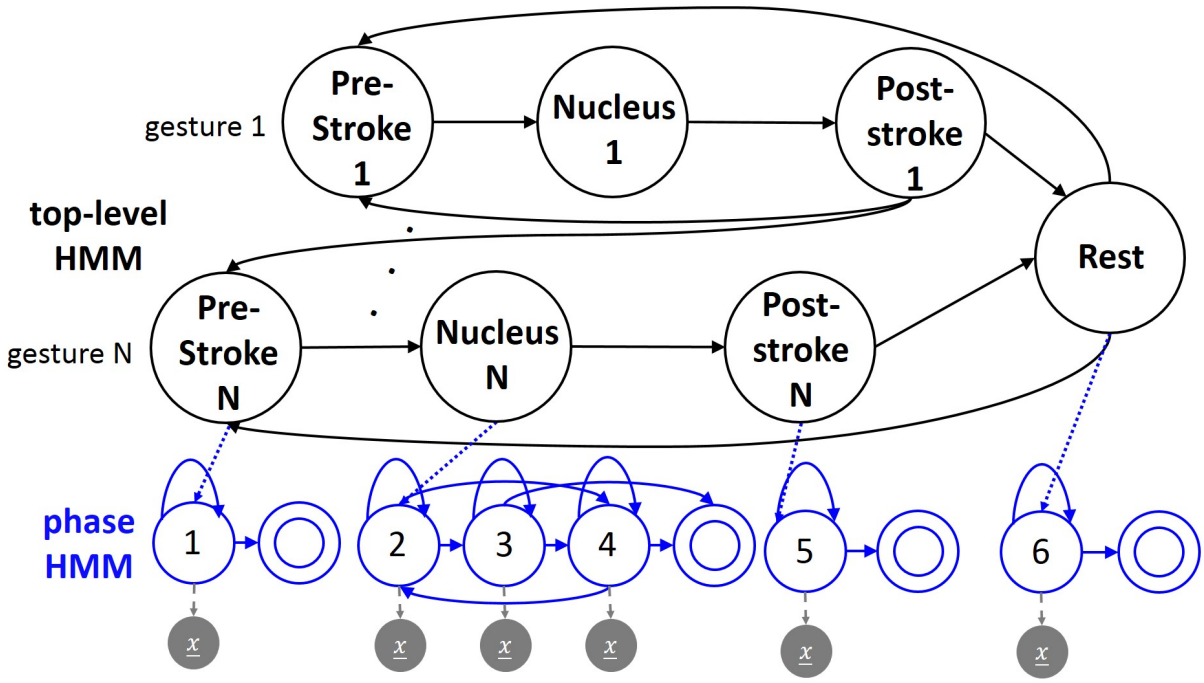


Figure 7-1: State transition diagram of the HHMM representation of the gesturing process. Solid arcs represent horizontal transitions between states; dotted arcs represent vertical transitions, i.e., calling a phase HMM (sub-HMM). Double-ringed states are end states. Only examples of transitions are shown here.

The HHMM is an extension of the HMM that is designed to model domains with hierarchical structure and/or dependencies at multiple length/time scales [42]. In an HHMM, the

states of the stochastic automaton can emit single observations or strings of observations. Those that emit single observations are called "production states" (blue circles in Figure 7-1, and those that emit sequences of observations are termed "abstract states" (black circles). The abstract states call sub-HMMs to which I refer as phase HMMs here.

We can represent the HHMM as a dynamic Bayesian network (DBN) (a directed graphical model) [42] as shown in Figure 7-2. The state of the whole HHMM at time t is encoded by the vector (G_t, P_t, S_t) where G_t is the gesture label, P_t is the phase label, and S_t is the hidden state representing a sub-stage in a gesture phase. F_t^d is a binary indicator variable that is "on" (has value 1) if the lower level HMM at time t has just "finished" (i.e., is about to enter an end state), otherwise it is "off" (value 0). The bottom level X_t are the observations.

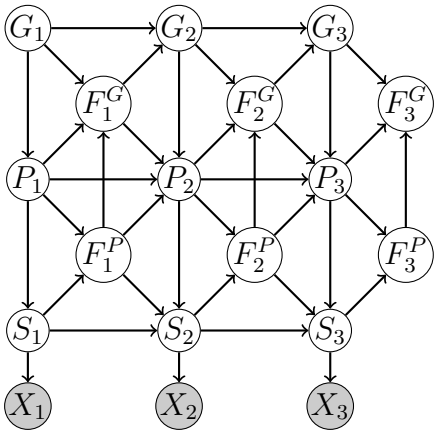


Figure 7-2: DBN representation of the HHMM for the temporal gesture model. G_t is the gesture label, P_t is the phase label, and S_t is the hidden state representing a sub-stage in a gesture phase. $F_t^d = 1$ if the HMM at the lower level has finished (entered its exit state), otherwise $F_t^d = 0$. Shaded nodes are observed; the remaining nodes are hidden.

The F_t^d binary indicator in the hierarchical model allows us to do simultaneous segmentation and recognition. I want to avoid doing segmentation first and then find the most likely HMM for the given sequence, because segmentation based on differentiating rest position versus non-rest position will not allow the system to respond fast enough. I want the system to respond at the beginning of the post-stroke phase rather then at the beginning of the rest position. In addition, making a hard decision on segmentation can introduce errors that are hard to correct later.

Even though the HHMM gives an appropriate model of temporal gestures, performing exact inference on it can be slow due to the loops in the graphical model (Figure 7-2). To

make possible a real-time system, I flatten the hierarchical HMM into a one-level HMM for fast training and inference. This is done by creating an HMM state for every leaf in the HHMM state transition diagram (i.e., every legal HHMM stack configuration) [42], assuming that the states in the phase-HMMs are not shared. The effect of the F_t^d binary indicator can be achieved by modeling the termination probability of each state, $t(\text{END}|s)$, i.e., the probability for state s to transit to the end state in the phase HMM.

When flattening the HHMM, we need to add to the flattened model all the transition probabilities among phase HMMs in the original hierarchal model. For example, after flattening the HHMM in Figure 7-1, we have (among others)

$$P_{flat}(5 \rightarrow 6) = P_h(5 \rightarrow \text{END} \rightarrow \text{Rest} \rightarrow 6)$$
$$P_{flat}(4 \rightarrow 5) = P_h(4 \rightarrow \text{END} \rightarrow \text{Post-stroke N} \rightarrow 5)$$

where P_h represents the probability in the HHMM.

Flattening the HHMM into an one-level HMM does have some disadvantages. First, flattening loses modularity, since the parameters of the phase HMMs get combined in a complex way [42]. A separate index is needed to map the hidden states back to the abstract states (the phase and gesture labels) they correspond to. Secondly, training HMMs separately and combining them together in one model requires segmented data, i.e., data about an individual gesture. However, getting separate training examples for different gestures is not a difficult task in this case. Lastly, an HHMM can represent and learn sub-models that are re-used in different contexts, but an HMM cannot do this [42].

7.2 Unified Framework

To include both path and pose gestures within the model, I use different topologies for their corresponding phase HMMs because they have different characteristics in terms of hand movement.

Based on the standard HMM[1] formulation, the probability of a sequence of observation,

[1]See Appendix D for more details about the HMM.

$\underline{x}_{1:T} = \underline{x}_1 \dots \underline{x}_T$, and the corresponding hidden states sequence, $s_{1:T} = s_1 \dots s_T$, is given by

$$P(\underline{x}_{1:T}, s_{1:T}; \underline{\theta}) = t(s_1)t(\text{END}|s_T) \prod_{t=2}^{T} t(s_t|s_{t-1}) \prod_{t=1}^{T} e(\underline{x}_t|s_t) \tag{7.1}$$

where $\underline{\theta}$ represents the model parameter vector which includes the initial state probabilities $t(s)$, the state transition probabilities $t(s'|s)$, the emission probabilities $e(\underline{x}|s)$, and the termination probabilities $t(\text{END}|s)$ for $s, s' \in \{1, 2, \dots H\}$. In this section, I describe how I model these conditional probability distributions (CPDs), and compute the model parameters for both path and pose gestures, combining them together under the unified HMM-based framework. Note that the term "unified framework" means that the two forms of gestures are combined under the same probabilistic framework, but there are still differences in the models for the two gestures, e.g., different topologies and different training strategies.

7.2.1 Path Gestures

If we have ground truth labels for the pre-stroke, the nucleus and the post-stroke phases (as we do in the ChAirGest dataset), we can train the phase HMMs for each phase and each gesture separately, then concatenate the phase HMMs (i.e., the concatenated HMMs[2]). However in practice, for example if we want users to be able to easily add their new gestures by giving a few examples, it will be tedious to manually label the start and the end of the three phases. In this case, I use embedded training [80], i.e. train each phase HMM embedded in an entire gesture segment (Figure 7-3).

With the YANG dataset, I choose to use one hidden state for pre-stroke and post-stroke phases through cross-validation. I use the Bakis (left-right with skips) topology [7] for the nucleus phase, but add a backward transition from the last hidden state to the first one for gestures with an arbitrary number of repetitions (e.g., the Wave gesture) (Figure 7-4). The left-right topology not only closely models the temporal evolution of a gesture, it also reduces the model complexity by constraining the number of possible transitions. Without this constraint, the model would have $O(H^2)$ transition parameters; with the left-right constraint, it only has $O(H)$ transition parameters where H is the number of hidden

[2]See my previous publication [78] for details.

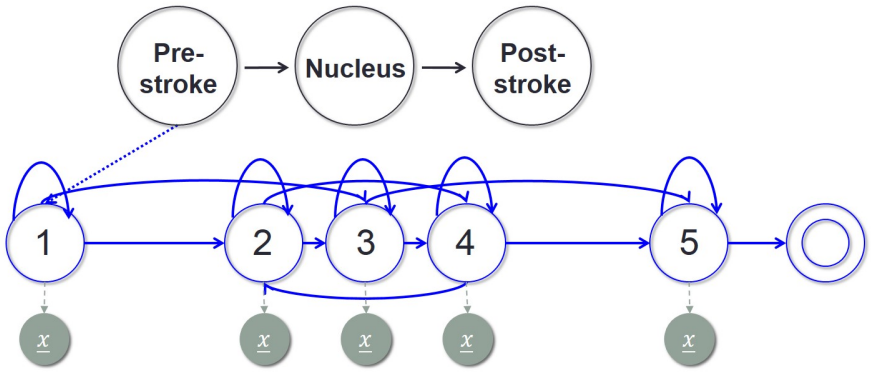


Figure 7-3: Embedding phase HMMs into an entire gesture.

states in the HMM.

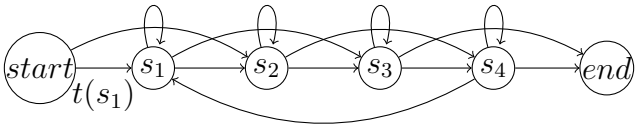


Figure 7-4: A state transition diagram of a modified 4-state Bakis model for the nucleus phase.

In Chapter 6, I showed that the features I use follow mixture of Gaussians (MoG) distributions, hence I use MoG with diagonal covariances to model the emission probability, i.e.,

$$e(\underline{x}|s) = \sum_{m=1}^{k} q(m|s)\mathcal{N}(\underline{x}; \mu_{s,m}, \Sigma_{s,m}) \tag{7.2}$$

where k is the number of mixtures. Figure 7-5 shows the DBN representation of an HMM of MoG emission probabilities. The covariance, $\Sigma_{s,m}$, has a fixed prior of $0.01 \times I$ which is added to the maximum likelihood estimate to prevent the covariance from shrinking to a point/delta function.

Training Strategies

I use two-pass training to estimate the model parameters for each path gesture HMM separately. Currently, training is done in MATLAB using Kevin Murphy's HMM toolbox[3].

[3] `https://code.google.com/p/bnt/`

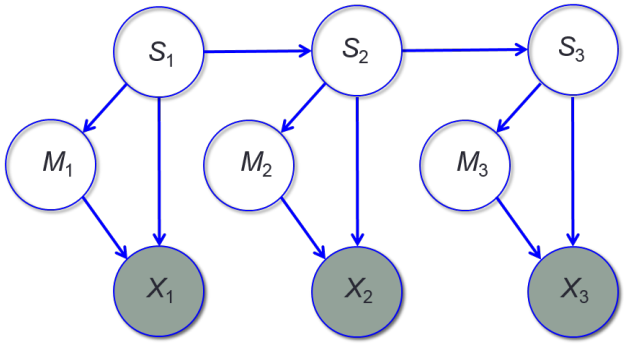


Figure 7-5: DBN representation of HMM with mixture of Gaussians emission probabilities.

The first pass is Viterbi training[4]. In this pass, only the pre-stroke hidden state can be the initial state and only the post-stroke hidden state can be the termination state. The emission probability parameters are initialized by dividing the data sequence into H segments of equal length (where H is the total number of hidden states in the embedded HMM) and associating each successive segment with successive states [80]. For each segment, the K-means algorithm is used to initialize MoG parameters. In each iteration of the training, the most likely path is computed using the Viterbi algorithm. For all the observations assigned to the same hidden state, K-means algorithm is used again to find the new MoG parameters for that state. The Viterbi training provides a better alignment of data with the hidden states, and thus gives a better initialization of the MoG parameters.

The second pass is Baum-Welch training that computes the maximum likelihood estimate of the parameters. It uses the estimated MoG parameters as the initial values. It also relaxes the initial state probabilities (allowing the first two states to be the start state) and the termination probabilities (allowing the last two states to be the termination state). This allows the possibility of transitions between nucleus phases without going through pre-strokes and post-strokes when we combine the individual gesture HMMs together. For natural interaction, this is important because sometimes users may do two gestures immediately after one another, with no post-stroke. When initializing the transition matrix, all the allowed transitions are set to the same value, so that no states are favored arbitrarily. Transitions that are not allowed are initialized to zero.

[4]See Appendix D.3.2 for details

The Baum-Welch estimation formulae for MoG parameters and transition parameters are well established. Here, I give the formula for the termination probability. Given N training sequences, the update for the termination probability during the ith iteration is

$$t^i(\text{END}|s) = \frac{\sum_{j=1}^{N} \overline{count}(j, s \rightarrow END; \underline{\theta}^{i-1})}{\sum_{j=1}^{N} \sum_{s'} \overline{count}(j, s \rightarrow s'; \underline{\theta}^{i-1})}$$

where $\overline{count}(j, s \rightarrow END; \underline{\theta}^{i-1})$ is the expected count of s being the end state. We can use the usual forward-backward algorithm to compute all the expected sufficient statistics by adding a dummy END state to the end of each sequence (see Appendix D.2 for details).

7.2.2 Pose Gestures

I use one hidden state, s_{pose}, to represent of the nucleus phase of a pose gesture (Figure 7-6). Within a user, there may be variation in the hand pose used for a particular pose gesture. For example, the Point hand pose can have different orientations. Hence, I also use the MoG for the emission probability $e(\underline{x}|s_{\text{pose}})$.

