A Wearable System that Learns a Kinematic Model and Finds Structure in Everyday Manipulation by using Absolute Orientation Sensors and a Camera

by

Charles Clark Kemp

S.B. Electrical Engineering and Computer Science (1997),
M.Eng. Electrical Engineering and Computer Science (1997),
Massachusetts Institute of Technology

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 2005

Author .....
Department of Electrical Engineering and Computer Science
May 20th, 2005

Certified by .....
Rodney Brooks
Matsushita Professor of Robotics
Thesis Supervisor

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

A Wearable System that Learns a Kinematic Model and Finds Structure in Everyday Manipulation by using Absolute Orientation Sensors and a Camera

by

Charles Clark Kemp

Submitted to the Department of Electrical Engineering and Computer Science
on May 20th, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis presents Duo, the first wearable system to autonomously learn a kinematic model of the wearer via body-mounted absolute orientation sensors and a head-mounted camera. With Duo, we demonstrate the significant benefits of endowing a wearable system with the ability to sense the kinematic configuration of the wearer's body. We also show that a kinematic model can be autonomously estimated offline from less than an hour of recorded video and orientation data from a wearer performing unconstrained, unscripted, household activities within a real, unaltered, home environment. We demonstrate that our system for autonomously estimating this kinematic model places very few constraints on the wearer's body, the placement of the sensors, and the appearance of the hand, which, for example, allows it to automatically discover a left-handed kinematic model for a left-handed wearer, and to automatically compensate for distinct camera mounts, and sensor configurations. Furthermore, we show that this learned kinematic model efficiently and robustly predicts the location of the dominant hand within video from the head-mounted camera even in situations where vision-based hand detectors would be likely to fail. Additionally, we show ways in which the learned kinematic model can facilitate highly efficient processing of large databases of first person experience. Finally, we show that the kinematic model can efficiently direct visual processing so as to acquire a large number of high quality segments of the wearer's hand and the manipulated objects.

Within the course of justifying these claims, we present methods for estimating global image motion, segmenting foreground motion, segmenting manipulation events, finding and representing significant hand postures, segmenting visual regions, and detecting visual points of interest with associated shape descriptors. We also describe our architecture and user-level application for machine augmented annotation and browsing of first person video and absolute orientations. Additionally, we present a real-time application in which the human and wearable cooperate through tightly integrated behaviors coordinated by the wearable's kinematic perception, and together acquire high-quality visual segments of manipulable objects that interest the wearable.

Thesis Supervisor: Rodney Brooks
Title: Matsushita Professor of Robotics

Acknowledgments

This thesis is dedicated to my Aunt Phyllis.

It's been a long journey. On the whole, I believe it was worthwhile. I've learned a great deal about machine intelligence, which I continue to believe has a good chance of demonstrably surpassing unaugmented human intelligence within the next half century. I've also had the pleasure of working with many wonderful people, starting with Cynthia Breazeal who helped bring me into Rod's lab. Back then, the old Cog group and the Zoo served as exciting inspirations for work on machine intelligence, especially in humanoid form. From then until now is a long and blurry history of cool people, strange stories, interesting projects, and gradual progress towards completion. I'm not going to attempt to name every important person over these years. Instead, I'm going to restrict these acknowledgments to the many people who were directly relevant to the specific research presented within this thesis.

I've got to start by thanking my wife, Dr. Melissa Kemp, who has been incredibly supportive of my doctoral work, even during some very hard times. I feel extraordinarily lucky to be married to Melissa. I acknowledge that I love her. Likewise, my parents and brother deserve credit for continuing to give me support and encouragement even when the task appeared daunting, if not insurmountable. The memory of my grandfather, John Russell Clark, also deserves credit for helping to instill a love of MIT, engineering, and the exploration of cutting-edge technologies.

After family, my thanks must go to Rod Brooks, who has supported my development as a researcher over many years. I have great admiration for Rod's contributions to the pursuit of AI. He has continued to be a bold visionary in his endeavors, even when the rest of the field has scorned his approach, or results have lagged his vision. In the long run, many of these bets have paid off, sometimes in unexpected ways, and they have consistently influenced the field in beneficial ways. Although we may sometimes disagree on details, I find that I usually agree with Rod's larger perspective, which is refreshingly exciting after a journey into the often prosaic literature of AI.

Next, I must thank Prof. Leslie Pack Kaelbling and Prof. Trevor Darrell, both of whom have been very patient and helpful over the course of this thesis.

After family and my thesis committee, my thanks should clearly go to Dr. Paul Fitzpatrick. Besides being a friend and fellow researcher with whom I've enjoyed many conversations and collaborations, Paul has been invaluable to the completion of my thesis work. More than anyone else outside of my family, Paul has encouraged my work and helped me overcome some of the darker moments in my doctoral pursuit. He's also provided incredibly generous practical help, from the mundane assistance of inserting references into a bibtex file during a desperate moment, to happily organizing a fun capture session with Noemi, to essential phone calls that helped clarify the ways forward. Thank you, Paul!

Jeff Weber comes next, since he donated his prodigious talent to the creation of the latest version of Duo's hardware. His great skill in design and construction helped move Duo from a duct-tape covered prototype to a highly presentable and robust wearable system. I am very grateful for his contribution to this transformation. Jeff is also a great deal of fun with an excellent sense of humor, so I'm glad he's a friend.

Eduardo Torres-Jara has also made significant contributions to Duo's hardware through his impressive expertise with electronics. He provided essential assistance in the creation of the circuitry governing the LED array for the old version of Duo, and the power conversion

and distribution board for the new version of Duo. He's also been a good friend and collaborator with whom I've enjoyed working on various projects, including Yuppy and Coco.

Although the ranking gets less clear, this seems to be a proper location at which to thank Marilyn Pierce, a powerful force in graduate administration for EECS at MIT. On several occasions, Marilyn has converted stressful and complicated situations with poor options, into positive ways to move forward. Her help has been very significant to the completion of this thesis in a variety of ways.

Besides many excellent conversations over the years, Matt Marjanović gave me the files and instructions necessary to use LyX to write this thesis. LyX greatly eased the writing process, which is painful enough without trying to mentally visualize figures and equations.

The entire LBR crew has been helpful, both by providing excellent company, and actually reading the document! Thank you Bryan Adams, Lijin Aryanada, Jessica Banks, Aaron Edsinger-Gonzales, Lorenzo Natale, and Una-May O'Reilly. Una-May also helped come up with the name Duo, and so deserves this special note of thanks.

Also, thanks goes to the two extremely lovely, talented and generous people who provided data!

Finally, I must thank the many talented people with whom I anonymously collaborated via open source software. Without the truly amazing open source software on which the Duo platform is built, this thesis would not have been possible.

As I mentioned at the start, there are many other people who have been vital to the positive aspects of my time as a graduate student. I look forward to thanking these people in another forum. If the reader is one of these people, please accept my thanks. You know who you are.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents Duo, the first wearable system to autonomously learn a kinematic model of the wearer via body-mounted absolute orientation sensors and a head-mounted camera. With Duo, shown in figures 1-1 and 1-2, we demonstrate the significant benefits of endowing a wearable system with the ability to sense the kinematic configuration of the wearer's body, see figure 1-3. We also show that a kinematic model can be autonomously estimated offline from less than an hour of recorded video and orientation data from a wearer performing unconstrained, unscripted, household activities within a real, unaltered, home environment. We demonstrate that our system for autonomously estimating this kinematic model places very few constraints on the wearer's body, the placement of the sensors, and the appearance of the hand, which, for example, allows it to automatically discover a left-handed kinematic model for a left-handed wearer, and to automatically compensate for distinct camera mounts, and sensor configurations. Furthermore, we show that this learned kinematic model efficiently and robustly predicts the location of the dominant hand within video from the head-mounted camera even in situations where vision-based hand detectors would be likely to fail. Additionally, we show ways in which the learned kinematic model can facilitate highly efficient processing of large databases of first person experience. Finally, we show that the kinematic model can efficiently direct visual processing so as to acquire a large number of high quality segments of the wearer's hand and manipulated objects.

Within the course of justifying these claims, we will present methods for estimating global image motion, segmenting foreground motion, segmenting manipulation events, finding and representing significant hand postures, segmenting visual regions, and detecting

Figure 1-1: The most recent version of the wearable system, Duo.

visual points of interest with associated shape descriptors. We will also describe our architecture and user-level application for machine augmented annotation and browsing of first person video and absolute orientations. Additionally, we will present a real-time application in which the human and wearable cooperate through tightly integrated behaviors coordinated by the wearable's kinematic perception, and together acquire high-quality visual segments of manipulable objects that interest the wearable.

1.1 The Data Sets

Although we have captured many hours of data, the results we present in this thesis primarily use three data sets that cover a number of different types of variation. The data sets are from a right-handed adult male, a left-handed adult female, and a right-handed adult female. They were taken in two different home environments across a number of rooms each.