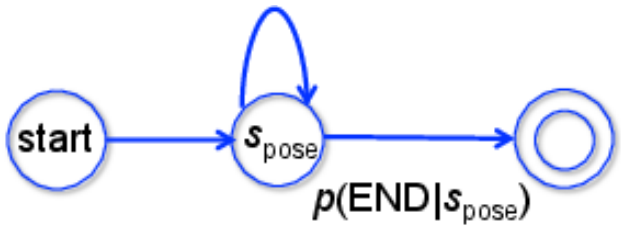


Figure 7-6: State transition diagram of a single state HMM for gestures with distinct hand poses.

The number of mixtures for pose gestures are greater than that of path gestures, made to capture the variation in hand poses people use. The number of mixtures can also be different for different pose gestures. This is because some hand poses may have more variation than others. For example, the Point gesture may have more variations in orientations than Grab gesture (hand in fist).

Training Strategies

Because there is only one hidden state, I directly compute the maximum likelihood estimates of the MoG parameters for emission probability of s_{pose} using an expectation–maximization (EM) algorithm, instead of doing embedded training.

The number of mixtures is determined using the Bayesian Information Criterion (BIC) [21]:

$$\text{BIC} \stackrel{\text{def}}{=\!=} 2\text{loglik}(\underline{x}_{1:n}, \underline{\theta}_k^*) - (\#\text{ params})\log(n)$$

where $\text{loglik}(\underline{x}_{1:n}, \underline{\theta}_k^*)$ is the maximized loglikelihood for the data and the model with k mixtures per state, (# params) is the number of independent parameters to be estimated in the model, and n is the number of observations in the data. Let d be the feature vector dimension, then $\mu_{s,m}$ for hidden state s and mixture m has d independent parameters, and the diagonal covariance matrix $\Sigma_{s,m}$ has d independent parameters as well. Because $\sum_{m=1}^{k} q(m|s) = 1$, the number of independent parameters for the weights $q(m|s)$ is $k-1$. Hence,(# params) is computed as:

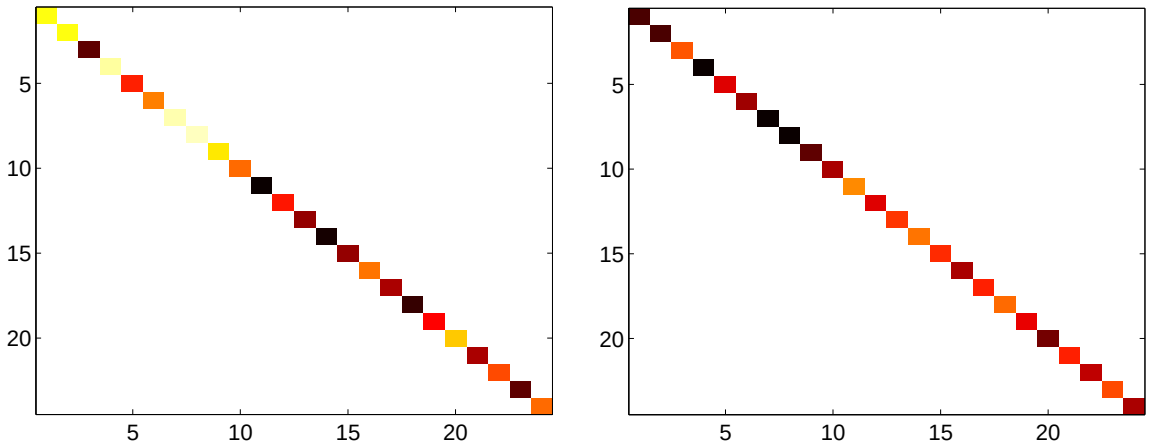
$$\#\text{ params} = (k-1) + (d+d) * k$$

The k that gives the highest BIC is chosen from a predetermined range (determined through cross-validation).

For each k, Expectation Maximization (EM) is used to estimate the means, covariance matrices and mixture probabilities of the MoG. As the EM algorithm may converge to a local optimum of the observed data likelihood [18], I repeat the optimization 3 times using K-means algorithm with random initialization to set the initial cluster centers, and choose the model that gives the maximum likelihood.

The training data for pose gestures should contain random variations in the hand movement so that the resultant variances for the motion features will be larger, making motion features less important in $e(\underline{x}|s_{\text{pose}})$. Figure 7-7 gives an illustration of this by comparing two covariance matrices of the MoG for hidden states from two *forms* of gestures. The variance values are normalized between the two matrices for each feature to make them comparable.

We can see that the variances for the motion features (features 1–9) are larger (with darker colors) for the pose gesture.



(a) Covariance of a MoG of a path gesture.

(b) Covariance of a MoG of a pose gesture.

Figure 7-7: Visualization of normalized covariance matrices of the MoG for different hidden states. The darker the color, the larger the variance.

It is possible that a gesture with a distinct path also has the same hand pose as another gesture with distinct hand pose, for instance, some user may prefer to do the Circle gesture with their hand in a point hand pose. In this case, the gesture will be recognized as the Circle gesture because the Circle gesture model matches both the hand pose and the path and thus will have a higher likelihood after considering a few consecutive frames.

Since there is only one hidden state for s_{pose}, its transition probability is 1. Its termination probability is estimated according to the expected duration of the gesture. The self-arc on a state in an HMM defines a geometric distribution over waiting time [42]. In the case of a single state HMM, the probability of remaining in state s_{pose} for exactly d steps is $P(d) = p(1-p)^{d-1}$, where $p = P(END|s_{\text{pose}})$ is the termination probability for s_{pose}. This means the expected number of steps remaining in state s_{pose} is $\frac{1}{p}$. I assume that the minimum duration of a path gesture is one second (30 frames). The termination probability $P(END|s_{\text{pose}})$ is then set to be less than $\frac{1}{30}$.

I use one hidden state to model the rest position in a similar way.

7.3 Real-time Gesture Recognition

I train one HMM, $\underline{\theta}_g$, for each gesture (path or pose) $g \in \{1 \dots G\}$, then combine them into a one-level HMM flattened from the HHMM in Figure 7-1, assuming uniform transition probabilities among gestures.

7.3.1 Combined HMM

I use a superscript c to denote the model parameters in the combined HMM. Let H_g be the number of hidden states for gesture g. The total number of hidden states in the combined HMM, H^c, is then

$$H^c = \sum_g H_g$$

The model for each path gesture starts with a hidden state for the pre-stroke, then 1 or 2 hidden states for the nucleus, followed by a hidden state for the post-stroke. Each pose gesture has a hidden state for the nucleus. The combined HMM has a sequential labeling for these hidden states, with the hidden state label for the pre-stroke of the second gesture model following the hidden state label for the post-stroke of the previous gesture, etc. Formally, the new hidden states labels are in $\{1 \dots H^c\}$, and the hidden states from $(\text{base}_i + 1)$ to $(\text{base}_g + H_g)$ belongs to gesture g where $base_g$ is the base index

$$\text{base}_g = \sum_{j=1}^{g-1} H_j$$

Figure 7-8 gives an example of the labeling scheme in the combined HMM. This scheme allows an easy mapping of hidden state labels back to their corresponding gestures and phases.

The non-trivial part in creating the combined HMM is computing the new transition probabilities. The initial state probability in the combined HMM, $t^c(s)$, for $s \in \{1 \dots H^c\}$ is

$$t^c(s) = \frac{t(s)}{\sum_{s'=1}^{H^c} t(s')}$$

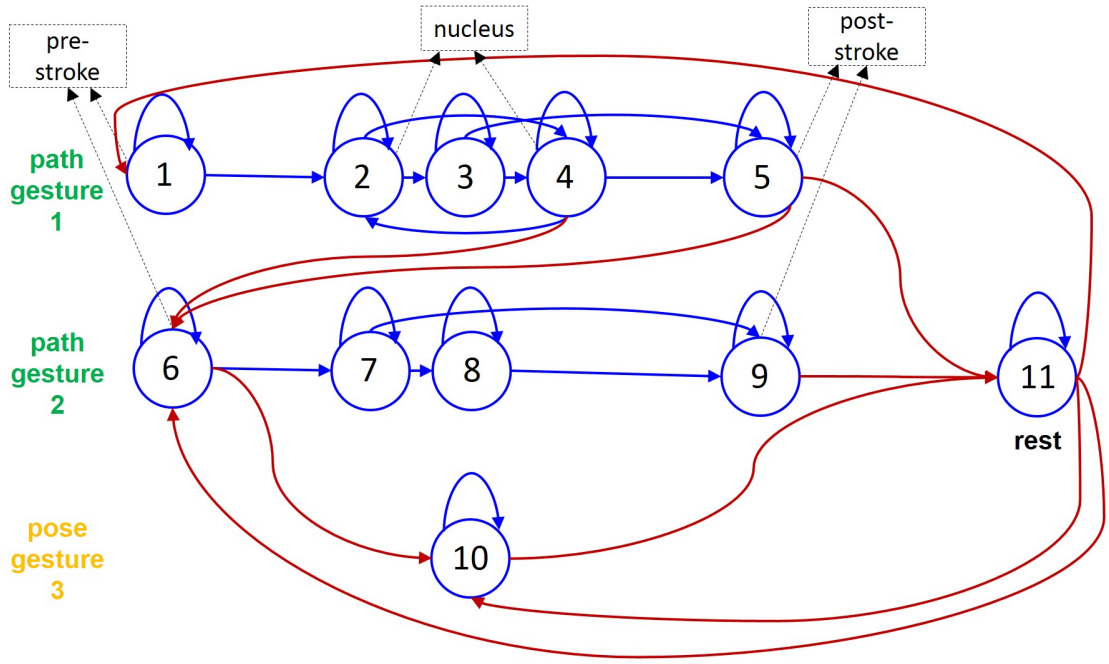


Figure 7-8: Combined HMM. The red lines are examples of transitions added to combine the individual gesture HMMs. To keep the diagram from getting cluttered, not all possible transitions are shown.

The probability of an s to s' transition in the combined model is the sum over all paths from s to s' for $s, s' \in \{1 \ldots H^c\}$ [42]. In our model,

$$t^c(s'|s) = t(s'|s)(1 - t(\text{END}|s)) + t(\text{END}|s)t^c(s') \tag{7.3}$$

where $t(s'|s)$ is the transition probability in the individual gesture HMM if s and s' are from the same gesture; otherwise it is zero. Note that $t(s'|s)$ is normalized such that $\sum_{s'} t(s'|s) = 1$ without taking into account the termination probability $t(\text{END}|s)$ (as the end state is not an actual hidden state), hence it is necessary to multiply $t(s'|s)$ by $(1-t(\text{END}|s))$ in Equation 7.3 to get the actual unnormalized probability.

As the pre-strokes and post-strokes for different gestures can be similar, I allow sharing of the pre-stroke hidden states and post-stroke hidden states among gestures by adding a small transition probability (0.01) prior among pre-stroke hidden states and among post-stroke hidden states (e.g., $t(s_{\text{pre-stroke}_i}|s_{\text{pre-stroke}_j}) = 0.01$). Pose gestures also share the pre-stroke and post-stroke hidden states of path gestures. The sharing of hidden states is similar to the

mechanism of parameter tying/sharing which is often used in speech recognition to improve the robustness of the model when training data is limited [80]. Note that these probabilities are not learned because the phase HMMs are trained separately and the inability to learn shared structures is a disadvantage of flattening the HHMM as mentioned earlier.

Finally, we need to make sure that the new transition matrix is stochastic, i.e.,

$$\sum_{s'=1}^{H^c} t^c(s'|s) = 1$$

by normalizing $t^c(s'|s)$.

7.3.2 Online Inference

Once we have a combined model, I use fixed-lag smoothing [42] to do online inference on the flattened HMM for real-time gesture recognition. Fixed-lag smoothing is a modified forward-backward algorithm. Unlike online filtering, which estimates the belief state at current time t using only a forward pass, we estimate the state at $t - l$, given all the evidence up to the current time t, i.e., compute $\gamma_{t-l}(s) \stackrel{\text{def}}{=\!=} P(S_{t-l} = s|\underline{x}_{1:t})$, where $l > 0$ is the lag (see Figure 7-9). Introducing lag time is a tradeoff between accuracy and responsiveness. Using some future evidence to smooth the estimate can increase the accuracy while adding some delay. However if the delay is small, it might be unnoticeable. In the Experiment Evaluation section (Section 8), I show details about the relationship between l and the recognition performance.

Fixed-lag smoothing can be implemented efficiently. I compute forward probabilities $\alpha_t(s) \stackrel{\text{def}}{=\!=} P(S_t = s|\underline{x}_{1:t})$ normally[5] and keep a history window of $\alpha_{t-l} \ldots \alpha_t$. At every time frame t, I compute backward probabilities $\beta_{t-l'}(s) \stackrel{\text{def}}{=\!=} P(\underline{x}_{t-l'+1:t}|S_{t-l'} = s)$ from the current time t to $t - l$, i.e., from $l' = 0$ to l. The base case is

$$\beta_t(s) = 1 \tag{7.4}$$

[5]It is more common (see e.g., [51]) to define $\alpha_t(s) = P(S_t = s, \underline{x}_{1:t})$; the difference is discussed in Appendix D.1.

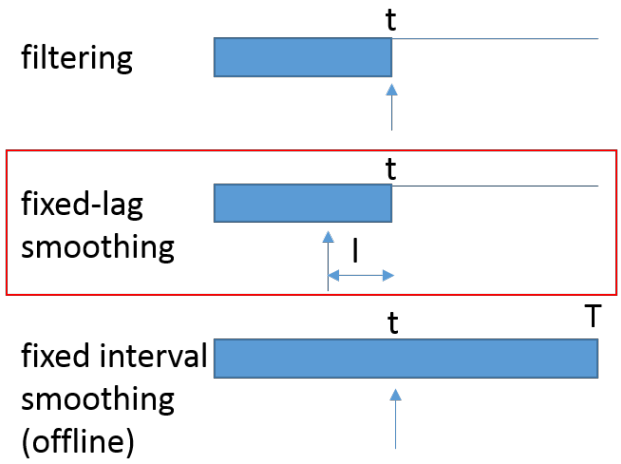


Figure 7-9: Figure adapted from [42] comparing different kinds of inference. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, and T is the sequence length (see Appendix C.2 for details).

Then γ can be computed as

$$\gamma_{t-l} \stackrel{\text{def}}{=\!=} P(S_{t-l} = s|\underline{x}_{1:t}) \propto \alpha_{t-l} \cdot \beta_{t-l} \tag{7.5}$$

The time complexity at each time frame is $O((H^c)^2 l)$. Note that at time t, the belief state at $t-l$ is committed, while the belief state from $t-l+1$ to t will still be revised later. The space complexity is $O(H^c l)$ mainly for storing a window of $\alpha_{t-1} \ldots \alpha_t$.

We can then compute the most likely hidden state at $t-l$:

$$\hat{s} = \arg\max_s \gamma_{t-l}(s) \tag{7.6}$$

The most likely hidden state is then mapped to the gesture label it belongs to (including the rest position) and the gesture phase.

Gesture events are detected at the boundary of a phase change: start pre-stroke, start gesture nucleus and start post-stroke. In this way we achieve simultaneous segmentation and recognition. The gesture event information, together with the gesture label for the nucleus phase, are sent to the application level. The system achieves real-time performance at 30FPS on a consumer level machine with a Intel Core2 Quad CPU (2.83GHz) and 8GB of memory.

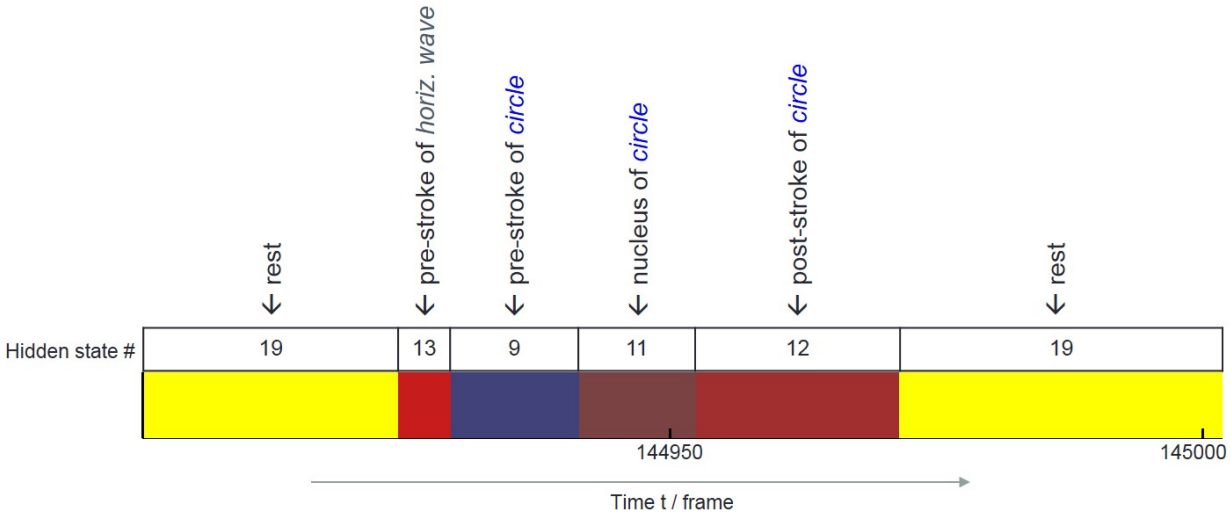


Figure 7-10: Most likely hidden states using fixed-lag smoothing from a segment of an input sequence. Different colors indicate different hidden states. Yellow indicates rest position.

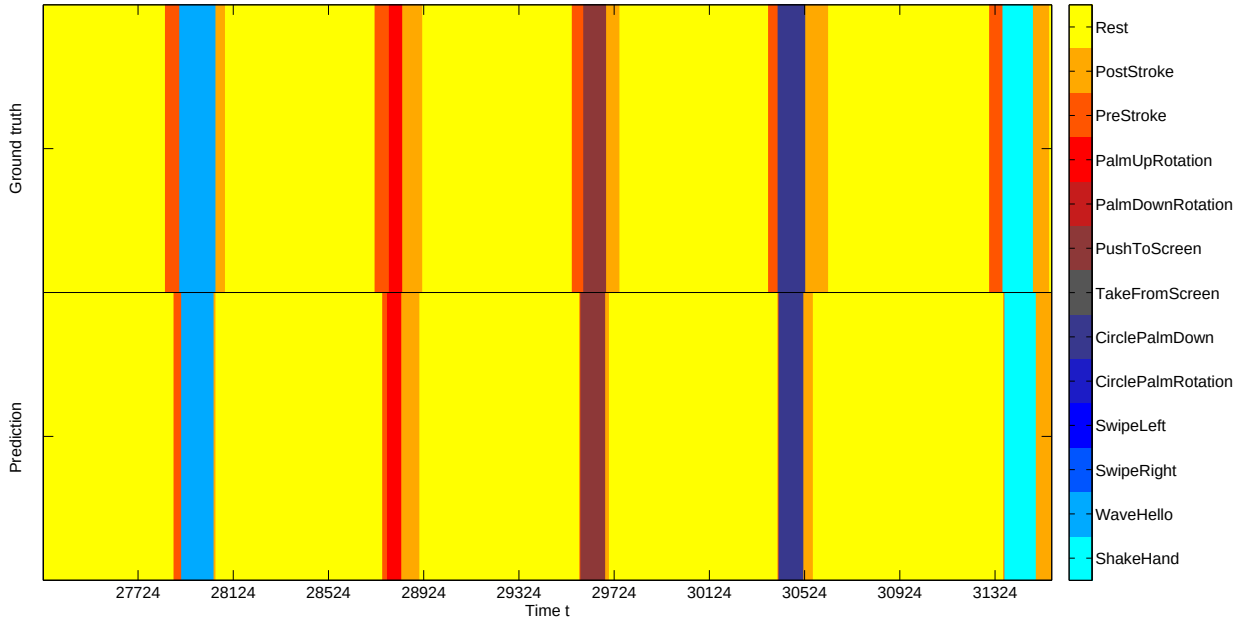
Figure 7-10 shows a visualization of the most likely hidden states based on online fixed-lag smoothing inference with $l = 5$ on a test sequence from the YANG dataset (only a segment is shown). Notice that at the beginning of the hand movement, the most likely hidden state is the pre-stroke for Horizontal Wave, but since a response is not required at this time, the wrong estimate does not matter. After a few more frames, the estimates are updated to have the correct most likely gesture label and the system responds correctly when it detects the start of the post-stroke of the Circle gesture.

7.4 Gesture Spotting

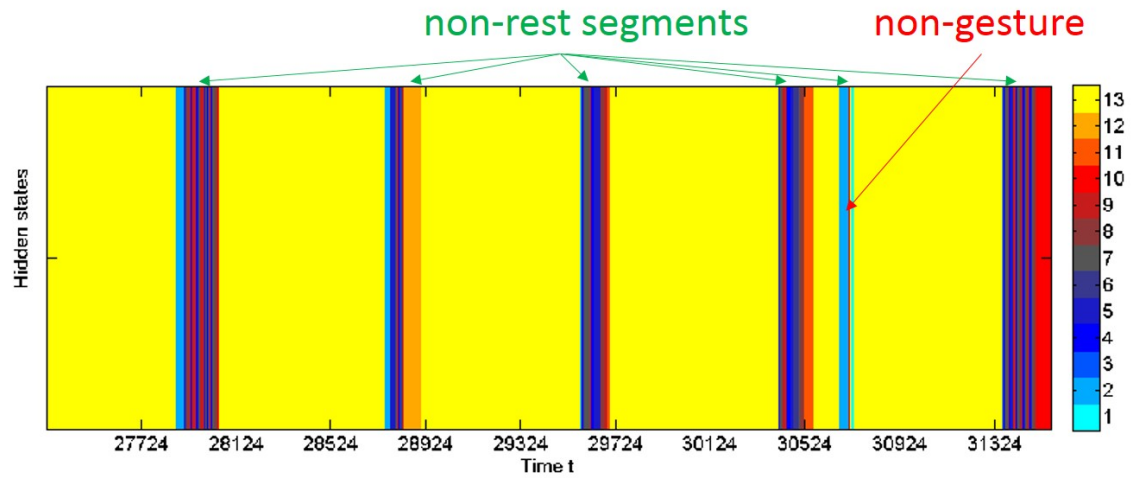
Identifying gesture phases allows us to distinguish gestures from non-gestures. As every gesture must have a nucleus, any hand movement that does not have a nucleus phase can be classified as a non-gesture.

For example, when performing offline recognition, I use MoG models for rest and non-rest positions to find non-rest segments (i.e., segments with hand movement), and for a non-rest segment, use the Viterbi algorithm to find the most probable hidden state sequence $\hat{s}_1 \dots \hat{s}_T$ using the mostly likely gesture model $\underline{\theta}_{\hat{g}}$ (see [78] for details). Again, each hidden state can by classified into a pre-stroke ($s_{\text{pre-stroke}}$), nucleus (s_{nucleus}), or post-stroke ($s_{\text{post-stroke}}$) state.

The start and the end time for a gesture nucleus are the first and the last time frame t where $\hat{s}_t \in s_{\text{nucleus}}$ respectively.



(a) Gesture label result. The pre-stroke and post-stroke phases are indicated by two orange colors (see the color bar).



(b) Most probable hidden states. Colors 1-3 indicate the pre-stroke hidden states, colors 4 - 9 indicate the nucleus hidden states, colors 10 - 12 indicate the post-stroke hidden states, and color 14 indicates the rest state.

Figure 7-11: Visualization of gesture recognition result. A non-rest segment without a nucleus phase (see $t \sim 30700$ in (b)) is not identified as a gesture (no label reported at the same time in (a).

Figure 7-11(a) shows the recognition result for one test sequence. The top row is the

ground truth, with different colors indicating different gesture phases or the rest position. The second row is my segmentation and recognition result. Figure 7-11(b) shows the color-coded most probable hidden states for the same sequence. If a non-rest segment does not contain hidden states belonging to the nucleus phase, it is ignored (see the blue bar at $t \sim 30700$ in Figure 7-11(b)). In this way, we can spot the actual gestures while filtering out other movements.

Using a thresholding method on the loglikelihood of a given segment may not be robust, because the maximum loglikelihood of a non-gesture segment among all gesture HMMs can be greater than that of a gesture segment. The HMM formulation (Equation 7.1) implies that the longer the sequence the smaller the likelihood (and hence the loglikelihood) because there are more probability terms which are all smaller than 1 in the product terms. For example, the loglikelihood of the fifth non-rest segment (a non-gesture) in Figure 7-11(b) has a maximum loglikelihood of -296.6, while the first non-rest segment (a gesture) has a maximum loglikelihood of -785.6. Even if we normalize the loglikelihoods by the segment lengths, the normalized loglikelihood for the non-gesture (-14.8) is still greater than that of the gesture (-17.9). As non-gestures are often shorter than gesture sequences, it would hard to set a good threshold.

My gesture phase based method can be used in addition to a thresholding method with a relative conservative threshold, i.e., a threshold that may cause false positives but no false negatives so that the gesture phase based method can be used to further filter out false positives.

7.5 Concatenated HMM versus LDCRF

Morency et al. formulated LDCRF [41] (an extension to CRF), and used it for head and eye gesture recognition. In contrast to HMM (a generative model), LDCRF is a discriminative model. It gives per frame classification, and hence, can be used to do simultaneous segmentation and labeling. I applied LDCRF to gesture phase segmentation and recognition, and compared the result with my concatenated HMM method [78].

As LDCRF requires per frame labeled training data, we need ground truth labeling for

the gesture phases. Hence, I use the ChAirGest dataset for this evaluation. For all the non-rest segments in the training set, the nucleus frames are labeled 1 – 10 according to their corresponding gesture labels. All the pre-stroke frames are labeled 11, and all the post-stroke frames are labeled 12. This means the pre-strokes of all the gestures share the same hidden states, and same for post-strokes. Using different hidden states for pre-strokes and post-strokes of different gestures would increase computation time significantly since it increases quadratically with the number of hidden states (see Appendix E.1). For each label, 6 hidden states are used, resulting in 72 hidden states in total.

During testing, frames from the non-rest segments are classified into gesture nucleus labels, pre-stroke or post-stroke using the trained LDCRF model. Using the pre-stroke and post-stroke labels, we can find the start and end of the nucleus phases. The HCRF library[6] which contains an implementation of the LDCRF model is used for both training and testing.

Table 7.1 shows a comparison of the results between using LDCRF and concatenated HMMs. The F_1 score of the concatenated HMMs model is 7.7 percentage points higher than the LDCRF model. However, the LDCRF gives a slightly higher (1.5%) temporal segmentation score, ATSR. Overall, the concatenated HMMs gives better performance (6.0% higher) and, importantly, takes 150 times shorter time to train (two orders of magnitude).

	LDCRF	**Concatenated HMMs**
F_1 Score	0.82 (0.03)	**0.897 (0.03)**
ATSR Score	**0.923 (0.02)**	0.907 (0.02)
Final Score	0.84 (0.02)	**0.90 (0.02)**
Training Time	18hr	**7min**

Table 7.1: Comparison of recognition results between LDCRF and concatenated HMMs using the ChAirGest dataset.

7.5.1 LDCRF Optimization

One reason that LDCRF is so slow is that it considers all pair-wise transitions among all the hidden states in the optimization process (72^2 transition features in this case). However, since the hidden states are not shared among gestures, we do not need to consider the

[6] `http://sourceforge.net/projects/hcrf/`

transitions between hidden states from different gestures. In order to decrease the training time, I reduce the time complexity of the model by constraining allowed transitions among hidden states.

In HMM, we can specify the topology by initializing transitions that are not allowed to zero. In LDCRF, I do the same by setting transitions that are not allowed to always be zero (or negative infinity if loglikelihood is used) in each update. The edge features only include allowed transitions as well, reducing the number of features in the model needs to be optimized. A mask matrix is used to specify allowed transitions, and I call this masked LDCRF. Using this optimization, I am able to reduce the training time to 11hr, but this still means that LDCRF takes 100 times longer to train.

8

Online Recognition Evaluation

The previous sections reported evaluation results most pertinent to the sections under discussion. This chapter presents results for the overall online recognition performance.

8.1 Evaluation Protocol

I evaluate the online gesture recognition performance using the YANG dataset and the hybrid performance metrics proposed in Section 4.3. The evaluation is based on the assumption that all the path gestures are discrete flow gestures, and pose gestures are continuous flow gestures. The survey results shown earlier in Section 3.2.4 show that it is important to allow users to quickly define and train their own gestures. Hence, I evaluate my system using user dependent training and testing. For each user in the dataset, I use the first 2 sessions of recording (6 samples per gesture) as training examples, and the last 2 sessions as testing

examples. The first 2 sessions have "Rest" prompts which help to do automatic segmentation on the training data. All the results reported are the average test results from 10 users.

Figure 8-1 shows a visualization of the recognition result on a test sequence. The figure highlights the challenges in the test sequences: there are 21 gestures in each continuous unsegmented sequence; gestures sometimes follow one another immediately.