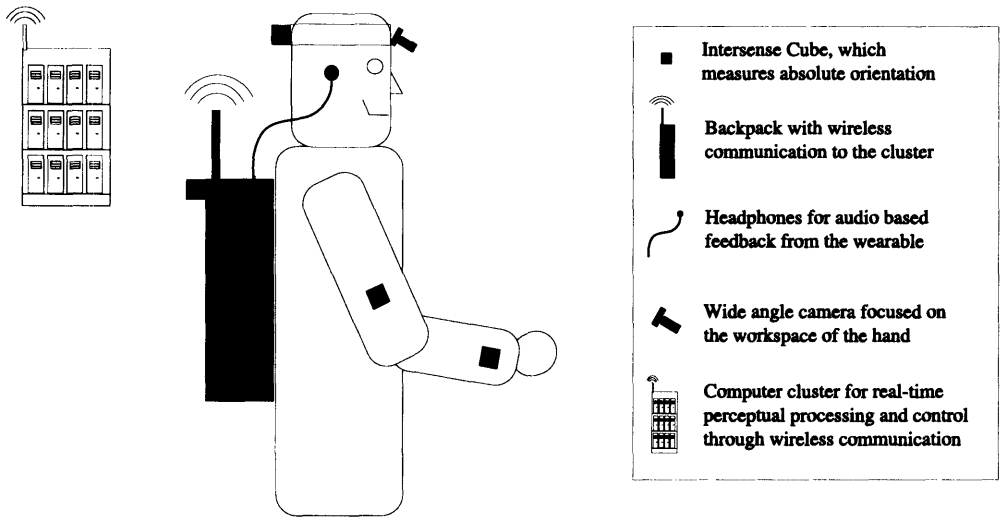


Figure 1-2: A diagram of the most recent version of the wearable system, Duo.

Figure 1-3: This figure shows snapshots of the data captured by Duo. The top row shows frames of first-person video, and the bottom row shows the etimated kinematic configuration of the wearer's body associated with each frame.

The two data sets from the same home were taken almost a year apart, during which time a number of features of the home changed including furniture, furniture arrangement, and pictures on the wall. The data sets also use two different versions of the capture hardware, the initial and the final versions, which have fundamental differences in sensor layout and orientation. The wearer in the first data set is the author who understood the mechanisms and objectives of the research and performed a variety of actions for testing. The wearer in the second data had not seen the system before and was unaware of any of the research details. She mostly performed household chores she had intended to perform that evening. The wearer in the third data set had seen the system before and understood the objectives of the research, but lacked any detailed knowledge of the research.

1.1.1 Data Set 1

This 18 minute data set was captured with the original version of the Duo hardware, which includes a camera mounted on the brim of a cap, which can be seen in the upper-left corner of the image, see figure 1-4, and a front mounted torso orientation sensor. This data set was taken while the right-handed adult male wearer performed a number of common manual activities within the first home environment.

1.1.2 Data Set 2

This hour long data set was captured with the most recent version of the Duo hardware, which includes a camera mounted like a headlamp. This data was captured within the second home environment while the right-handed adult female wearer performed a number of common manual activities within the second home environment, see figure 1-5.

1.1.3 Data Set 3

This 48 minute long data set was captured with the most recent version of the Duo hardware. This data was captured within the second home environment while the left-handed adult female wearer performed a number of common manual activities over a year later within the first home environment, see figure 1-6.

Figure 1-4: This image and kinematic visualization shows an example from data set 1. The kinematic model shown is the hand-tuned model for the latest version of Duo. We use this same model without alteration to visualize the examples from each data set. Using this non-matching model shows how the three data sets differ in their orientation data. In this case the torso sensor is placed differently than in the other two data sets, which leads to the clearly incorrect torso visualization.

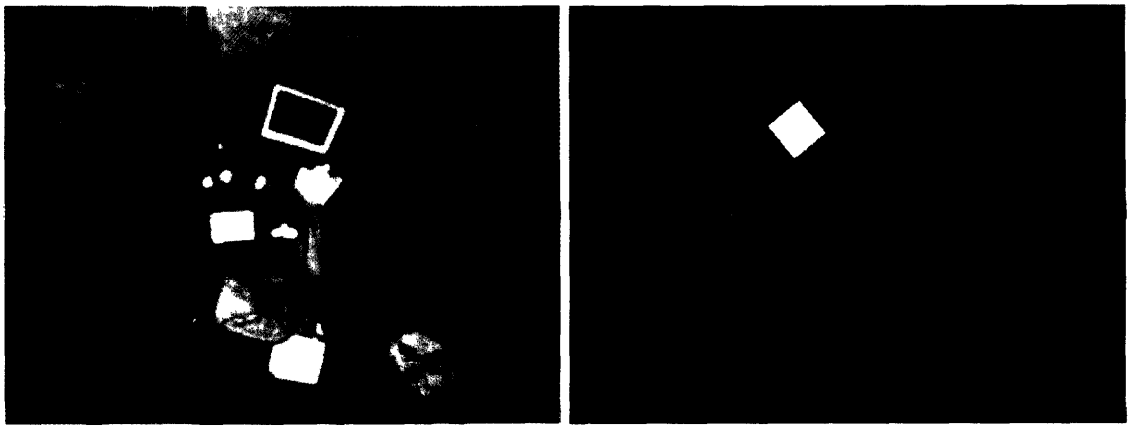


Figure 1-5: This image and kinematic visualization shows an example from data set 2. The kinematic model shown is the hand-tuned model for the latest version of Duo, which in this case matches well with the body of the wearer.

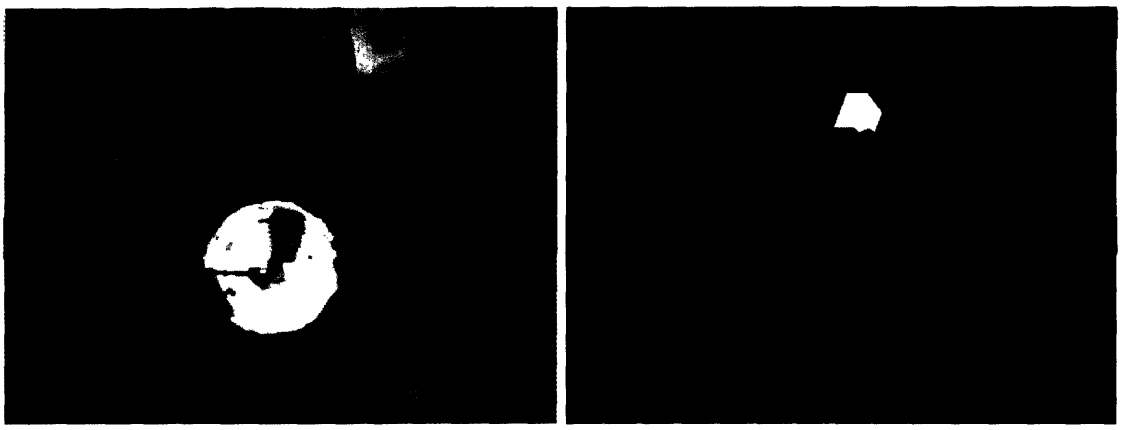


Figure 1-6: This image and kinematic visualization shows an example from data set 3. The kinematic model shown is the hand-tuned model for the latest version of Duo, which would be correct for a right handed wearer, but the wearer is left handed, which results in the incorrect position of the arm on the body of the visualized body.

1.2 The Benefits of a Kinematic Model for Wearable Systems

The primary claim of this thesis, is that a learned kinematic model of the wearer, autonomously estimated via body-mounted absolute orientation sensors and a head-mounted camera, can greatly benefit wearable systems. Within this section, we discuss several specific ways in which a learned kinematic model of the wearer can benefit wearables. First, we discuss why an autonomously learned model is to be preferred to a hand-tuned kinematic model. Second, we look at the extensive advantages of using a kinematic model that is registered with a head-mounted camera to predict the location of the hand within images captured from this camera. Third, we touch on the utility of giving wearables kinematic sensing, without considering the direct relationship between the kinematic model and the camera. Fourth, we look at the significant utility of incorporating captured kinematic data into databases that attempt to record the entire life of an individual from a first-person perspective.

1.2.1 Why Learn the Kinematic Model Autonomously?

The strongest reason for autonomously learning the kinematic model is that the estimation method is able to explicitly find and represent a mathematical relationship between the

kinematic model and the view from the head-mounted camera. This relationship allows us to use the kinematic model and the measured absolute orientations of the wearer's body parts to make predictions about the appearance of the hand within the images captured by the camera. The benefits of this estimation are thoroughly described within the next subsection. Specifying this relationship by hand would be challenging, time-consuming, and error-prone. As a by product of the estimation process, the orientation of the camera with respect to world coordinates is estimated, which opens up a number of perceptual opportunities, such as detecting edges within the image that are aligned with gravity, or the major axis of the forearm, see appendix A.