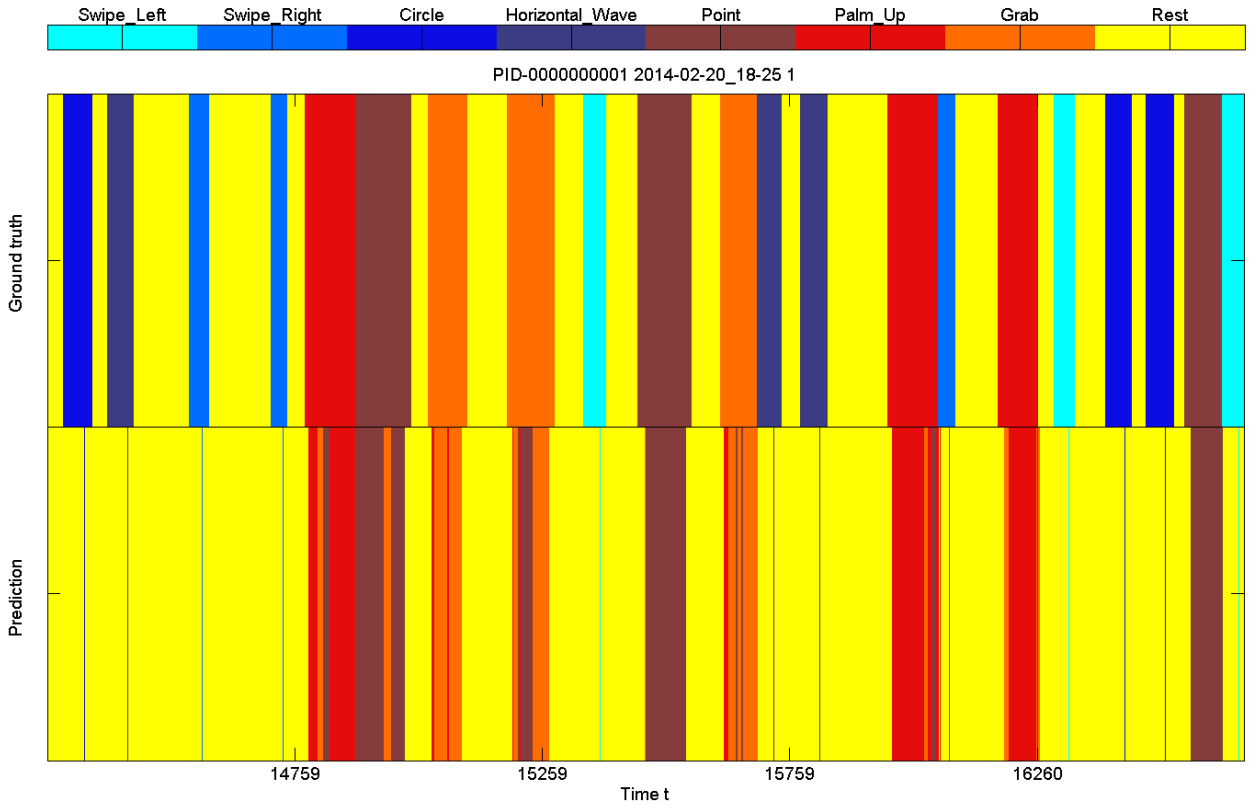


Figure 8-1: Comparison between recognition result using online inference and ground truth. The colors correspond to different gestures. For discrete flow gestures (Swipe Left/Right, Circle, Horizontal Wave), one color segment with a fixed length is shown at the time of response. For continuous flow gestures, the recognized gesture is shown at each frame indicating frame-by-frame responses.

8.2 Effect of the Number of Principal Components

PCA reduces the dimensionality of the feature vectors, and hence, reduces the computational complexity. I use cross-validation to find the best number of principal components that accounts for the variation in the data while keeping the dimension at the minimum. Figure 8-

2 shows how the recognition F_1 scores depends on the number of principal components used for the HOG descriptor. The best average score is obtained with 15 principal components.

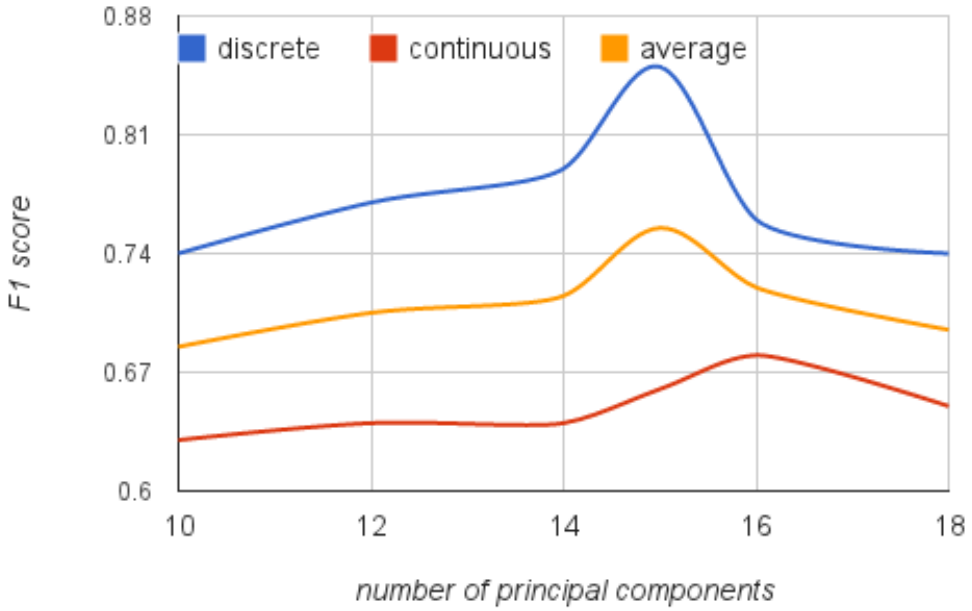


Figure 8-2: Graph showing how F_1 scores for discrete flow gestures, continuous flow gestures and the average scores change with the number of principal components used for the HOG descriptor.

8.3 Compare Different Topologies

In this section, I compare my unified framework with a common HMM-based approach used in previous works [55, 63], which use the same topology for all gestures.

Table 8.1 compares the results between the two methods. The third column is the result from treating the two *forms* of gestures in the same way, i.e., all gestures have the same left-right Bakis topology for their nucleus phases. The fourth column is the result from using a left-right Bakis topology for path gestures and a single state topology for pose gestures. To ensure a fair comparison, all hidden states have 3 mixtures of Gaussians. As Table 8.1 shows, having different HMM topologies for the two *forms* of gestures significantly increases

the recall and F_1 score for pose gestures.

		Same topology for two forms of gestures	**Different topologies for two forms of gestures**
Path & discrete flow gestures	Precision	**0.84 (0.16)**	0.82 (0.16)
	Recall	0.87 (0.17)	0.87 (0.18)
	F_1	**0.85 (0.16)**	0.84 (0.16)
	Responsiveness (s)	0.6 (0.3)	0.6 (0.3)
Pose & continuous flow gestures	Precision	**0.65 (0.14)**	0.63 (0.11)
	Recall	0.41 (0.15)	**0.65 (0.09)**
	F_1	0.50(0.14)	**0.64 (0.10)**
Average	F_1	0.675 (0.150)	**0.740 (0.130)**

Table 8.1: Results from using different topologies. The numbers in parentheses are standard deviations. The results are based on using 3 mixtures of Gaussians for all hidden states, and lag time $l = 8$ frames.

For gestures with arbitrary movement (e.g., pose gestures), it is difficult to use embedded training to accurately align pre-stroke, nucleus and post-stroke phases, because training sequences can have very different lengths, and so do the testing sequences. Figure 8-3 shows the estimates of the most likely hidden states for a Palm Up gesture sequence when the same left-right topology are used for both *forms* of gestures is used. The main (center) part of the gesture, which should be the nucleus of the gesture, is identified as the post-stroke. This is why it is important to have different topologies and different training strategies for the two forms of gestures.

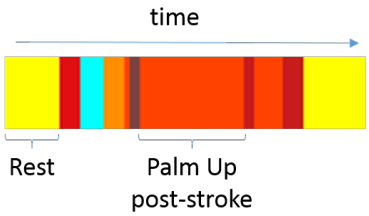


Figure 8-3: Estimated hidden states for a Palm Up gesture using the left-right model the same as path gestures. Different colors correspond to different hidden states.

8.4 Effect of Different Numbers of Mixtures

The previous section shows that using different topologies for path and pose gestures give better recognition performance. So using this model, I investigate the effects of the number of mixtures (k) in the MoG emission probabilities.

First, I set k be the same for both forms of gestures. Fig. 8-4 shows that the F_1 score increases as the number of mixtures increases until $k = 3$. After that, we start to see the effect of overfitting.

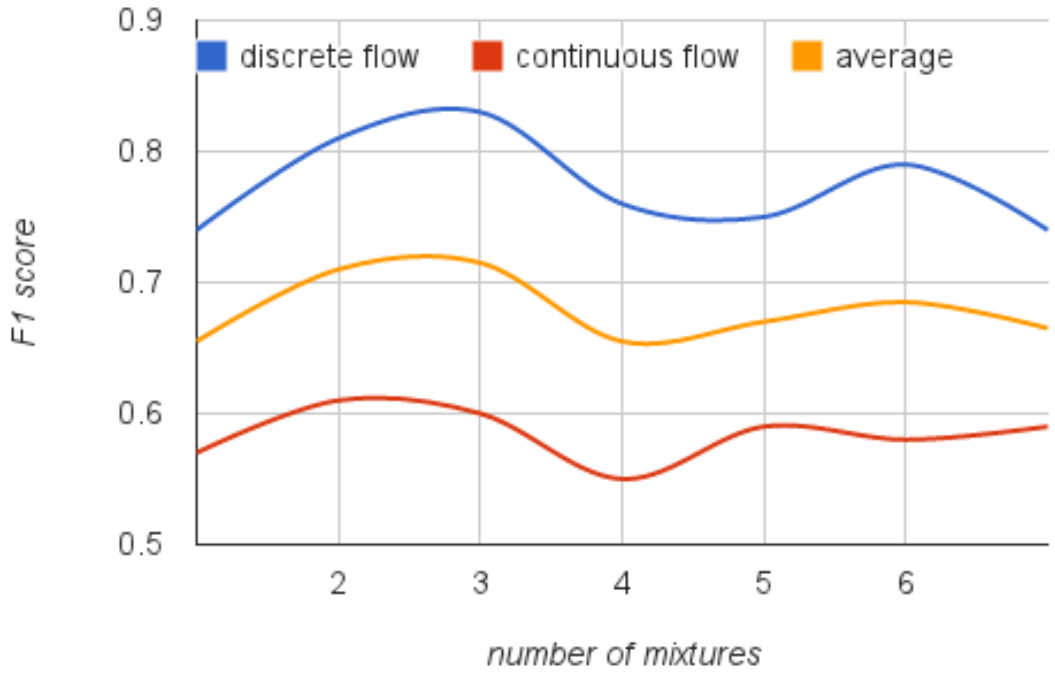


Figure 8-4: F1 scores versus number of mixtures.

I then experimented with using different k's for path and pose gestures. For path gestures, I set $k^{\text{path}} = 3$, and use a different number of mixtures, $k_g^{\text{pose}} \in \{3 \ldots 6\}$, for each pose gesture g. Each k_g^{pose} is chosen according to the Bayesian Information Criterion (see Section 7.2.2). Using this method, I am able to improve precision, recall and F_1 scores for both forms of the gestures even further (see Table 8.2).

		Use different topologies and numbers of mixtures
Path & discrete flow gestures	Precision	**0.94 (0.05)**
	Recall	**0.93 (0.08)**
	F_1	**0.93 (0.06)**
	Responsiveness (s)	0.6 (0.2)
Pose & continuous flow gestures	Precision	**0.68 (0.10)**
	Recall	**0.69 (0.08)**
	F_1	**0.68 (0.09)**
Average	F_1	**0.805 (0.075)**

Table 8.2: Results from using different numbers of mixtures of Gaussians for the emission probabilities ($l = 8$ frames).

8.5 Effect of Different Lag Times

Figure 8-5 shows how the F_1 scores change with respect to the lag time (l) in fixed-lag smoothing. The performance increases as l increases, and plateaus at $l = 8$ frames which is about 0.3s at 30 FPS. This shows that more evidence does help to improve the estimates until a limit, and we do not need to sacrifice too much delay to reach the limit.

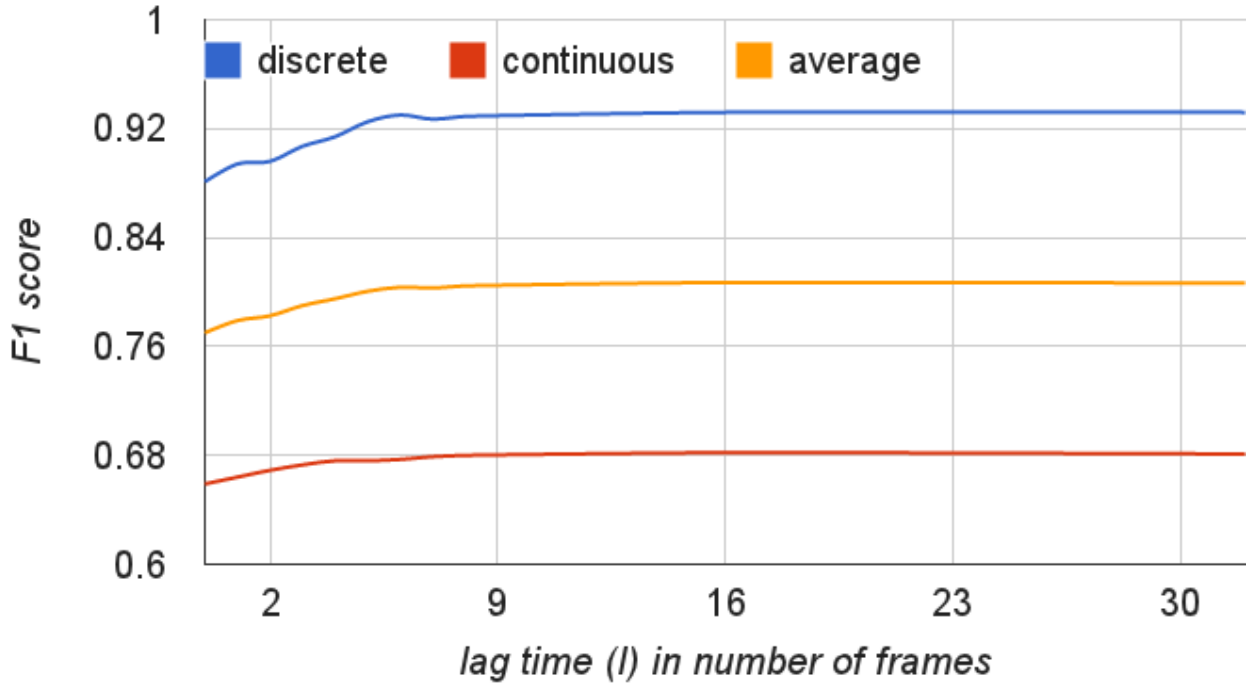


Figure 8-5: F_1 score versus lag time l.

8.6 Training Time

The HMM-based unified framework is fast to train. The average computation time for training the model for one user is about 5s with 7 gestures and 6 training examples per gesture.

Because of the fast training process, the system is easily extensible. New gestures can be added by recording 3-6 repetitions of the gesture using the Kinect; the system will train an HMM model for the gesture and add it to the existing combined HMM. This process, including the recording time, takes only a few minutes.

8.7 User Independent Evaluation

Our survey (Section 3.2.4) indicated that users generally prefer a system to have a predefined gesture set and then add or change gestures later according to their own preferences. This means that having a user independent base model will be useful as well. If the system has user provided training examples for certain gestures, it uses the user dependent models for those gestures; otherwise it backs off to the base model. This adaptation strategy is similar to [79].

To evaluate the user independent model, I did a 10-fold cross-validation where in each fold, the data from one user's last 2 sessions are used for testing and the data from the remaining 9 users' first 2 sessions are used for training. Table 8.3 shows the average, best and worst results among the users. We can see that there is large variation among the users. Users who want to do the gestures differently need to train their user dependent models. For example, the results based on user dependent model for user PID-02 is much better than the user independent model.