We have had some success approximately hand-tuning kinematic models for the wearer, given a particular configuration of the sensors. These hand-tuned models do not produce reasonable predictions of the hand's appearance in the image, but they do provide good models for visualization of the orientation data. We have also used them with some success in a real-time application in which the wearable would respond to the kinematic activities of the wearer, but even in this application, careful tuning was required for a single user. The resulting hand-tuned kinematic models are only judged based on their qualitative appearance, which makes it difficult to quantify how they relate to the captured data, and to maintain consistency across sessions of captured data. Tuning these models is a time consuming process that requires human intervention anytime a component of the system has changed, the wearer has changed, or greater fidelity is required. Additionally, the hand-tuned models would often require careful record keeping in order to remember the relevant aspects of the wearer and the wearable's sensor configuration, so that an appropriate hand-tuned model could be more easily created and associated with the captured data. Without knowing the sensor ordering and approximate orientations on the body parts ahead of time, hand-tuning a kinematic model could become a very daunting task.

By automating all aspects of the creation of a kinematic model, these concerns are eliminated. Record keeping becomes unnecessary, since the captured data itself holds the appropriate information. Likewise, the optimization criteria for the kinematic model becomes explicit, which allows us to better understand the model's strengths and weaknesses. In addition, errors due to improper setup, calibration, or an improper hand-tuned model are reduced or eliminated. Some similar motivations have driven the automated estimation of kinematic models for motion capture data that includes position information [47, 6], al-

though they do not have the incentive of finding a relationship between a kinematic model and a head-mounted camera. Besides giving quantitatively better performance, and greater consistency, automation gives the designer or researcher greater flexibility to test design ideas without jeopardizing the captured data. Overall, the system becomes much easier to use by both machines and people.

1.2.2 Predicting the Hand's Appearance in Images

A number of projects have demonstrated the value of monitoring the wearer's hands through a head-mounted video camera. A kinematic model autonomously estimated by the methods of this thesis, can significantly improve the reliability of hand detection in images and dramatically lower the required computational costs.

Many researchers have developed gesture-based interfaces for wearables that rely on detection of the hand in video from a head-mounted camera [41]. With gesture-based interfaces, hand activity serves as a natural way for the wearer to communicate with the wearable in mobile situations where a keyboard or mouse may be inappropriate. With sufficient perceptual sophistication, gesture-based interfaces could emulate the natural gestures used by people to communicate with one another. For example, in an augmented reality application or prosthetic memory application, the wearer might point to an object of interest within the world and receive relevant information about that object, or have that object tagged for future reference. Likewise, in a manner very similar to the real-time application we present later in this thesis, holding an object up for inspection might trigger the wearable to provide information about the object, or tag the object for future reference. Gesture-based interfaces are just one example of the many uses for visual perception of the hand in wearable computing. In Starner's work on a wearable system that interprets American Sign Language [57], the system tracked hands in video from a hat-mounted camera. The positions and sizes of the hands within the images provided much of the information used by the HMMs that recognized signing by the wearer. Likewise, from the inception of wearables, researchers have worked to create wearables that assist or instruct the wearer in manual tasks, such as airplane maintenance or equipment repair [48, 53]. Reliably perceiving hand activity would help a wearable better interpret the actions of the wearer with respect to the task, and therefore help the wearer better achieve the current goals. Finally, knowing where the hand is in the image, can help machines learn about human manipulation by

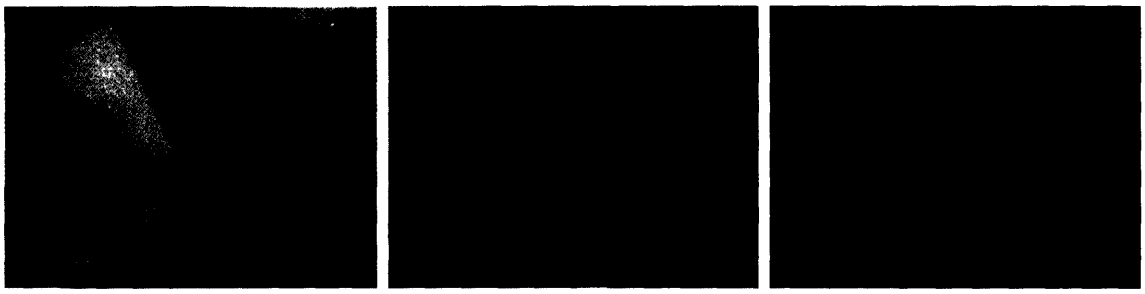
facilitating the observation of everyday manipulative actions performed by the wearer.

All of the applications mentioned within the previous paragraph require full mobility and robust hand detection in order to be practical for day to day use. However, purely vision-based hand detection methods will encounter common situations that are very challenging to interpret visually. At minimum, purely visual hand detection methods will require a great deal of computation in order to be robust with respect to large variations across users and environments. An overview of many common, yet challenging, hand detection situations follows, in the form of seven example situations each accompanied by a description and three example images. Except for one image of a blue hand, all the frames come from the three test data sets described in the previous section. In addition, all of these test frames display a white and blue circle that mark an area that the appropriate autonomously learned kinematic model expects to be near the hand in the image. These kinematically estimated hand positions are not exact, but they are of high enough quality to be very useful.:

1. Depending on the configuration of the head mounted camera, the hand may often be out of view. Determining that the wearer's hand is not visible is a challenging task, especially given the noisy, real-world images available to a wearable.

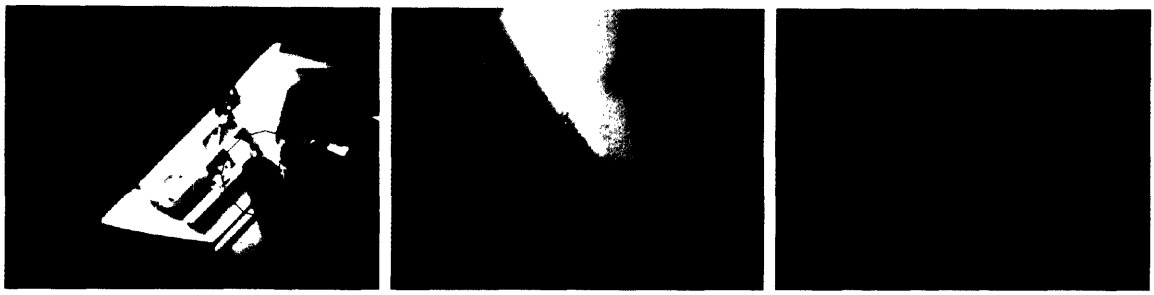
2. Humans are social, and will often be interacting with other people. Consequently, hands will frequently appear in images that are not the wearer's hands. In these situations, another person's hands may occlude the wearer's hands or be visible when the wearer's hands are out of the image.

3. Dark and noisy images that obscure the appearance of the wearer's hands are common in real-world situations, because the dynamic range of the camera is much narrower than the dynamic range of human vision. Human vision dictates the characteristics of common lighting found in human environments, so artificial mechanisms for image capture are often at a disadvantage, as indicated by the frequency with which a flash photography is required in environments that are clearly visible to the naked eye.



4. The hand is a near-field object, so its size varies over a large range when viewed from a head-mounted camera. This large variation in projected size necessitates detection methods that perform well over a wide range of scales, which usually means that the methods must perform more search in order to account for this variation in the hand's size in the image.

5. The color of the image of the hand changes based on the illumination of the environment, which along with example 6, can foil hand detectors that use skin color. The example image on the far right is the only example image that was not captured by Duo. It is indicative of a situation that has occurred a number of times in other capture sessions, where indirect light from a window can make the hand appear blue.



6. Common materials used within human environments result in images with colors that are similar to skin, which, along with example 5, can foil hand detectors that use skin color. For example, wood floors and wooden tables often produce colors that are similar to skin color.

7. During everyday activity, the hand may be obscured by the object being manipulated or covered by an article of clothing, such as a glove, mitten, sleeve, or pocket.

Tracking the hand over multiple frames and detecting the arm can help mitigate these problems. However, a kinematic model registered with the head-mounted camera circumvents these problems altogether with minimal computation by allowing the wearable to directly estimate the presence or absence of the hand within the image, and the position of the hand within the image using direct measurements of the body's configuration. For some tasks, no further hand detection will be required. For others, the kinematic model can be used to place significant constraints on the detector's search for the hand, which can lead

to drastically reduced computational requirements and the opportunity to use detection strategies that would be inadequate in isolation. Furthermore, this strategy for hand detection requires that the kinematic model be related to the camera, since otherwise we would have no way of knowing the visual implications of a particular body configuration. The methods we present within this thesis automatically estimate an explicit kinematic model and its relationship to the camera given very modest assumptions that place few constraints on the wearer's body, the placement of the sensors, and the appearance of the hand. It's worth noting that humans also make use of kinematic information to help them detect their hands, rather than relying on visual detection alone. Humans have highly refined sensory mechanisms with which they directly estimate their body's configuration [29].

1.2.3 Detecting Invisible Kinematic Activity

Besides helping a wearable detect the wearer's hand within images, a kinematic model can be used to perceive important kinematic activity that occurs outside the view of the camera. People often perform significant manipulation tasks without observing the hand, such as when searching through a pocket, holding a suitcase, or swinging a bat. Similarly, wearables can benefit from the perception of hand activity outside of the camera's view.

Researchers have convincingly demonstrated that body-mounted inertial sensors can be used to help detect various types activities, such as sitting, standing, sleeping and walking, that are important for establishing the context of the wearable's behavior [32][31]. By perceiving this contextual information, wearables can better serve the needs of the wearer [54, 9]. For example, if the wearer is sleeping, the wearable might avoid disturbing the wearer unless the notification is vitally urgent. With a kinematic model, a wearable can use these same detection methods, which are based on coarse motion estimation. But additionally, a kinematic model allows the wearable to perceive the static configuration of the body, which can lead to more robust and informative methods of detection and recognition.

The static configuration of the body can be highly informative, independent of whether or not the hand is visible in images from the camera. For example, as will be demonstrated later in this thesis, during everyday activity the hand can often be well-characterized as being in one of four distinct positions relative to the torso. One position corresponds with when the hand is at rest by the wearer's side, such as during walking activity. Another position corresponds with when the wearer is holding his hand close to his head, such as

when eating, drinking, or visually inspecting an object. A third position corresponds with holding or manipulating an object in front of the midpoint of the torso, and the final position corresponds with reaching out into the world. It's worth noting that even if a wearable could visually observe the hand in these significant positions, the perceptual system would need to compensate for the constantly changing viewing angle of the head-mounted camera in order to recognize the hand's position relative to the torso. More specifically, over a series of images, the hand might be stationary with respect to the torso, but moving significantly within the images.

It is not difficult to imagine a variety of exciting, and useful applications for wearable systems with kinematic perception. For example, a wearable could help the wearer improve at a sport or other physical task by directly monitoring the wearer's kinematic activity and giving feedback on his performance. With sophisticated kinematic sensing, as described within this thesis, an instructional wearable might at first be worn by experts performing the task so that the wearable could learn how the task should be performed. So, for instance, a wearable might first capture the golf stroke of a professional golfer while out on a real golf course performing in an actual tournament, and then later advise novices out on the same golf course as to how they could alter their swing in order to more closely match the swing of the professional. The wearable offers the advantage of both capturing expert activity and directing novice activity under real-world conditions, rather than the closed laboratory conditions often associated with motion capture systems. Clearly, these types of applications are more feasible with a system such as Duo, that can directly measure the kinematic configuration of the body, than with a system that attempts to infer the kinematic information from other sensory information. Of course, for many of these tasks, more than four absolute orientation sensors would be beneficial, so that the wearable could directly perceive more than the kinematic chain from the head to the dominant hand. Many of the methods from this thesis could be extended to a system with 10 to 12 body mounted orientation sensors. In particular, a system with 6 sensors to measure both arms would primarily require alterations to the algorithm for assigning sensors to body parts.

Direct sensing of kinematic activity that is outside the view of the camera, also presents distinctive opportunities for machines to learn about everyday human manipulation.

	images 24 hrs/day	angles 24 hrs/day
second	270KB	5.6KB
minute	16MB	340KB
hour	1GB	20MB
day	24GB	480MB
week	168GB	3GB
year	9TB	150GB
decade	90TB	1.5TB
lifetime (75 years)	4PB	12TB

Table 1.1: This table shows the approximate storage requirements for our system if run for 24 hours per day over various time scales that are relevant to human experience (of course, just a factor of 3 larger than 2.1).

1.2.4 Mining a Lifetime of Data

Capturing and analyzing a lifetime of first person experience serves as a motivating goal for a variety of research groups [20, 62, 25, 19, 33, 10, 18, 30, 40]. Most of these projects rely on information such as email, photos, GPS, first person video, and audio. In order to meet these goals in practice, very efficient and robust methods will be required for browsing and annotating the potentially vast stores of accumulated data illustrated by table 1.1. This thesis shows that augmenting first person video with absolute orientations from the wearer's body facilitates highly efficient methods for browsing and annotating first person experience. For example, if we ignore the processing involved in learning the kinematic model, then given the task of acquiring segments of the wearer's hand, kinematic processing can throw out around 95% of the video data before performing any video processing. Since, even scripted, unoptimized, kinematic processing code that estimates the hand's position in the image runs at over 1200 frames per second, this results in an enormous savings relative to trying to visually detect the hand in each frame or performing other forms of image processing on each frame of the captured database. If we take the notion of recording data 24 hours a day seriously, then we quickly realize that anything that on average runs below frame rate will result in a rapidly growing surfeit of unprocessed data, since there are only 24 hours in a day for the offline processing to spend analyzing the

previous day's recordings. Many useful image processing algorithms currently run below frame rate, so if they are to be used, they must be applied to a rapidly selected subset of the database. Kinematic data provides a powerful modality by which to make these rapid selections. More generally, we hypothesize that many of the interesting moments in life are sparsely distributed, and that kinematic information is a powerful way to efficiently find these sparsely distributed moments of interest. Unfortunately, we would need more natural data in order to appropriately investigate this hypothesis, so a thorough investigation must be saved for future work. Within this thesis, we show efficient kinematic processing that can kinematically estimate the visibility of the hand in the image, the location of the hand within the image, the 3D location of the hand in a variety of coordinate systems, as well as points in time that are likely to segment significant kinematic actions, and kinematic position clusters that meaningfully categorize the position of the hand.

1.3 Thesis Overview

Within this chapter we presented our claims and argued for their merit. In the next chapter we describe the platform Duo, which uses commercially available components to capture and process estimates of the absolute orientations of the wearer's head, torso, and arm along with video from a head-mounted camera. The wearable can run on batteries for approximately seven hours between charges, and is able to wirelessly communicate with a computing cluster for additional computational power.

In chapter 3, we present methods that use this orientation data and video to automatically estimate the assignment of sensors to body parts, the orientations of the sensors with respect to the body parts, a kinematic model of the wearer, the configuration of the camera with respect to the kinematic model, and the orientation of the camera with respect to the world. These methods allow the system to autonomously adapt to the body of the wearer.

Within chapter 4, we present methods for attention and segmentation in the kinematic and visual modalities. The attention system directs the visual segmentation system based on both visual and kinematic information, such as the estimated position of the hand. We describe methods for kinematic segmentation, visual attention, and visual segmentation, including a new interest point detector with associated shape descriptors.

For chapter 5, we present results from autonomous exploration of the structure within

kinematic segments and visual segments collected from real data. Hand positions associated with the kinematic segmentation points strongly cluster around a few locations when viewed from the proper coordinate system. Segments collected using a visual attention system specialized for the detection of hand activity cluster into hand segments and segments from manipulated objects, as well as some background clutter. Tracking visual segments over time finds salient and related sets of segments.

Then in chapter 6, we discuss the system we have developed for rapid, machine assisted annotation and browsing of databases of captured experiences. This system facilitates offline cooperation between a human assistant and autonomous processes for learning and perception.

In chapter 7, we describe a demo application of a real-time wearable system that facilitates real-time cooperation between a human assistant and the wearable through tightly coupled behaviors linked via kinematic sensing.

Finally, in chapter 8, we conclude with a summary of this thesis and a reiteration of our claims.

Chapter 2

The Platform

The platform is named Duo in order to emphasize cooperation between the human and the wearable. Duo is strictly a research platform and as such emphasizes technical simplicity, ease of servicing, and functionality over other design considerations important to wearable computing, such as comfort and style. Technologies that merge sophisticated sensing apparatus and computers with the human body are still in their infancy. Likewise, standard software infrastructures for such systems have yet to emerge. As much as possible we have attempted to use commercially available hardware and standard open source software to create our platform. Despite these efforts to simplify development, a substantial amount of work went into various iterations of both the hardware and software infrastructure. In this chapter we describe the latest version of the platform, the constraints that influenced the design, and some of the earlier versions of the platform.

2.1 Hardware

The most recent Duo platform is a fully mobile wearable system that integrates kinematic and visual sensing, connects wirelessly to off-board computation, and can communicate with the user by speech, see figures 1-1 and 1-2. Duo's sensors consist of a head-mounted Firewire camera and 4 absolute orientation sensors. These sensory systems are connected to a laptop computer mounted on a backpack. The backpack also contains rechargeable batteries that support mobility by providing power to the laptop, camera, orientation sensors, and other peripherals for approximately seven hours between charges. The laptop can wirelessly communicates with a dedicated cluster of computers via 802.11b. With the

increasing availability of economical broadband wireless connectivity this type of system may someday function over an entire city, but for now the system is primarily used inside a home with a dedicated wireless network. In contrast to wearable systems designed solely for data capture and offline processing [10], Duo's design has been motivated by applications that require computationally intensive real-time processing of the sensory input to support relevant communication via speech. The computer cluster facilitates intensive real-time sensory processing. When unable to connect with the cluster Duo can either shut down, perform more limited perceptual processing, or just capture the sensory data to the hard disk for offline processing.