8.8 Discussion

Our overall best performance for the YANG dataset is reported in Table 8.2. The performance for the pose and continuous flow gestures is relatively low compared to that of the path and discrete flow gestures. Most confusions are among the 3 pose gestures, i.e., Point,

		Average	Best (PID-01)	Worst (PID-02)	User dep. (PID-02)
Path & discrete flow gestures	Precision	0.72 (0.18)	1.00	0.35	0.96
	Recall	0.72 (0.16)	1.00	0.72	1.00
	F_1	0.70 (0.16)	1.00	0.47	0.98
	Responsiveness (s)	0.6 (0.3)	0.6	0.4	0.4
Pose & continuous flow gestures	Precision	0.59 (0.12)	0.79	0.49	0.67
	Recall	0.56 (0.16)	0.77	0.21	0.67
	F_1	0.56 (0.13)	0.78	0.29	0.67
Average	F_1	0.63 (0.15)	0.89	0.38	0.83

Table 8.3: User independent model 10-fold cross validation results ($l = 8$ frames). The last column is the user dependent result for user PID-02 for comparison.

Palm Up, Grab (see Figure 8-6). There are big variations in the recognition results for the pose gestures among users: the highest F_1 score for pose gestures is 0.81 and the lowest is 0.58. For the ones with low F_1 scores, confusion occurs when there is significant motion blur (see Figure 8-7). These correspond to the users who move relatively fast when doing the pose gestures. This suggests that with the frame rate of the Kinect sensor used (30Hz for the Kinect version One), users may need to perform pose gestures relatively slowly in order to get better recognition results, or a better (higher rate) sensor is needed.

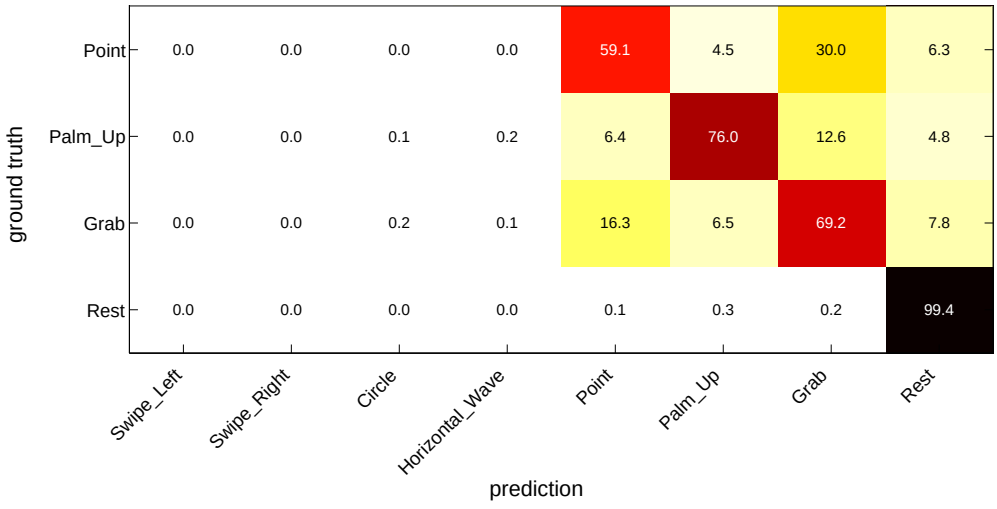


Figure 8-6: Confusion matrix for pose gestures.

Figure 8-7: This frame is mistakenly classified as Grab while the true gesture is Point. Motion blur is significant.

Each gesture is created at the moment of speaking and highlights what is relevant and the same entity can be referred to by gestures that have changed their form.

David McNeill, *Hand and mind: what gestures reveal about thought*

9 Gestural Interaction

This chapter discusses application level considerations for natural multimodal interaction.

9.1 Client Server Architecture

I developed a multimodal input recognition engine that consists of a gesture recognition engine and a speech recognition engine. The gesture recognition engine includes the hand tracking, feature extraction and gesture recognition modules described in the previous chapters. For speech recognition, I used the application programming interface (API) from the Windows for Kinect SDK[1].

The multimodal input recognition engine runs as a server sending recognized gesture and speech events over a WebSocket[2]. Any client application can subscribe to the input events

[1] `http://msdn.microsoft.com/en-us/library/jj131035.aspx`
[2] http://en.wikipedia.org/wiki/WebSocket

by connecting to the server socket. There are two types of events: gesture and speech, and they are serialized in JSON format[3] (Listing 9.1 and 9.2).

Listing 9.1: Gesture event JSON object

```
{"gesture": <gestureName>,
 "eventType": <"StartPreStroke" | "StartNucleus" | "StartPostStroke">,
 "phase": <"PreStroke" | "Nucleus" | "PostStroke">,
 "rightX": <rightHandXWorldCoordinate>,
 "rightY": <rightHandYWorldCoordinate> }
```

Listing 9.2: Speech event JSON object

```
{"speech": <speechTag>}
```

9.2 Gesture Controlled Presentation

To demonstrate the use of multimodal input recognition system in a real life application, I developed a gesture controlled presentation application[4], acting as a client. The application is HTML-based and uses the `reveal.js` framework[5]. The framework's API allows me to control the presentation directly (e.g., changing slides, showing an overview, pausing the slide show) instead of controlling it by binding to mouse events (as is commonly done in many previous demonstrations of gesture control). There are only 2 or 3 buttons on a mouse, and its functionality is confined to clicking and dragging. By mimicking mouse events, one limits the full potential of gesture input interface, because our hands have five fingers and are much more versatile than doing just clicking and dragging.

9.2.1 Handling Different Categories of Gestures

This application demonstrates the use of path and pose gestures and shows how the system respond differently for discrete *flow* and continuous *flow* gestures. I map the gestures in the YANG dataset to presentation control actions (Table 9.1).

[3] `http://www.json.org/`
[4] See `http://groups.csail.mit.edu/mug/projects/gesture_kinect/` for demo videos.
[5] `http://lab.hakim.se/reveal-js/`

Gesture	Action
Swipe Left	Next slide (DF)
Swipe Right	Previous slide (DF)
Circle	Toggle overview (show all slides) (DF)
Horizontal Wave	Toggle pause (turn screen black) (DF)
Point	Show a round cursor corresponding to hand position (CF)
Palm Up	Show a square cursor and seek video forward or backward (CF)

Table 9.1: Mapping from gestures to presentation control actions. DF stands for discrete *flow* gesture and CF stands for continuous *flow* gesture.

An interface should reflect the system's current state of understanding of the users' actions to help users develop the right mental model for interaction. For discrete *flow* gestures, the interface needs to respond at the StartPostStroke event. This responsiveness of the system helps users understand the relation between their actions and the system's responses.

For continuous *flow* gestures, the interface needs to show visual feedback of frame by frame changes corresponding to certain hand parameters. In addition, as pose gestures have distinct poses but arbitrary movement, it is important to have different visualizations show the system's understanding of the different poses. Hence, for a Point gesture, the system shows a round cursor moving according to the user's hand position (Figure 9-2), and for a Palm Up gesture, it shows a square cursor (Figure 9-1). Listing 9.3 shows the corresponding client code in CoffeeScript.

Listing 9.3: Client code mapping gesture events to actions in CoffeeScript.

```
switch ge.eventType
  when 'StartPostStroke'
    switch ge.gesture
      when 'Swipe_Left'
        if @_config.mirror then Reveal.right() else Reveal.left()
      when 'Swipe_Right'
        if @_config.mirror then Reveal.left() else Reveal.right()
      when 'Circle' then Reveal.toggleOverview()
      when 'Horizontal_Wave' then Reveal.togglePause()
  else
    switch ge.gesture
      when 'Point' then @_view.updateCirclePointer(ge.rightX, ge.rightY,
                                                   @_config.mirror)
      when 'Palm_Up' then @_view.updateSquarePointer(ge.rightX, ge.rightY,
                                                     @_config.mirror)
      when 'Rest' then @_view.reset()
```

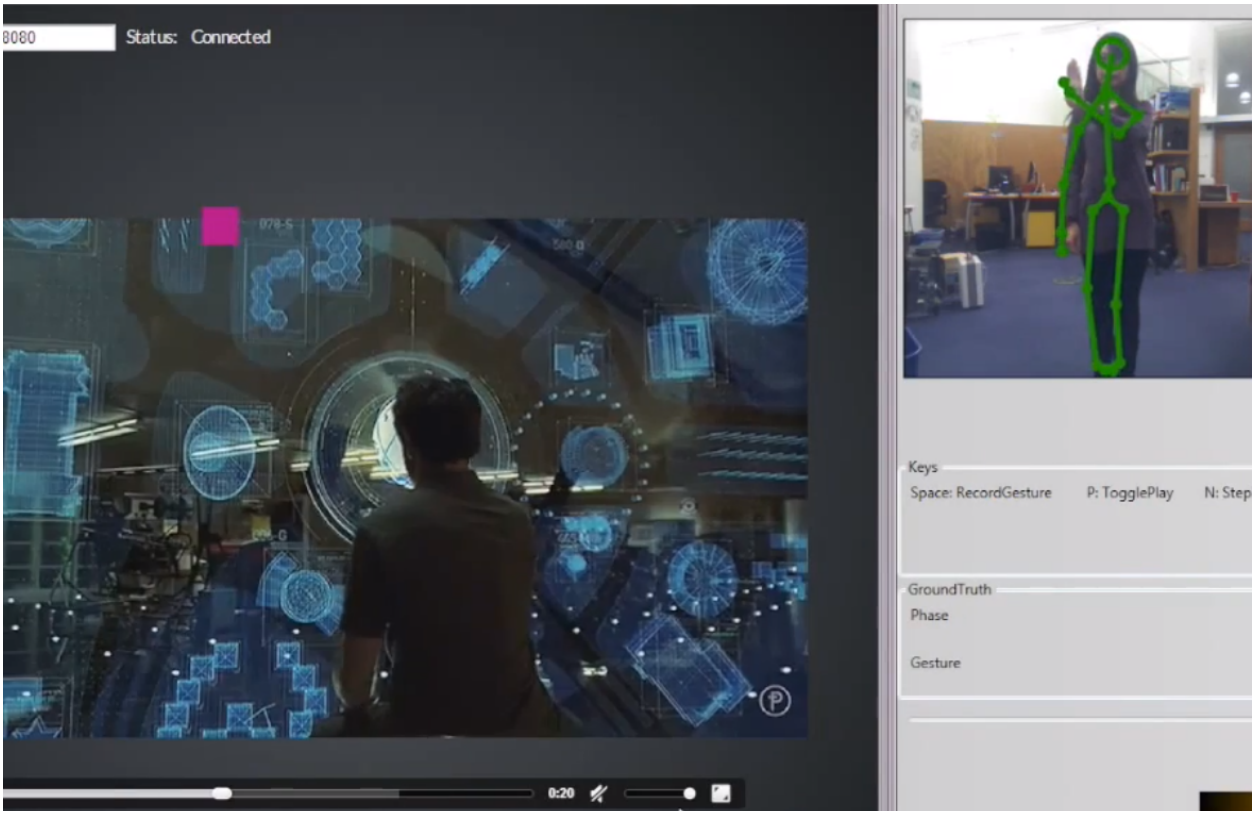


Figure 9-1: Square cursor for Palm Up gesture to seek video forward or backward by moving hand left or right.

9.2.2 Gesture and Speech

Traditionally, gestures are used to augment speech in interaction, as pioneered by Bolt's "Put That There" system [10]. My previous user study [76] shows that speech also can augment gestures.

When using gesture to augment speech, gesture is often used to indicate location. This type of gesture, also called deictic gesture (a pointing gesture), is very effective in conveying spatial information, and is often used to supplement a spoken deictic marker (this, those, here, etc.) [46, 67]. In my application, the user can show a slide by pointing at it in the overview and saying "show this slide" (Figure 9-2). Synonyms of the verb (e.g., "display", "go to") can be used as well to improve flexibility and naturalness.

When speech is used to augment gesture, the spoken words are adjectives and adverbs to modify actions indicated by gestures [76]. This combination is particularly useful when there is a limitation in the expressiveness of the gesture or in the physical space of the gesture. In

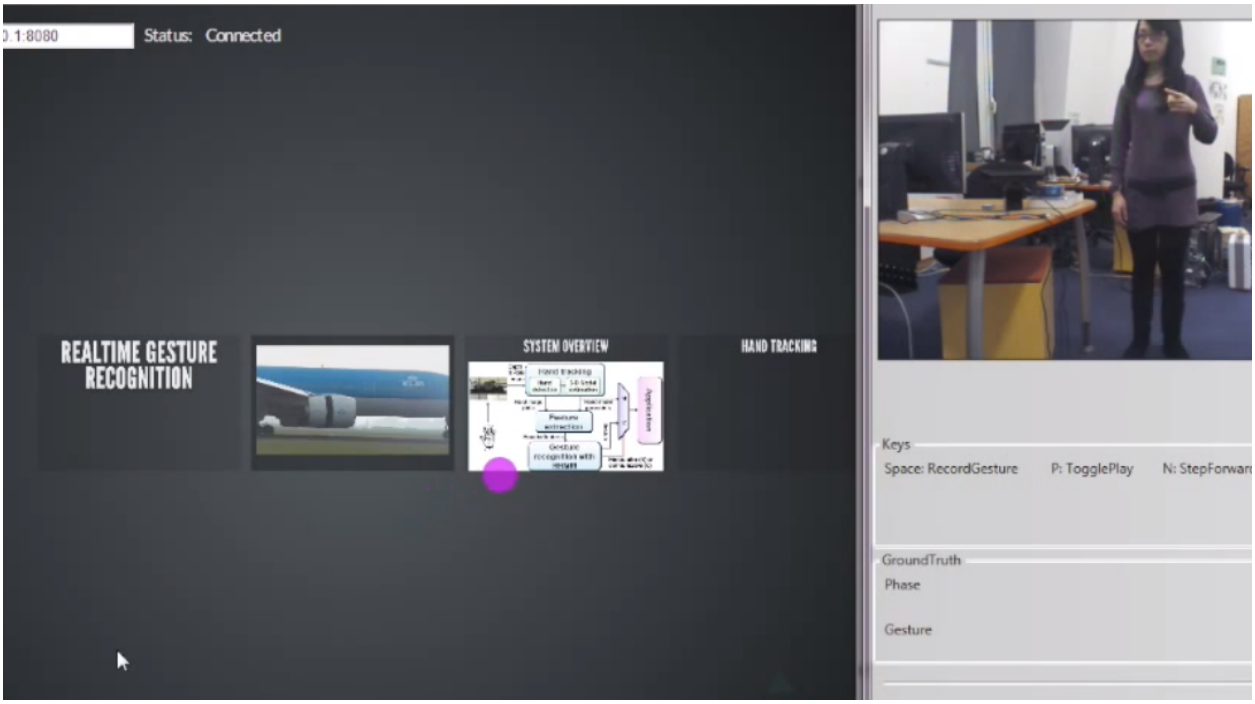


Figure 9-2: In the overview mode, user can point to a slide and say "show this slide" to display it.

my application, when using the Palm Up gesture to move a video forward and backward the user can say "faster" or "slower" or their synonyms to control video speed.

The speech recognition in my system is based on key word spotting by defining grammars in XML (Listing 9.4). The application needs to connect a behavior to each speech tags.

Listing 9.4: An example of grammar definition in XML.

```
<item>
  <tag>SHOW</tag>
  <one-of>
    <item>show</item>
    <item>open</item>
    <item>display</item>
    <item>go to</item>
  </one-of>
</item>
```

When speech and gesture are used together, they not only complement each other, but also reinforce each other. In the presentation application, the behavior to speech events is dependent on gesture event (see Listing 9.5). The dependency helps to reduce false positives

such that if the user says some command words defined in the grammar without doing the gesture, the system will not respond. This allows the user to say those words freely in other context.