In building the system we used commercially available components as much as possible with the belief that this would reduce the cost of the system, simplify its construction, and ease maintenance. The first version of Duo was incrementally fabricated and frequently in flux. As such, the methods of construction included inelegant materials such as duct-tape and cable ties. Although it was useful, this version of Duo was inconvenient in a number of ways. For examples, accessing the laptop was difficult, recharging and powering the system required changing multiple connections, the cables for sensing and power were easily tangled, and the camera mount was difficult to adjust. The latest version of Duo corrected most of these problems, while using most of the same internal components.

2.1.1 Acknowledgments

The physical design and construction for the latest version of Duo resulted from a collaboration with Jeff Weber, a very talented robotics engineer working in Prof. Rod Brook's lab. The custom power conversion and distribution board for the latest version of Duo resulted from a collaboration with graduate student, and electronics expert, Eduardo Torres-Jara. Eduardo Torres-Jara also helped significantly with the design of the circuitry necessary for the LED array to function, which is described in chapter 7.

2.1.2 The Components

The Camera

The vision system uses a Dragonfly IEEE1394 camera from Point Grey Research that uses 1/3" CCD to produce uncompressed progressive scan images of 640x480 8bit pixels color

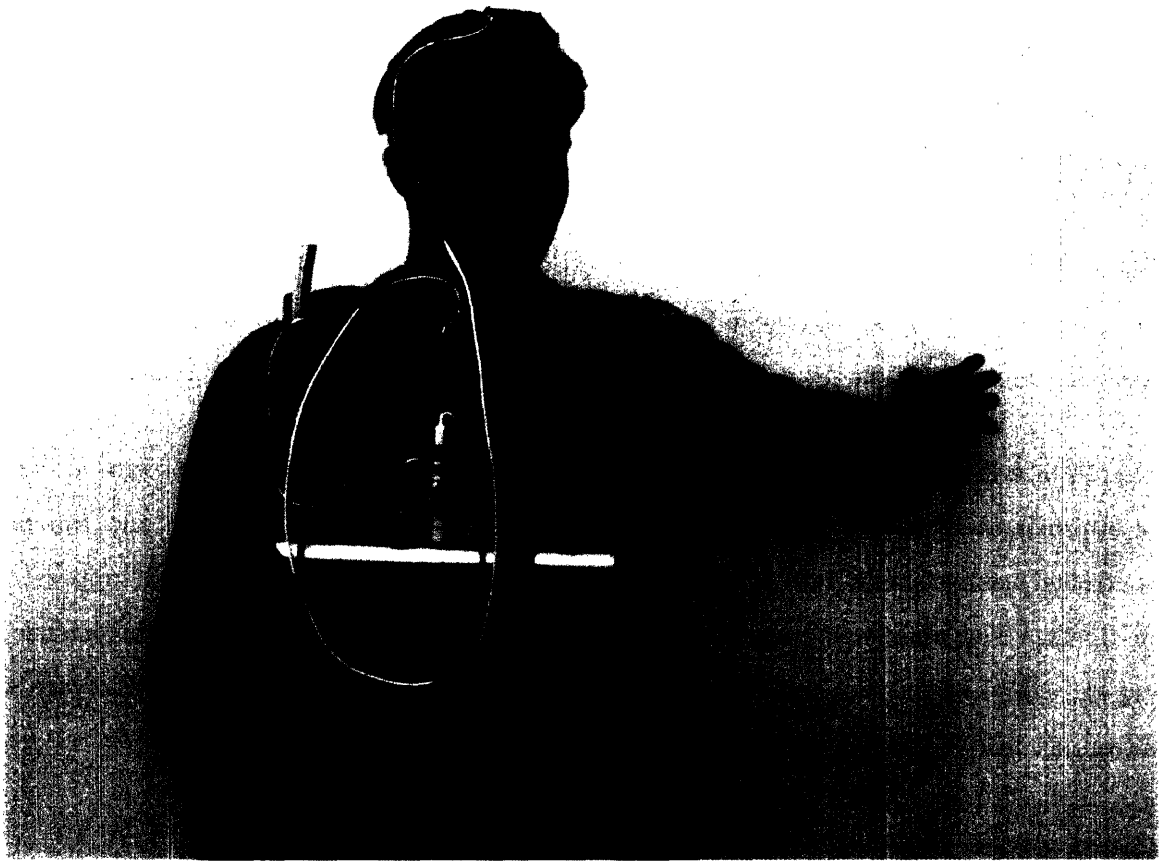


Figure 2-1: The latest version of the wearable system Duo, viewed from behind.

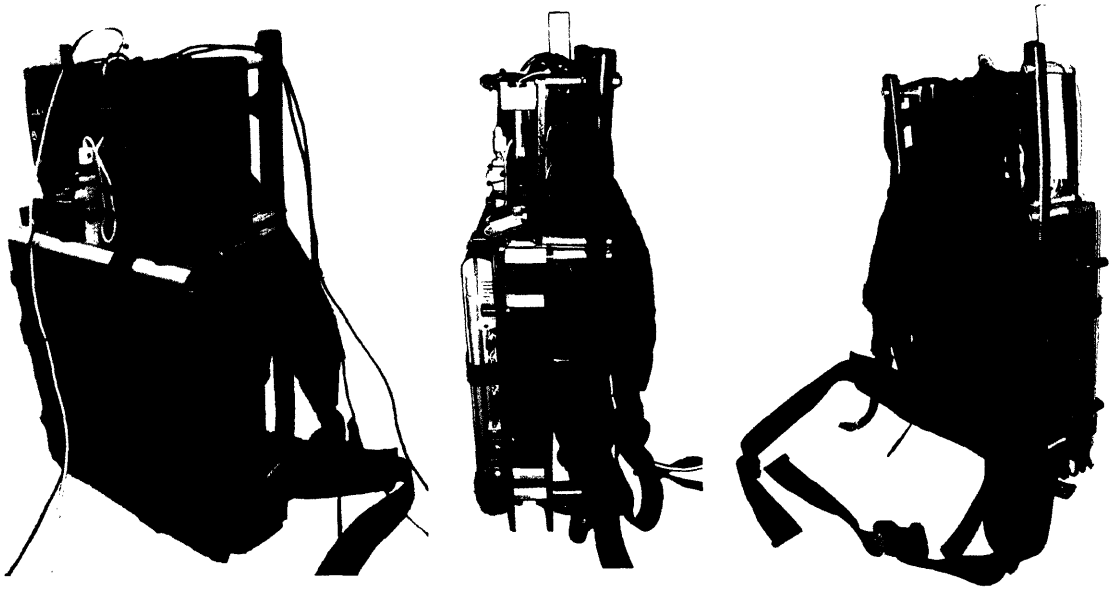


Figure 2-2: Three views of the most recent version of the wearable system, Duo. The edges of the three batteries are visible in the center image of Duo's profile. The construction used a small aluminum frame from The Long Trail Jr. by Kelty, a backpack for kids.

filtered in a Bayer pattern at 30 frames per second. We use a wide angle lens with a view of approximately 90 degrees. The camera produces images of good quality and can be controlled using version 1.30 of the digital camera specification, which is well supported in Linux. The ability to turn off auto-adjustment of the gain, shutter, white balance, exposure, and brightness, and efficiently control and monitor these settings over Firewire is advantageous for machine vision.

Orientation Sensors

Each orientation sensor is an InertiaCube2 made by the company InterSense. The company claims that they have an accuracy of 1 degree RMS and an angular resolution of 0.01 degree RMS. We have not attempted to verify these claims, but we have found their performance to be suitable for our application. Each cube is 28.29mm x 24.38mm x 33.91mm. They are still much too large to go unnoticed when attached to one's arms. Each of the orientation sensors estimates its absolute orientation in the world by combining MEMs based inertial measurements with gravimetric and magnetic measurements at around 180Hz. The gravimetric and magnetic measurements provide an absolute reference frame with which the

device compensates for drift from integrating the inertial estimates. One serious draw back of these sensors is that their orientation estimates depend on the Earth's magnetic field and consequently can be disturbed by interference from other magnetic fields and some large metal objects. In practice, we have found that this problem does not happen very often, but when it does occur the orientation estimates can be nearly worthless, particularly for the kinematic estimates on which our system depends. Dramatically incorrect orientations are easy to detect, so a sensible way to deal with this is to detect these obviously incorrect sensor estimates using statistics from normal human motion, and inform the wearer or the offline processing system when the problem occurs. The wearer then has the option of avoiding these problematic situations, which are usually tied to particular locations in the world.

Communication

For visual depictions of the the power and communication systems for Duo, refer to figures 2-3 and 2-4. The laptop communicates with the four orientation sensors through a USB connection to a Keyspan USB 4-port serial adapter (Part# USA-49WLC). The rate of orientation data is relatively small at around 100 Hz * (4 sensors * 3 orientations) * (1 float) = 100*4*3* (4 bytes)= 4800KB/s = 38.4Kb/s. The camera connects directly to the laptop Firewire port for communication, although it is not powered from the laptop. The rate of uncompressed image data is much more formidable at 320x240 pixels * 3 bytes * 15 Hz = 3.456 MB/s = 27M.648 Mb/s. Since the image data must flow through the small wireless pipe, and we wish to store many hours worth of images, we compress each image independently as a JPEG image using the ImageMagick++ libraries and a quality of 80. This results in an image typically of a size around 18KB, leading to a total bandwidth of approximately 18KB * 15Hz = 270KB/s = 2.16Mb/s. Although independently compressing each image requires more storage and bandwidth than video compression methods that use correlations across time, individual images are more convenient to work with since they allow rapid random access to individual frames. Time-based compression requires that more than a single frame be accessed in order to decompress the frame of interest, and often requires that a key frame and a series of frames be uncompressed in order access the frame of interest. Using individual JPEG compressed images for video is essentially the MJPEG format frequently used with DV cameras, although we store each image as a

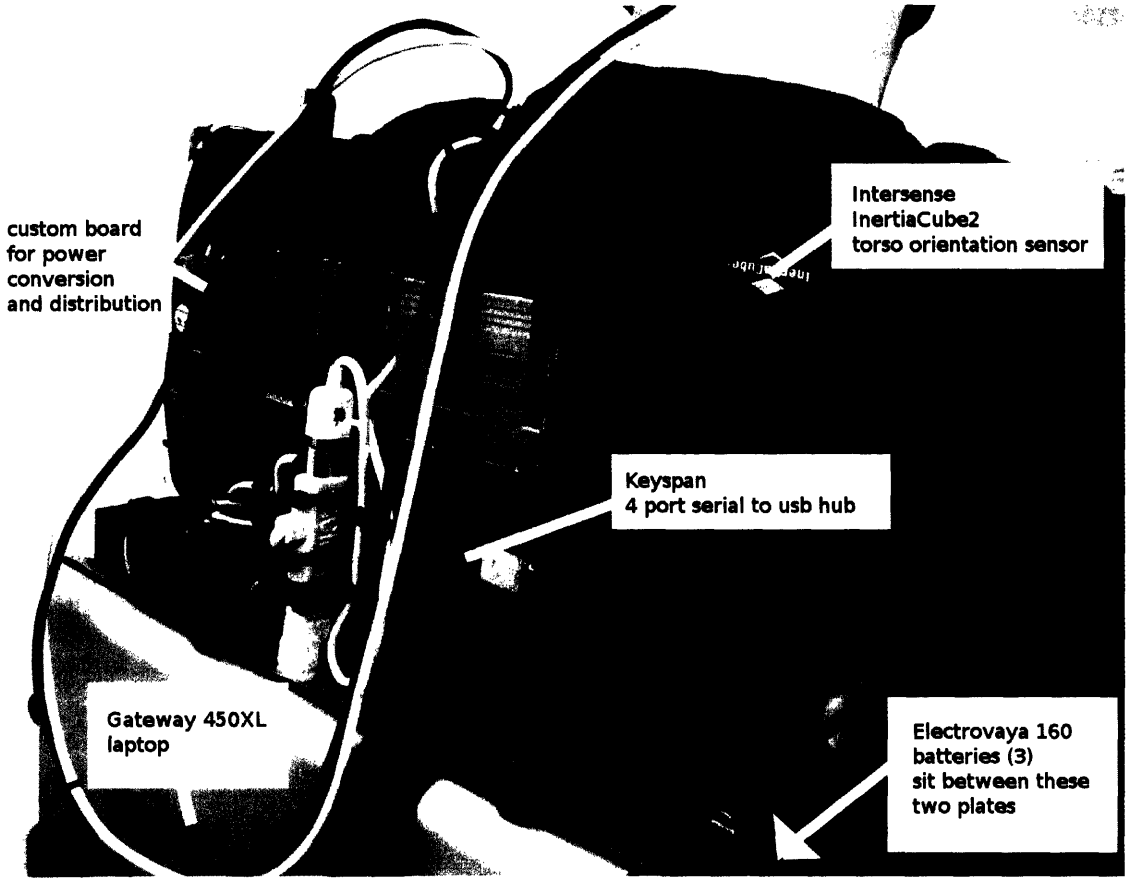


Figure 2-3: A closeup view of the back of the platform with labels for several components. The holes in the board simplify mounting devices and cables.

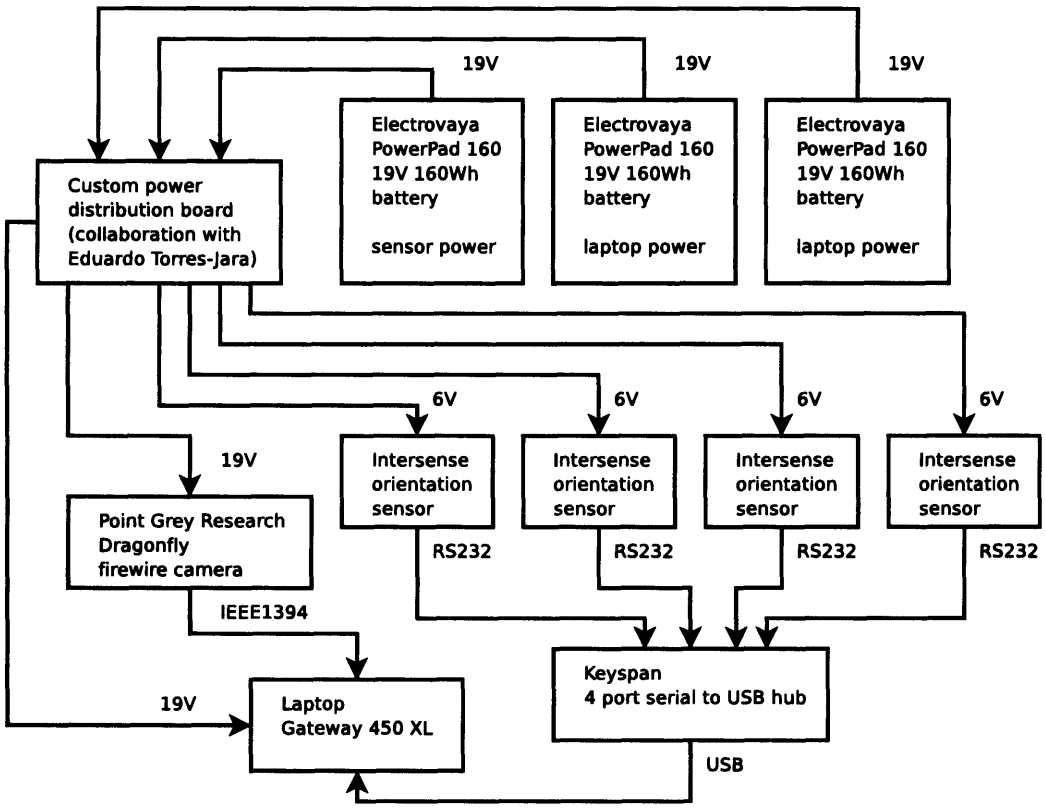


Figure 2-4: A schematic showing the major components and interconnections for the wearable system, Duo.

distinct file with a file name that encodes the image's time stamp.

Power

A schematic in figure 2-4 shows most of the power distribution and conversion for Duo. The Keyspan 4-port serial adapter is powered from the laptop through the USB connection. The four IntertiaCube2 sensors, which are each rated at 100mA at 6VDC for a total of 2.4W, and the camera, which is rated at less than 2W, are powered by an Electrovaya 160 battery through a custom power conversion and distribution board. The Electrovaya battery is rated to have 160Wh at 19V. The custom board has a DC to DC converter that takes the 19VDC battery output and provides 6VDC for the four sensors in parallel. The Firewire camera is powered directly from the same battery, since the IEEE1394 standard allows for a wide range of voltages. In an earlier version, the orientation sensors were powered by two batteries from DigitalCameraBattery.com with each battery powering two InertiaCube2's. InterSense sells simple, but very expensive D-cell battery packs for mobile operation, so the battery power we provide through the power distribution board is appropriate. The InertiaCube2's are the most expensive components of the platform, since the technology has not yet been commoditized. We expect this to change fairly rapidly, since sensor technology of this sort has a wide variety of applications and is beginning to show up in a variety of products, such as 3D mice and medical devices. The laptop which is based on Intel's Centrino chipset is rated at 19V 4.74A for 90W. The laptop uses most of the system's power through two additional Electrovaya 160 batteries. The output from these batteries is combined on the custom power board through two diodes. We removed the laptop's original batteries, so as not to load the external batteries and to avoid the higher weight to power ratio of the standard laptop batteries.