Listing 9.5: Code for speech events in CoffeeScript.

```
switch speechText
  when 'MORE'
    @_view.onMore() if @_currentGesture is 'Palm_Up'
  when 'LESS'
    @_view.onLess() if @_currentGesture is 'Palm_Up'
  when 'SHOW'
    @_view.onShowSlide() if @_currentGesture is 'Point'
```

In many commercial speech input interface, a trigger action is required to tell the system that the following speech is intended as a command. The trigger action can either be actions such as holding down a button (e.g., iPhone's Siri interface and Google Now) or specific speech utterance (e.g., "OK Glass" for Google Glass and "Xbox" for Xbox Kinect voice commands). Using a combination of speech and gesture provides an alternative and more natural way to engage the system for voice command.

9.2.3 Natural Direction

The natural direction of interaction depends on the relative position of the display, the user and the sensor. There are two common modes of interaction: users facing the display (e.g., controlling TV); or users facing away from the display (e.g., doing presentation). If the user is facing the display, the UI should show mirrored effects. For example, a Swipe Left gesture would naturally corresponds to "go to the next slide" action. On the other hand, if the user and the display face the same direction (Figure 9-3), the Swipe Left gesture would corresponds more naturally to "go to the previous slide" action.

As a result, the UI responses need to adapt to the sensor and the display coordinate spaces (Figure 9-4). For example, if the display and the sensor are facing in the same direction, when the user points towards the positive x direction in the sensor's coordinate space, the corresponding cursor on the display should also move to the positive x direction in the display's coordinate space (Figure 9-4(a)). On the other hand, if the display and

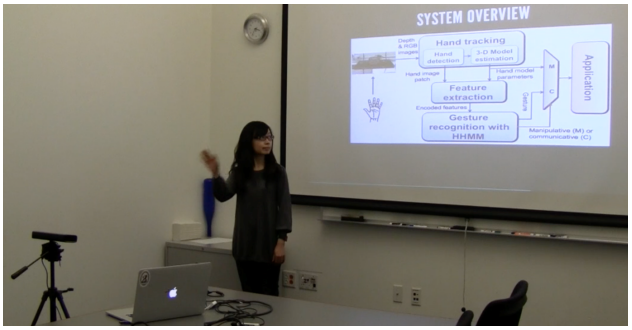
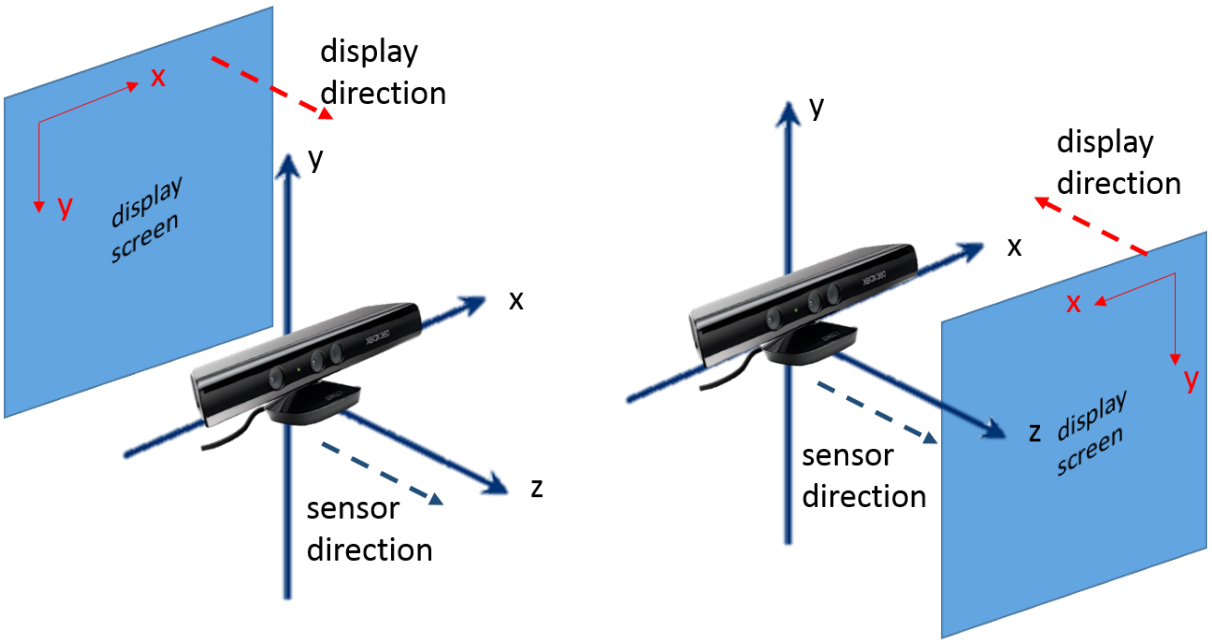


Figure 9-3: The user and the display face the same direction during presentation.

the sensor are facing the opposite directions, when the user points towards the positive x direction in the sensor's coordinate system, the corresponding cursor should move to the negative x direction in the display's coordinate space (Figure 9-4(b)). My system can be easily configured using a flag for these two modes (see Listing 9.3).



(a) Display and sensor face the same direction. (b) Display and sensor face the opposite directions.

Figure 9-4: Sensor and display coordinate spaces.

9.3 Adding New Gestures

Listing 9.6 shows an example of the gesture definition file the system uses. To add a new gesture, the user just needs to add a new line to the file specifying the name of the gesture, the form of the gesture (D for dynamic path gesture, S for static pose gesture), whether the gesture has arbitrary number of repetitions like the Wave gesture (0 means no repetition) and the number of hidden states per gesture (for pose gestures, the number of hidden states must be 1). Then the user can specify the number of examples per gesture to provide and start the training process (Figure 9-5). The system records the new gesture examples and updates the gesture model to include the new one.

Listing 9.6: Gesture definition file.

```
#name,          form, repeat, nHiddenStates
Swipe_Left,       D,    0,     4
Swipe_Right,      D,    0,     3
Circle,           D,    0,     4
Horizontal_Wave,  D,    1,     4
Point,            S,    0,     1
Palm_Up,          S,    0,     1
Next_Point,       D,    0,     3
```

9.4 Discussion

The client server architecture provides a clean separation of the user interface and the back-end recognition engine, hiding the complexity of the multimodal input recognition from application developers. The gesture event based API gives a rich set of input compared to the mouse events (e.g., mouse entered, mouse clicked, mouse dragged), and allows easy mapping to application functionality. The limited functionality of a mouse forces the Windows-Icons-Menus-Pointer (WIMP) paradigm in UI design that has not changed for decades. With a new richer set of gesture input, we open up the potential for designing simpler, more natural, and more intuitive user interfaces.

Figure 9-5: Training interface.

Language is a part of social behavior. What is the mechanism whereby the social process goes on? It is the mechanism of gesture...

George Herbert Mead, *Mind, self, and society*

10 Conclusion

I developed a real-time continuous gesture recognition system for natural human computer interaction. The unified probabilistic framework for handling two forms of gestures is a novel approach, and the evaluation shows promising results: an average online recognition F_1 score of 0.805 using the hybrid performance metric, on a challenging dataset with unsegmented gestures of different forms. The system also achieves real-time performance at 30FPS. Using the framework, I developed a gesture controlled presentation application similar to the one described at the beginning of this paper. All the code is open-source and is available online[1].

Another novel approach is using embedded training and hidden state information to detect gesture phases, allowing the system to respond more promptly. On average, for discrete flow gestures, the system responds 0.6s before the hand comes to rest.

I collected a new dataset that includes two forms of gestures, a combination currently

[1] `http://groups.csail.mit.edu/mug/projects/gesture_kinect/index.html#code`

lacking in the community, and plan to make it public. I also proposed a hybrid performance metric that is more relevant to real-time interaction and different types of gestures.

10.1 Limitations

Currently, the system handles only single-hand gestures. It will be great to add support for two-hand gestures. One challenge for this is handling cases where there is occlusion between the two hands. Once features from both hands are computed, they can be concatenated, and the recognition module could remain the same.

The number of hidden states per gesture needs to be specified by the user or the application developer in the gesture definition file. In the future work, we will make the system automatically learn the number of hidden states through cross-validation and by grouping similar frames [62].

The performance for pose gestures is lower than that for path gestures. Two factors contribute to this problem: the limitation in both the pixel and the depth resolutions of the Kinect sensor, and motion blur. It would be interesting to test with the new version of the Kinect sensor, which uses a time-of-flight depth sensor and is reported to have a higher resolution. Other feature descriptors (e.g., histograms of optic flows) and encoding methods (e.g, sparse coding [36]) can be explored as well.

The performance of a user independent model is relatively low. This is expected because there is large variation among users and the number of users in our dataset is relatively small to train a general model. Even though users can quickly define and train their own models, a relatively accurate base model is always valuable. Hence, more work can be done to improve the base model, e.g., using mixture of HMMs [30]. The mixture of Gaussians used in the current system accounts for variation in each hidden state, but it cannot model variations in transition probabilities between the hidden states. The mixture of HMMs can fill the gap in this regard.

10.2 Future Work

In addition to work that address the limitations, future research should also consider pushing ahead on natural human computer interaction.

Sensors In this thesis, I considered the Kinect and the IMU as input sensors. The gesture controlled presentation application currently only uses the Kinect data. We can explore even more combinations of sensors by including the Leap Motion Controller and smartwatches. Each sensor has strengths of its own, so by combining them we can get the best system overall. The Leap Motion Controller is used mostly as an environmental sensor, but we can also explore its use case a wearable sensor (e.g., attaching it to the arm or wearing it in front of the chest). Its high accuracy in finger tracking may help improve recognition performance for pose gestures.

User adaptation This is a hot topic in both machine learning and HCI, and is applicable to many domains such as speech recognition and touch screen keyboard input [79]. Currently, the system uses a binary decision to do user adaptation: if a user-trained model is present, the system will use that model; otherwise it uses the base model. It would be more convenient if the system learned the user dependent model implicitly while the user is using the system, and used soft decisions to combine the base model and user dependent model, e.g., weighted combination.

Interactive training Due to the large variation in the way users performing gestures and differences in user preference, the gesture recognition system would benefit from having an easy to use interface for fast training. Many efforts in machine learning research assume the availability of training data. Creating interfaces that are easy for users to provide training examples, either in a separate session, or when using the system and provide feedback to the system for wrong recognition, would be an interesting interdisciplinary topic in machine learning and HCI.

Context-aware gesture recognition When I was using the gesture controlled presentation application during a talk, I had to be careful not to do a gesture that the system would recognize and respond to even though that was not my intention. This kind of restriction can make users feel more constrained and reduce the naturalness of the interaction. The

reason for this is that currently, whenever the system detects a movement that matches a gesture model, it will respond. People are much better at discerning whether or not a gesture is intended as a command, based on the context. It will be interesting to add this capability to the system as well. For example, in the gesture controlled presentation application, we might model the context based on what the user is saying and the state of the application (e.g., the content on the slide and the progression on the slide) and make gesture recognition dependent on the context.

A

Review of F-measure

In statistical analysis of binary classification, the F-measure is a measure of test's accuracy, combining both the precision and the recall of the test. Precision is the fraction of correct results from all the returned results (i.e., the number of correct results divided by the number of all returned results), and recall is the fraction of correct results from all the results that should be returned (i.e., the number of correct results divided by the number of results that should have been returned) [3]. Note that both precision and recall have the same numerator.

The general formula for F-measure is

$$\begin{aligned} F_\beta &= (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \\ &= \frac{1+\beta^2}{\frac{1}{\text{precision}} + \frac{\beta^2}{\text{recall}}} \end{aligned}$$

which is the weighted harmonic mean of precision and recall. The F_1 score ($\beta = 1$) is the

most commonly used one where precision and recall have equal weight:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Two other commonly used F measures are the F_2 measure, which weighs recall higher than precision, and the $F_{0.5}$ measure, which puts more emphasis on precision than recall.

B

Principal Component Analysis Optimizations

Principal Component Analysis (PCA) computes a new basis to re-express a dataset. The new basis is a linear combination of the original basis. All the new basis vectors are orthonormal and are in the directions of largest variations in the data [57]. This appendix explains some optimizations I did to speed up PCA computation.

B.1 No Scaling

Assume we have n data points and each data point is represented by a feature vector with dimension d. Let matrix X be the "centered" data (i.e., every feature has mean 0), where columns are data points and rows are features (i.e, a $d \times n$ matrix), then $n\Sigma = XX^T$, where Σ is the covariance matrix of the data and n is the number of data points. The principal components of the data are the eigenvectors of Σ.

Since XX^T and Σ are real symmetric matrices, their eigenvectors can be chosen such that they are real, orthogonal to each other and have norm one. Let $Q_X \Lambda_X Q_X^T$ be the eigendecomposition of XX^T and $Q\Lambda Q^T$ be the eigendecomposition of Σ, and Q_X and Q have unit column vectors, then we have

$$\begin{aligned} nQ\Lambda Q^T &= Q_X \Lambda_X Q_X^T \\ \Lambda &= \frac{1}{n}\Lambda_X \\ Q &= Q_X \end{aligned}$$

This shows that PCA is invariant to the scaling of the data, and will return the same eigenvectors regardless of the scaling of the input [6]. Only the eigenvalues will be scaled by the same factor if the input is scaled.

The total variance in the data is defined as the sum of the variances of the individual components, i.e., the sum of eigenvalues of Σ. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of Σ (sorted in descending order), i.e., the diagonal entries of Λ. Variation explained by k principal components is then given by

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \tag{B.1}$$

We can use the diagonal entries from Λ_X to compute the variation explained because the scaling factor will be canceled out.

B.2 Transpose X

If d is much larger than n, it is more efficient[1] to compute $X^T X$ which is a $n \times n$ matrix instead of $d \times d$. If we find the eigenvectors of this matrix, it would return n eigenvectors, each of dimension n.

Let $Q_{X^T} \Lambda_{X^T} Q_{X^T}^T$ be the eigendecomposition of $X^T X$. Let v_i be an eigenvector of $X^T X$ (i.e., v_i is a column vector of Q_{X^T}), and u_i be an eigenvector of XX^T (i.e., u_i is a column

[1] `http://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/`

vector of Q_X). We can show that

$$(XX^T)^2 = XQ_{X^T}\Lambda_{X^T}Q_{X^T}^T X^T = Q_X \Lambda_X^2 Q_X^T$$

This means that $u_i = s_i X v_i$ where s_i is a scaling factor that normalizes Xv_i. Let λ_i^v and λ_i^u be the eigenlavues corresponding to v_i and u_i.

$$\begin{aligned} X^T X v_i &= \lambda_i^v v_i \\ XX^T X v_i &= \lambda_i^v X v_i \\ XX^T u_i &= \lambda_i^v u_i = \lambda_i^u u_i \end{aligned}$$

This proves that $\lambda_i^u = \lambda_i^v$.

The above shows that we can obtain u_i and λ_i^u from v_i and λ_i^v.

C

Review of State-space Model

State-space models (e.g., HMMs) are used to model sequential data. A dynamic Bayesian network[1] (DBN) [17] provides an expressive language for representing state-space models. It is a way to extend Bayes nets to model probability distribution over a semi-infinite collection of random variables.

C.1 Representation

Let S_t be the hidden state and X_t be the observation at time t. Any state-space model must define a prior, $P(S_1)$, a state-transition function, $P(S_t|S_{t-1})$, and an observation function, $P(X_t|S_t)$. An HMM is one way of representing state-space models.

[1] This part is mainly based on the Kevin Murphy's Ph.D. thesis [42].

C.2 Inference

A state-space model is a model of how S_t generates or "causes" X_t and S_{t+1}. The goal of inference is to invert this mapping, i.e., to infer $S_{1:t}$ given $X_{1:t}$. There are many kinds of inference one can perform on state-space models. Here I list the ones that are relevant to this thesis. See Figure 7-9 for a summary.

C.2.1 Filtering

The most common inference problem in online analysis is to recursively estimate the current belief state at time t using Bayes's rule:

$$\begin{aligned} P(S_t|x_{1:t}) &\propto P(x_t|S_t, x_{1:t-1})P(S_t|x_{1:t-1}) \\ &= P(x_t|S_t)\left[\sum_{s_{t-1}} P(S_t|s_{t-1})P(s_{t-1}|x_{1:t-1})\right] \end{aligned}$$

This task is traditionally called "filtering", because we are filtering out the noise from the observations. We can find this value using the forward algorithm.

C.2.2 Smoothing

Smoothing means we use all the evidence up to the current time to estimate the state of the past, i.e., compute $P(S_{t-l}|x_{1:t})$, where $l > 0$ is the lag. This is traditionally called "fixed-lag smoothing". In the offline case, this is called (fixed-interval) smoothing which corresponds to computing $P(S_t|x_{1:T})$ for all $1 \leq t \leq T$.

C.2.3 Viterbi Decoding

In Vertibi decoding (computing the "most probable explanation"), the goal is to compute the most likely sequence of hidden states give the data:

$$s^*_{1:t} = \arg\max_{s_{1:t}} P(s_{1:t}|x_{1:t})$$

C.2.4 Classification

The likelihood of a model, θ, is $P(x_{1:t}|\theta)$. This can be used to classify a sequence as follows:

$$C^*(x_{1:T}) = \arg\max_C P(x_{1:T}|\theta_C)P(C)$$

where $P(x_{1:t}|\theta_C)$ is the likelihood to the model for class C, and $P(C)$ is the prior for class C. This method ahs the advantage of being able to handle sequences of variable-length. By contrast, most classifiers work with fixed-size feature vectors.

D

Hidden Markov Models

An HMM is an example of a state-space model. It is a stochastic finite automaton, where each state generates (emits) an observation [42].

D.1 Inference

The main inference algorithm for HMMs is the forward-backward algorithm. We compute α recursively in the forward pass as follows:

$$\begin{aligned}\alpha_t(s) &= P(S_t = s|\underline{x}_{1:t}) \\ &= \frac{P(\underline{x}_{1:t-1})P(S_t = s, \underline{x}_{1:t})}{P(\underline{x}_{1:t})P(\underline{x}_{1:t-1})} \\ &= \frac{1}{c_t}P(S_t = s, \underline{x}_t|\underline{x}_{1:t-1})\end{aligned}$$

where

$$\begin{aligned}
P(S_t = s, \underline{x}_t | \underline{x}_{1:t-1}) &= \frac{P(S_t = s, \underline{x}_{1:t})}{P(\underline{x}_{1:t-1})} \\
&= \frac{P(S_t = s, \underline{x}_{1:t-1}) P(S_t = s, \underline{x}_{1:t})}{P(\underline{x}_{1:t-1}) P(S_t = s, \underline{x}_{1:t-1})} \\
&= P(S_t = s | \underline{x}_{1:t-1}) P(\underline{x}_t | S_t = s) \quad \text{(Markov property)} \\
&= \sum_{s'} P(S_t = s, S_{t-1} = s' | \underline{x}_{1:t-1}) P(\underline{x}_t | S_t = s) \\
&= \left[\sum_{s'} P(S_t = s | S_{t-1} = s') P(S_{t-1} = s' | \underline{x}_{1:t-1}) \right] P(\underline{x}_t | S_t = s) \\
&= \left[\sum_{s'} P(S_t = s | S_{t-1} = s') \alpha_{t-1}(s') \right] P(\underline{x}_t | S_t = s)
\end{aligned}$$

and

$$c_t = P(\underline{x}_t | \underline{x}_{1:t-1}) = \sum_s P(S_t = s, \underline{x}_t | \underline{x}_{1:t-1})$$

Without the normalization term c_t, we would be computing $\alpha_t(s) = P(S_t = s, \underline{x}_{1:t})$ which is the original definition of the forward probability. Normalization prevents underflow, and also gives a more meaningful quantity (a filtered state estimate) [42]. We keep track of the normalizing constants so that we can compute the likelihood of the sequence:

$$P(\underline{x}_{1:T}) = P(\underline{x}_1) P(\underline{x}_2 | \underline{x}_1) P(\underline{x}_3 | \underline{x}_{1:2}) \dots P(\underline{x}_T | \underline{x}_{1:T-1}) = \prod_{t=1}^{T} c_t$$

In the backward pass, we compute $\beta_t(i) \stackrel{\text{def}}{=\!=} P(\underline{x}_{t+1:T} | S_t = i)$, the sum of probabilities of all paths starting with state s at position t and going to the end of the sequence. Combining forward and backward passes produces the final answer

$$\gamma_t(i) \stackrel{\text{def}}{=\!=} P(S_t = i | \underline{x}_{1:T}) \propto \alpha_t(s) \beta_t(s)$$

D.2 Termination Probability

Suppose $s \in \{1, 2, \ldots, K\}$, we add an special state END, so $s \in \{1, 2, \ldots, k, END\}$. We also add a special observation at the end of the sequence $x_{1:T}$ to get $(x_{1:T}, x_{END})$. The model takes the following form:

$$P(x_{1:T}, x_{END}, s_{1:T}, END; \underline{\theta}) = t(s_1)t(END|s_T)\prod_{t=2}^{T} t(s_t|s_{t-1})\prod_{t=1}^{T} e(x_t|s_t)$$

We define the following

$$\begin{aligned}
e(x_{END}|END) &= 1 \\
e(x_{END}|s) &= 0 \text{ for } s \neq END \\
e(x_i|END) &= 0 \text{ for } i \neq END \\
t(s|END) &= 0 \quad \forall s \\
\alpha_t(END) &= \begin{cases} \sum_{s' \in \{1 \ldots K\}} \hat{\alpha}_{t-1}(s') \times t(END|s'), & \text{if } t == T+1 \\ 0, & \text{if } t \leq T \end{cases} \\
\beta_{T+1}(s) &= 1 \quad \forall s \\
\beta_T(s) &= \beta_{T+1}(END) \times t(END|s) = t(END|s) \\
\beta_t(END) &= 0 \text{ for } t \leq T
\end{aligned}$$

We want to find $t(END|s)$ for $s \neq END$. The EM update for $t(END|s)$ is

$$t(END|s) = \frac{\sum_{i=1}^{n} \overline{\text{count}}(i, s \to END; \underline{\theta}^-)}{\sum_{i=1}^{n} \sum_{s'} \overline{\text{count}}(i, s \to s'; \underline{\theta}^-)}$$

$$\begin{aligned}
\overline{\text{count}}(i, s \to END; \underline{\theta}) &= P(S_T = s, S_{T+1} = END | x_{i,1:T}, x_{END}; \underline{\theta}); \\
&\propto \alpha_T(s) \times t(END|s) \times \beta_{T+1}(END) \\
&= \alpha_T(s) \times t(END|s) \\
&= \alpha_T(s) \times \beta_T(s) \propto \gamma_T(s)
\end{aligned}$$

We do not need to compute the constant of proportionality explicitly because we normalize $\gamma_t(s)$ at each time step.

Finally, the likelihood of the sequence is

$$P(x_{1:T}, x_{END}; \underline{\theta}) = \prod_{i=1}^{T} c_i \sum_{s \in \{1 \ldots K\}} \alpha_T(s) \times t(END|s)$$

D.3 Learning

D.3.1 Baum-Welch Training

The Baum-Welch algorithm uses the well know Expectation Maximization (EM) algorithm to find the maximum likelihood estimate of the parameters of a HMM model given a set of observed feature vectors [2].

For discrete output, the update for the emission probability at each iteration is

$$e(x|s) = \frac{\sum_{i=1}^{n} \overline{\text{count}}(i, s \rightsquigarrow x; \underline{\theta}^-)}{\sum_{i=1}^{n} \sum_x \overline{\text{count}}(i, s \rightsquigarrow x; \underline{\theta}^-)}$$

where $\overline{\text{count}}(i, s \rightsquigarrow x; \underline{\theta})$ is the expected number of times the state s is paired with paired with the emission x in the training sequence i for parameters $\underline{\theta}$.

For continuous output with Gaussian emission probability, the updates are

$$\begin{aligned}
\underline{\mu}_s &= \frac{\sum_{i=1}^{n} \sum_{t=1}^{T} P(S_t = s|\underline{x}_{i,1:T}; \underline{\theta}^-) \underline{x}_{i,t}}{\sum_{i=1}^{n} \sum_{t=1}^{T} P(S_t = s|\underline{x}_{i,1:T}; \underline{\theta}^-)} \\
&= \frac{\sum_{i=1}^{n} \sum_{t=1}^{T} \gamma_j(s) \underline{x}_{i,t}}{\sum_{i=1}^{n} \sum_{t=1}^{T} \gamma_t(s)} \\
\Sigma_s &= \frac{\sum_{i=1}^{n} \sum_{t=1}^{T} \gamma_t(s) (\underline{x}_{i,t} - \underline{\mu}_s^-)(\underline{x}_{i,t} - \underline{\mu}_s^-)^T}{\sum_{i=1}^{n} \sum_{t=1}^{T} \gamma_t(s)}
\end{aligned}$$

The $^-$ superscript denotes parameters from the previous iteration.

For continuous output with mixture of Gaussians emission probability, the update are

$$\underline{\mu}_{s,q} = \frac{\sum_{i=1}^{n}\sum_{t=1}^{T} P(S_t = s, Q_t = q|\underline{x}_{i,1:T};\underline{\theta}^{-})\underline{x}_{i,t}}{\sum_{i=1}^{n}\sum_{t=1}^{T} P(S_t = s, Q_t = q|\underline{x}_{i,1:T};\underline{\theta}^{-})}$$

$$\Sigma_{s,q} = \frac{\sum_{i=1}^{n}\sum_{t=1}^{T} p(S_t = s, Q_t = q|\underline{x}_{i,1:T};\underline{\theta}^{-})(\underline{x}_{i,t} - \underline{\mu}_s^{-})(\underline{x}_{i,t} - \underline{\mu}_s^{-})^T}{\sum_{i=1}^{n}\sum_{t=1}^{T} p(S_t = s, Q_t = q|\underline{x}_{i,1:T};\underline{\theta}^{t-1})}$$

$$\begin{aligned} P(S_t = s, Q_t = q|\underline{x}_{i,1:T};\underline{\theta}^{-}) &= \gamma_t(s)P(Q_t = q|S_t = s, \underline{x}_{i,t}) \\ &= \gamma_t(s)\frac{P(Q_t = q|S_t = s)N(\underline{x}_{i,t};\mu_{s,q},\Sigma_{s,q})}{e(\underline{x}_{i,t}|s)} \end{aligned}$$

D.3.2 Viterbi Training

Instead of summing over all possible paths as in Baum-Welch training, Viterbi training only considers the single most likely path computed using the Viterbi algorithm. In each iteration, the Viterbi path for each training sequence is computed, and the model parameters are updated. For single Gaussian emission probability, the update at iteration t is

$$t^t(s'|s) = \frac{\sum_{i=1}^{n}\text{count}(i, s \to s';\underline{\theta}^{t-1})}{\sum_{i=1}^{n}\sum_{s'}\text{count}(i, s \to s';\underline{\theta}^{t-1})}$$

$$\underline{\mu}_s^t = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\underline{x}_{i,j}}{\sum_{i=1}^{n}\sum_{s'}\text{count}(i, s \to \underline{s}')} \quad \text{s.t. } S_j = s$$

where $\text{count}(i, s \to s';\underline{\theta}^{t-1})$ is the number of times the transition $s \to s'$ is seen in the most likely path for sequence i.

Viterbi training is much faster than Baum-Welch training, but does not work quite as well. However, it can be used in the initialization step to provide better start values for Baum-Welch training.

D.4 Embedded Training

Embedded training simultaneously updates all of the HMMs in a system using all of the training data. In HTK [80] for speech recognition, it is suggested to perform one iteration of EM update for embedded training after individual phone HMMs are trained. This is because repeated re-estimation may take an impossibly long time to converge. Worse still, it can lead

to over-training since the models can become too closely matched to the training data and fail to generalize well on unseen test data.

E

Review of Conditional Random Fields

This chapter is based on [66]. Let variables $\mathbf{y}$ be the attributes of the entities that we wish to predict, with input variables $\mathbf{x}$ representing the observation about the entities. Conditional random fields [32] is simply a conditional distribution $p(\mathbf{y}|\mathbf{x})$ with an associated graphical structure.

In a CRF with a general graphical structure, we can partition the factors of the graph into $\mathcal{C} = \{C_1, C_2, \ldots, C_p\}$ where each C_p is a *clique template* whose parameters are tied. The clique template C_p is a set of factors which has a corresponding set of sufficient statistics $\{f_{pk}(\mathbf{x}_p, \mathbf{y}_p)\}$ and parameters $\underline{\theta}_p \in \mathcal{R}^{K(p)}$ where $K(p)$ is the number of feature functions in C_p. The CRF can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \underline{\theta}_p)$$

where each factor is parameterized as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \underline{\theta}_p) = \exp\left\{\sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)\right\}$$

and the normalization function is

$$\begin{aligned} Z(\mathbf{x}) &= \sum_{\mathbf{y}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x_c}, \mathbf{y_c}; \underline{\theta}_p) \\ &= \sum_{\mathbf{y}} \exp\left\{\sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)\right\} \end{aligned}$$

The conditional log-likelihood is given by

$$\ell(\underline{\theta}) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x})$$

The partial derivative of the log-likelihood with respect to a parameter λ_{pk} associated with a clique template C_p is

$$\begin{aligned} \frac{\partial \ell}{\partial \lambda_{pk}} &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \frac{1}{Z(\mathbf{x})} \frac{\partial Z(\mathbf{x})}{\partial \lambda_{pk}} \\ &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}} \exp\left\{\sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)\right\} \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \\ &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}) \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \\ &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \frac{1}{Z(\mathbf{x})} \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}) f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \\ &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \frac{1}{Z(\mathbf{x})} \sum_{\Psi_c \in C_p} \sum_{\mathbf{y'_c}} p(\mathbf{y'_c}, \mathbf{x}) f_{pk}(\mathbf{x}_c, \mathbf{y'}_c) \\ &= \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y'_c}} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) p(\mathbf{y'_c} | \mathbf{x}) \end{aligned}$$

E.1 Linear-Chain CRF

In linear-chain CRF, each feature function has the form $f_k(y_t, y_{t-1}, \underline{x}_t)$. We have one feature $f_{ij}(y, y', \underline{x}) = \mathbb{1}(y = i)\mathbb{1}(y' = j)$ for each transition (i, j) and one feature $f_i(y, y', \underline{x}) = \underline{x}$ for each state. Let M be the total number of hidden states, and d be the dimension of the observation $\underline{x}$. The number of model parameters is $O(M^2 + dM)$. The total training cost is $O(TM^2NG)$ where T is sequence length, N is the number of training examples and G is the number of gradient computations required by the optimization procedure. Hence if the number of states is large, the computation can be expensive [66].