Computation

A laptop serves as the mobile computing portion of the platform. A Gateway 450XL laptop with 1GB of RAM, a 1.6GHz Pentium M processor, a 60GB hard drive, a Firewire connector, a PCMCIA slot with an Enterasys 802.11b lucent wavelan driver compatible card, and a USB port performs all of the data capture, processing, wireless communication, and local storage for the platform. Using a laptop has great advantages for research platforms as opposed to more traditional embedded computers, since consumer laptops tend to be well-

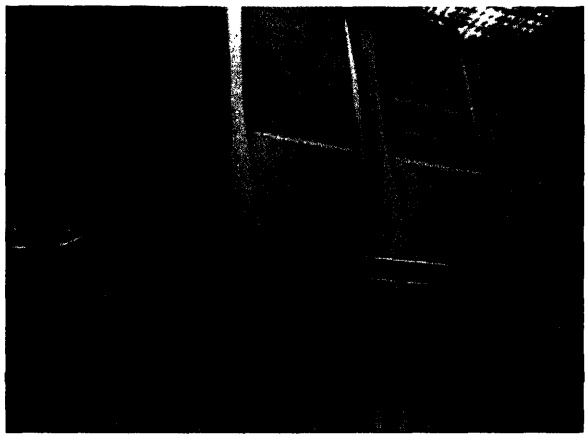


Figure 2-5: The original computer cluster consisted of eleven machines some of which are shown in the picture. These machines were 600MHz and 1GHz Intel PIIIs inherited from another project. Many of these machines eventually failed due to bad motherboards. The current computer cluster is much more compact and a little more powerful.

supported by the open-source community, low cost, and fully configured with ethernet, a hard drive, battery power, low-power processors, and many peripheral interfaces. When we originally acquired the laptop for the system, we wished to maximize the processing power and the options for peripheral interfaces. The selection was also influenced by our plan to use the laptop for day to day work, which was a bad idea and should be avoided if one has the resources. If we were to redesign the system, we would almost certainly use a smaller laptop.

Wireless connectivity to a small cluster of computers goes through a Netgear WGT624v2 Wireless firewall router which is configured to solely serve as an access point providing 802.11b service for the entire apartment and the immediately surrounding area. When the platform is within range, it can take advantage of the computing power for real time behaviors and visualization. When outside of range, it can store the data it collects on its local hard drive for off-line processing.

The computer cluster currently consists of 4 machines with battery backup. Two machines are 3200+ AMD 64 small form factor, headless, Shuttle boxes, each with 2GB of RAM. They are relatively inexpensive, take little space, consume relatively little power at 250W, are quiet, and have good performance. The cluster also includes a generic 3.0 GHz PIV based desktop with 2GB of RAM, and a laptop with a 1.8GHz Pentium M processor

Figure 2-6: The older versions of Duo used a standard fabric backpack to hold the laptop and other equipment. The image on the left shows the internal equipment for this version of Duo, which consisted of batteries, an IEEE 1394 hub and a USB to 4-port serial converter all mounted on a large flat battery with dimensions similar to the laptop. This equipment was placed within the backpack first with the back of the large flat battery next to the wearer's back. Within the image on the right the laptop is visible through ventilation holes cut out of the backpack's sides. This image shows the system just prior to being dismantled.

and 1GB of RAM, which we also use for development. The original cluster is described and pictured in figure 2-5.

Internals of the Previous Version

In the first versions of Duo, the camera was powered through a Firewire hub, which was powered by an independent 12VDC NiMH Powerbase battery rated at 7500mAH and designed to power laptops. The hub and the battery sat in the backpack underneath the laptop. The Powerbase battery served as the structural backing for the equipment in the backpack and had a similar profile to the laptop with dimensions of 11.8"x8.9"x0.82". The hub could connect up to three Firewire cameras to the laptop through a small 4-pin unpowered IEEE1394 connector, as is common on many laptops today. In spite of initial ambitions to use two cameras, in practice we only used a single camera on the system at a time, and consequently did not add a hub to the latest version of Duo.

2.1.3 Mounting Difficulties

Mounting sensing devices on the wearer presents a serious challenge for wearable systems. The sensory systems for robots and animals typically do not need to be removed each day, and are an integral part of the body's design. In contrast, at the present time few people would be willing to have their bodies altered in order to better affix a camera. Likewise, comfort, appearance, and ease of use place practical constraints on mounting methods. With our wearable platform we can not expect the position and orientation of a sensor to have the same consistency and precision across days as a robot's sensor or a person's perceptual organs. Moreover, during the day, "motion noise" must be expected with wearable sensors as they are likely to move independently of the body to some extent due to inertia and imperfect physical coupling. Being placed on the outside of the body also makes wearable sensors more vulnerable to being jostled by contact with the environment, especially since the wearer will not have had a lifetime to learn how to maneuver with them. In addition, the wearer may be compelled to adjust the sensor for comfort or end up fidgeting with the sensor, as people often do with jewelry and other articles. When mounting the sensors we strive to minimize this variability in position and orientation. However, as we discuss, the remaining variability has strong implications on the types of algorithms we use.

Camera Mounting: Glasses, Hat, and Band

Over the course of Duo's development we have used three different methods to mount the camera and orientation sensor to the wearer's head: glasses, a baseball cap, and a head band (see figures 2-7 and 2-8). Several design issues played a role in this progression. Most importantly, the camera needs a good view of the workspace of the dominant hand, and the camera and orientation sensor should be in rigid alignment with the wearer's head and each other. In addition, we would like for a wide variety of wearers to be able to use the system. Other desirable properties for the head gear are comfort, ease of donning, and repeatability of the sensor configuration with respect to the head. One complicating factor specific to the camera we used, is that the small remote head of the camera must stay near a control board which is relatively large and requires a Firewire cable attaching it to the laptop.

In the first prototype shown on the left side of figure 2-7, we mounted the camera on eye glasses, and did not use a head-mounted orientation sensor. As will be made clear in the next

Figure 2-7: This figure shows the first two versions of Duo's head gear. On the left, the camera is mounted on the glasses and no orientation sensor is used. On the right, the camera is mounted on the brim of the baseball cap and the orientation sensor is at the back of the head, but not visible. The baseball cap version is shown just prior to being dismantled, and is noticeably tattered from use and attempts to find ways of reinforcing the camera mount in an adjustable way.

Figure 2-8: This figure shows the latest version of Duo's head gear. On the left, the blue orientation sensor is visible. The black plastic rectangle which holds this sensor originally served as the battery case for the head lamp. On the right, the camera and the camera board are visible. The camera is mounted on a one degree of freedom pivot, which originally held the lamp.

chapter, the head mounted orientation sensor proved to be very useful. For the next revision shown on the right side of figure 2-7, we mounted the camera on the brim of a baseball cap and the orientation sensor on the back rim of the hat. The position on the brim of the hat provided a better perspective on the workspace of the wearer's dominant hand by allowing the camera to be positioned out in front of the face with a slight downward tilt. A hat was also more accommodating for additional equipment of various sizes and weights, such as an LED array described in figures 7-1 and 7-2, and the orientation sensor, which we wanted to be closely coupled with the camera. We found the hat to be more comfortable as well, especially with the asymmetrically distributed weight of the camera and orientation sensor along with the standard thick Firewire cable we initially used. The Firewire and orientation sensor cables trailed off of the head at the back and center of the hat, which kept them out of the way during head movement and helped balance the hat. When compared to glasses, one weakness of the hat as a camera is that it's position is less constrained. Glasses have three points of contact which successfully reduce the effective degrees of freedom along which the glasses, and hence the camera, can move. If appropriately fitted, glasses can be fairly consistent in their orientation and position, with motion up and down the nose being the most likely degree of freedom. On the other hand, even a tightly fitting hat tends to have three degrees of freedom, in the manner of a ball and socket joint with the head being the ball and the hat being the socket. A snugly fitting hat with a clear forward direction helps compensate for the degrees of freedom, but is not as consistent as a mechanical constraint.