The forward-backward algorithm for linear-chain CRF is identical to the HMM version except that the transition weights $\Psi_t(j, i, \underline{x}_t)$ are defined differently.

The Viterbi recursion for linear-chain CRF can be computed similarly as in HMMs as well

$$\delta_t(s) = \max_{s' \in \mathcal{S}} \Psi_t(s, s', \underline{x}_t)\delta_{t-1}(s')$$

Like in HMMs, the vectors α_t, β_t and δ_t are normalized to prevent underflow.

E.2 LDCRF

LDCRF has a linear chain graphical structure as well. The objective function to learn the parameter $\underline{\theta}^*$ is:

$$l(\underline{\theta}) = \sum_{i=1}^{n} \log P(y_{i,1:T}|\underline{x}_{i,1:T}, \underline{\theta}) - \frac{1}{2\sigma^2}||\underline{\theta}||^2$$

where

$$\begin{aligned}
\log P(y_{i,1:T}|\underline{x}_{i,1:T},\underline{\theta}) &= \log \sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} P(h_{1:T}|\underline{x}_{1:T},\underline{\theta}) \\
&= \log \sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} \frac{1}{Z(\underline{x}_{1:T})} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right) \\
&= \log \sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right) - \log Z(\underline{x}_{1:T})
\end{aligned}$$

The gradient of $\log P(y_{i,1:T}|\underline{x}_{i,1:T},\underline{\theta})$ with respect to the parameter λ_k is

$$\frac{\sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right) f_k(h_{1:T},\underline{x}_{1:T})}{\sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right)} - \frac{\sum_{h_{1:T}} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right) f_k(h_{1:T},\underline{x}_{1:T})}{\sum_{h_{1:T}} \exp\left(\sum_k \lambda_k f_k(h_{1:T},\underline{x}_{1:T})\right)}$$

$$= \sum_{h_{1:T}:\forall h_t \in \mathcal{H}_{y_t}} P(h_{1:T}|y_{1:T},\underline{x}_{1:T},\underline{\theta}) f_k(h_{1:T},\underline{x}_{1:T}) - \sum_{h_{1:T}} P(h_{1:T}|\underline{x}_{1:T},\underline{\theta}) f_k(h_{1:T},\underline{x}_{1:T})$$

F

Notation and Abbreviations

We adopt the standard convention that random variables are denoted as capital letters, and instantiations of random variables (values) are denoted as lower-case letters. We use underlines for vector-valued quantities to distinguish them from scalar-valued ones. So x refers to a scalar, while $\underline{x}$ refers to a vector. We use caligraphic letters to denote sets.

Symbol	Meaning
S_t	Hidden state variable at time t
X_t	Observation (output) at time t
T	Length of sequence
H	Size of hidden state space
l	Lag
k	Number of mixtures in MoG
$\underline{x}_{1:T}$	Sequence of observation
$t(s)$	Initial state probability: $P(S_1 = s)$
$t(s'\|s)$	Transition probability: $P(S_t = s'\|S_{t-1} = s)$
$e(\underline{x}_t\|s)$	Emission probability at time t: $P(\underline{x}_t\|S_t = s)$
$\alpha_t(s)$	$P(S_t = s\|\underline{x}_{1:t})$
$\beta_t(s)$	$P(\underline{x}_{t+1:T}\|S_t = s)$
$\gamma_t(s)$	$P(S_t = s\|\underline{x}_{1:T})$

Table F.1: Notation for HMMs.

Abbreviation	Meaning
ATSR	Accurate temporal segmentation rate
BIC	Bayesian Information Criterion
CF	Continuous flow
CPD	Conditional probability distribution
CRF	Conditional random fields
DBN	Dynamic Bayesian network
DF	Discrete flow
FPS	Frame per second
IMU	Inertia measurement unit
HCI	Human computer interaction
HHMM	Hierarchical hidden Markov model
HMM	Hidden Markov model
HOG	Histogram of oriented gradients
LDCRF	Latent dynamic conditional random field
MoG	Mixture of Gaussians
PCA	Principal component analysis
SVM	Support Vector Machine
TP	True positive
TPR	True positive rate
UI	User interface

Table F.2: List of abbreviations

Bibliography

[1] Principal component analysis. https://inst.eecs.berkeley.edu/~ee127a/book/login/l_sym_pca.html.

[2] Baum–Welch algorithm. http://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm, Apr 2014.

[3] F1 score. http://en.wikipedia.org/wiki/F1_score, Apr 2014.

[4] Kinect Space user manual. https://kineticspace.googlecode.com/files/kinetic_space.pdf, May 2014.

[5] Leap Motion. http://en.wikipedia.org/wiki/Leap_Motion, 2014.

[6] PCA. http://deeplearning.stanford.edu/wiki/index.php/PCA, Feb 2014.

[7] B. Bauer and H. Hienz. Relevant features for video-based continuous sign language recognition. In *FG*, page 440, 2000.

[8] A. Ben-Hur and J. Weston. A users guide to support vector machines. In *Data mining techniques for the life sciences*, pages 223–239. Springer, 2010.

[9] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[10] R. A. Bolt. "Put-That-There": Voice and gesture at the graphics interface. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270, New York, NY, USA, 1980. ACM.

[11] G. Bradski and A. Kaehler. *Learning OpenCV*. The Art of Computer Programming. O'Reilly, first edition, 2008.

[12] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface, 1998.

[13] S. K. Card, G. G. Robertson, and J. D. Mackinlay. *The information visualizer, an information workspace*, pages 181–186. ACM, 1991.

[14] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: multimodal interaction for distributed applications. In *MULTIMEDIA '97: Proceedings of the fifth ACM international conference on Multimedia*, pages 31–40, New York, NY, USA, 1997. ACM.

[15] R. Cutler and M. Turk. View-based interpretation of real-time optical flow for gesture recognition. In *Proc. Automatic Face and Gesture Recognition*, pages 416–421, 1998.

[16] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886–893, June 2005.

[17] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, Dec. 1989.

[18] S. Dicintio. Comparing approaches to initializing the expectation-maximization algorithm. Master's thesis, The University of Guelph, 2012.

[19] S. Escalera, J. Gonzàlez, X. Baró, M. Reyes, O. Lopes, I. Guyon, V. Athitsos, and H. Escalante. Multi-modal gesture recognition challenge 2013: Dataset and results. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 445–452. ACM, 2013.

[20] S. Fels, R. Pritchard, and A. Lenters. Fortouch: A wearable digital ventriloquized actor. In *New Interfaces for Musical Expression*, pages 274–275, 2009.

[21] C. Fraley and A. E. Raftery. Mclust version 3: an r package for normal mixture modeling and model-based clustering. Technical report, DTIC Document, 2006.

[22] J. Françoise, B. Caramiaux, and F. Bevilacqua. Realtime segmentation and recognition of gestures using hierarchical markov models. *Mémoire de Master, Université Pierre et Marie Curie–Ircam*, 2011.

[23] W. T. Freeman and M. Roth. Orientation histograms for hand gesture recognition. In *FG*, volume 12, pages 296–301, 1995.

[24] I. Guyon, V. Athitsos, P. Jangyodsuk, and H. J. Escalante. The chalearn gesture dataset (CGD 2011), 2013.

[25] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[26] C. Harrison, H. Benko, and A. Wilson. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 441–450. ACM, 2011.

[27] S. Izadi, S. Hodges, S. Taylor, D. Rosenfeld, N. Villar, A. Butler, and J. Westhues. Going beyond the display: a surface technology with an electronically switchable diffuser. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 269–278. ACM, 2008.

[28] M. W. Kadous. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. PhD thesis, The University of New South Wales, 2002.

[29] A. Kendon. *Current issue in the study of gesture*. Lawrrence Erlbaum Assoc., 1986.

[30] C. Keskin, E. Berger, and L. Akarun. A unified framework for concurrent usage of hand gesture, shape and pose. `http://goo.gl/2icUwv`.

[31] K. Khoshelham. Accuracy analysis of kinect depth data. In *ISPRS workshop laser scanning*, volume 38, page 1, 2011.

[32] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[33] I. Laptev and T. Lindeberg. Space-time interest points. In *Proc. ICCV*, pages 432–439 vol.1, 2003.

[34] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, pages 1–8, 2008.

[35] E. Larson, G. Cohn, S. Gupta, X. Ren, B. Harrison, D. Fox, and S. Patel. Heatwave: thermal imaging for surface user interaction. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2565–2574. ACM, 2011.

[36] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.

[37] S. Marcel. Hand posture and gesture datasets. http://www.idiap.ch/resource/gestures/.

[38] A. Marcos-Ramiro, D. Pizarro-Perez, M. Marron-Romera, L. S. Nguyen, and D. Gatica-Perez. Body communicative cue extraction for conversational analysis. In *Automatic Face and Gesture Recognition*, 2013.

[39] D. McNeill and E. Levy. *Conceptual representations in language activity and gesture*. Wiley, 1982.

[40] P. Mistry and P. Maes. Sixthsense: a wearable gestural interface. In *ACM SIGGRAPH ASIA 2009 Sketches*, page 11. ACM, 2009.

[41] L. Morency, A. Quattoni, and T. Darrell. Latent-dynamic discriminative models for continuous gesture recognition. In *CVPR*, 2007.

[42] K. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[43] A. Ng and A. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

[44] I. Oikonomidis, N. Kyriazis, and A. A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, pages 1–11, 2011.

[45] K. Oka, Y. Sato, and H. Koike. Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications*, 22(6):64–71, 2002.

[46] S. Oviatt, A. DeAngeli, and K. Kuhn. Integration and synchronization of input modes during multimodal human-computer interaction. In *Referring Phenomena in a Multimedia Context and their Computational Treatment*, pages 1–13. Association for Computational Linguistics, 1997.

[47] V. I. Pavlović, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:677–695, 1997.

[48] B. Peng and G. Qian. Online gesture spotting from visual hull data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (99):1–1, 2011.

[49] PrimeSence Inc. *PrimeSense™NIT Algorithms*, 1.5 edition, 2011.

[50] F. Quek. Eyes in the interface, 1995.

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

[52] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[53] I. Rauschert, P. Agrawal, I. R. Pyush, and R. Sharma. Designing a human-centered, multimodal GIS interface to support emergency management, 2002.

[54] S. Ruffieux, D. Lalanne, E. Mugellini, and D. Roggen. Chairgest - A challenge for multimodal mid-air gesture recognition for close HCI. In *ICMI*, 2013.

[55] R. Sharma, J. Cai, S. Chakravarthy, I. Poddar, and Y. Sethi. Exploiting speech/gesture co-occurrence for improving continuous gesture recognition in weather narration. In *FG*, pages 422–427, 2000.

[56] M. C. Shin, L. V. Tsap, and D. B. Goldgof. Gesture recognition using bezier curves for visualization navigation from registered 3-D data. *Pattern Recognition*, 37(5):1011 – 1024, 2004.

[57] J. Shlens. A tutorial on principal component analysis. *Systems Neurobiology Laboratory, University of California at San Diego*, 82, 2005.

[58] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Commun. ACM*, 56(1):116–124, Jan. 2013.

[59] Y. Song, D. Demirdjian, and R. Davis. Multi-signal gesture recognition using temporal smoothing hidden conditional random fields. In *Proc. FG*, pages 388–393, 2011.

[60] Y. Song, D. Demirdjian, and R. Davis. Tracking body and hands for gesture recognition: Natops aircraft handling signals database. In *FG*, pages 500–506, March 2011.

[61] Y. Song, D. Demirdjian, and R. Davis. Continuous body and hand gesture recognition for natural human-computer interaction. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2012.

[62] Y. Song, L.-P. Morency, and R. Davis. Action recognition by hierarchical sequence summarization. In *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3562–3569, Portland, OR, June 2013.

[63] T. Starner and A. Pentland. Visual recognition of American sign language using hidden Markov models. In *FG*, pages 189–194, 1995.

[64] E. B. Sudderth, M. I. M, W. T. Freeman, and A. S. Willsky. Visual hand tracking using nonparametric belief propagation. In *IEEE Workshop on Generative Model Based Vision*, page 189, 2004.

[65] P. Suryanarayan, A. Subramanian, and D. Mandalapu. Dynamic hand pose recognition using depth data. In *ICPR*, pages 3105–3108, 2010.

[66] C. Sutton and A. McCallum. *An introduction to conditional random fields for relational learning*, volume 2. Introduction to statistical relational learning. MIT Press, 2006.

[67] E. Tse, S. Greenberg, C. Shen, and C. Forlines. Multimodal multiplayer tabletop gaming. *Comput. Entertain.*, 5(2), Apr. 2007.

[68] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, C. Schmid, et al. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009.

[69] R. Wang, S. Paris, and J. Popović. 6d hands: markerless hand-tracking for computer aided design. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 549–558. ACM, 2011.

[70] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 28(3), 2009.

[71] S. Wang, A. Quattoni, L. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *CVPR*, volume 2, pages 1521–1527, 2006.

[72] J. A. Ward, P. Lukowicz, and H. W. Gellersen. Performance metrics for activity recognition. *ACM Trans. Intell. Syst. Technol.*, 2(1):6:1–6:23, Jan. 2011.

[73] C. wei Hsu, C. chung Chang, and C. jen Lin. A practical guide to support vector classification, 2010.

[74] J. Wobbrock, M. Morris, and A. Wilson. User-defined gestures for surface computing. In *CHI*, pages 1083–1092, 2009.

[75] H. Yang, A. Park, and S. Lee. Gesture spotting and recognition for human–robot interaction. *Robotics, IEEE Transactions on*, 23(2):256–270, 2007.

[76] Y. Yin. Toward an intelligent multimodal interface for natural interaction. Master's thesis, 2010.

[77] Y. Yin and R. Davis. Toward natural interaction in the real world: Real-time gesture recognition. In *Proc. of ICMI*, November 2010.

[78] Y. Yin and R. Davis. Gesture spotting and recognition using salience detection and concatenated hidden markov models. In *Proc. of ICMI*, pages 489–494, 2013.

[79] Y. Yin, T. Y. Ouyang, K. Partridge, and S. Zhai. Making touchscreen keyboards adaptive to keys, hand postures, and individuals - a hierarchical spatial backoff model approach. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI'13, pages 2775–2784, Paris, France, April 2013.

[80] S. J. Young. The HTK hidden Markov model toolkit: Design and philosophy. *Entropic Cambridge Research Laboratory, Ltd*, 2:2–44, 1994.

[81] Z. Zhang. Microsoft kinect sensor and its effect. *MultiMedia, IEEE*, 19(2):4–10, Feb 2012.