For the final version of the head gear shown in figure 2-8, we modified a commercially available Gemini headlamp from Black Diamond designed for hikers, spelunkers, and others who would like light to project from their forehead toward the area they are observing. We replaced the forehead located lamp with a camera and the battery pack at the back of the head with an orientation sensor. We mounted the awkward camera board on the top band. As shown in figure 1-1, some hats can be worn on top of the head gear for warmth or aesthetics. With the baseball cap head-gear, reliably adjusting the view of the camera was a significant challenge. Because the camera was located on the side of the baseball cap's brim, adjustments over all three orientation axes, and possibly position, would be required to fine-tune the camera's wedge of visibility. Likewise, the camera would require a different configuration for a left-handed wearer and might have difficulty observing the left arm's actions. Moving the camera to the center of the brim might have helped with these

problems. In the final head-band version, the camera is in the center of the forehead and only requires a single degree of freedom to effectively adjust the viewpoint. Also, the camera's viewpoint more closely relates to the perspective of the wearer, which can be advantageous for applications that involve capturing and interpreting the wearer's experiences. The head band does a good job of rigidly coupling the camera and the orientation sensor with the wearer's head by keeping the sensors compressed against the wearer's skull. During a session, the orientation sensor and camera are kept in rigid alignment with each other through the wearer's head, but their relative orientations and positions are likely to change between sessions. With the methods we present within the next chapter, we are able to compensate for these between session changes. The wearer does need to be careful not to apply too much pressure with the camera over long periods of time, since this can lead to some irritation of the skin underneath the camera. We eventually added extra padding underneath the camera plate in order to reduce irritation. Unlike the baseball cap, the head band can be easily adjusted to fit different wearers.

Body Orientation

In addition to the head-mounted orientation sensor described above, three orientation sensors are used to measure the absolute orientation of the torso, upper arm, and forearm. The clearest and easiest location for mounting an orientation sensor is on the wrist in the same manner that a watch is worn. The top of the wrist tends to be flat and have less compliance than other parts of the body, which makes it well suited to forming a stable coupling between the orientation sensor and the forearm. Additionally the location is comfortable, easy to use, and fairly repeatable due to landmarks such as the hand, the flat top of the wrist, and the wrist's bony structure. A number of the results from this thesis might be applicable to a system consisting of a single orientation sensor on the wrist and a head mounted camera with an orientation sensor. For the upper arm, we use an arm band, which is a commonly used method of wearing consumer devices, such as portable music players during exercise. Achieving a stable coupling between the orientation sensor and the upper arm is difficult because of the substantial soft tissue surrounding the bone and the ease with which this tissue can be moved. In particular, twisting motion around the bone occurs without much force. Fortunately, our work primarily uses the estimated orientation of the major axis of the upper arm which is less sensitive to these twisting motions. The

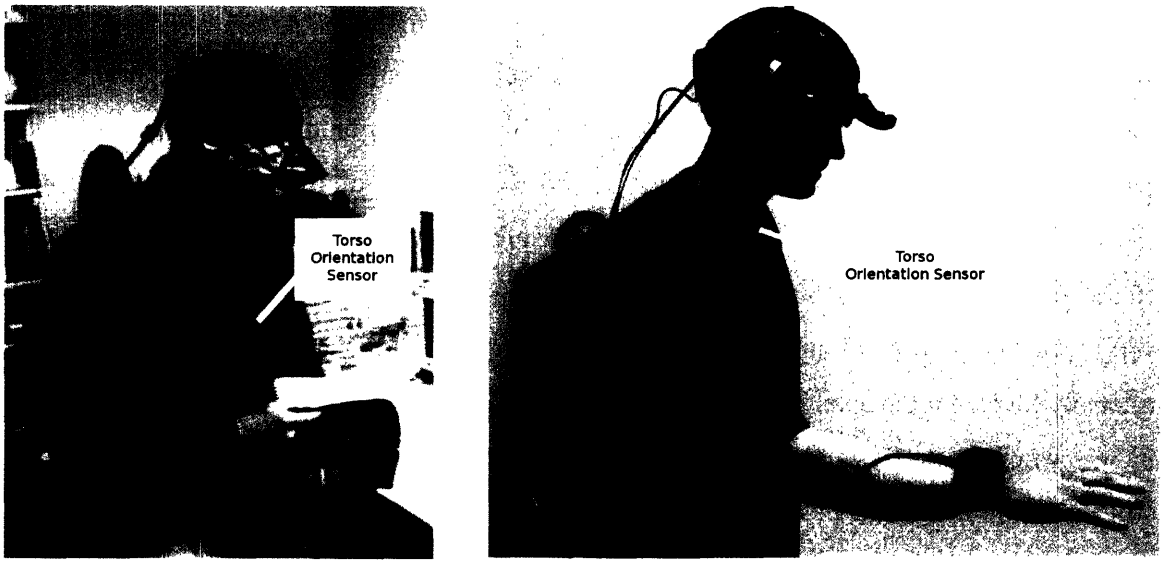


Figure 2-9: This figure shows how the torso orientation sensor was mounted in previous versions of Duo. On the left, the torso orientation sensor is attached using an elastic band in the same way that the wrist and upper arm sensors are attached. On the right, the torso orientation sensor is mounted on the right backpack shoulder strap. The sensors can be worn underneath or above clothing.

more uniform structure of the upper arm also makes consistent placement of the arm band more difficult, which gives additional motivation for the methods of autonomous adaption we present in the next chapter.

The initial wrist and arm bands used elastic bands of appropriate length with Velcro fasteners taken from exercise equipment. They were somewhat uncomfortable, since they needed to be wrapped tightly in order to securely fasten the sensors, which were directly attached to the elastic material and held in place by the force of the elastic band wrapped around the sensor and arm. For the latest version of Duo, the sensors are securely mounted to a padded plate through which wider and more comfortable elastic material is threaded. The width of the plate helps take the pressure off of the arm, and the threading makes the bands easier to put on, since the wearer only needs to put his arm through the bands and then cinch them.

During Duo's development, we have used three different methods for mounting the torso orientation sensor. Initially we used a long band wrapped around the torso with the orientation sensor sitting on the sternum, as shown on the left side of figure 2-9. We

next affixed the orientation sensor onto the upper part of the backpack's shoulder strap, as shown on the right side of figure 2-9. In the final version, we attached the orientation sensor to the rigid part of the backpack in the far corner to help avoid electro-magnetic interference, as shown in figure 2-3. The torso is a very flexible body part that is able to bend and twist in complicated ways. Consequently, what constitutes the orientation of the torso is not well-defined. At the same time, the general orientation of the torso is very informative about human activity, since changes in major activities and locomotion typically involve large changes in torso orientation. The orientation of the torso tends to be indicative of the contextual orientation of the body during activities, remaining relatively still while the arms and head move around. Intuitively, keeping the torso twisted for long periods is uncomfortable, and it is a large mass, so significantly changing its orientation is a non-trivial mechanical process. Besides these coarse measurements of torso orientation, the system is concerned with modeling the mechanical chain between the head-mounted camera to the hand. For this estimate the upper part of the torso is most relevant, so the three methods of mounting have focused on the upper part of the torso. Another source of unmodeled complexity worth mentioning is in the shoulders, which have a surprisingly large range of independent motion.

2.2 Software

An extensive software infrastructure supports research on Duo. The code base, most of which will eventually be released as open source, consists of custom code for inter-process communication, distributed computing, database backed storage and retrieval of captured data, tools for data exploration and visualization, tools for annotation, perceptual processing, and more. As with the hardware, the software infrastructure has evolved significantly over time. Initially everything was coded in C++, including custom software for distributed computation, and Matlab was used for interactive data analysis. Today the majority of the non-vision code base uses Python, while critical vision code and a few other components are written in C++ with Python interfaces using Swig.

2.2.1 OS

The wearable system and all the machines in the cluster run the testing version of the Debian distribution of GNU/Linux, named Sarge. The laptop used in the wearable runs a custom compiled kernel, version 2.4.27, in order to include the firmware for the Keyspan four port serial device, which is not included in the stock Debian kernels. All the other machines run stock Debian kernels, and the cluster nodes run the most recent 2.6 series kernel released by Debian. Nearly all of the software from outside parties that our software uses exists as Debian packages on official Debian mirrors and unofficial package repositories and can thus be easily downloaded, installed, and maintained through the Debian package management system. Consequently, the majority of the software required to integrate a new machine into the cluster can be installed and configured using a script that makes use of the command apt-get, which automates most of the process of installing and configuring software from a local repository or over the internet. This type of infrastructure provided by Debian, as well as the ease of customization, has been extremely useful.

2.2.2 Laptop Software

The software on the wearable laptop can be minimal or a complete development environment. We have run the system with a minimal setup consisting of essential services and no graphical display, which was sufficient with the older version of Duo, since we worked to minimize the need to remove the laptop from the backpack. With the current version, we keep a complete development environment on the laptop, which is convenient for development since the latest design for Duo makes the laptop easily accessible, as shown in figure 2-10. On boot it seeks out the wireless network and starts up the wearable software, which includes an image server and kinematic server. Maintenance and development can be performed over ssh. Each machine participating in the application has it's own identical copy of the entire software directory tree. A script in the clustering software is uses rsync to synchronize the development version of the wearable software directory tree across the wearable laptop and machines in the cluster. This approach is used instead of remotely mounting the development directory in order to minimize network traffic, ease setup, increase robustness, and allow the wearable laptop to function away from the cluster using its own copy of the software, since the cluster is at times unreachable.

Figure 2-10: The laptop can be used by taking the backpack off, undoing three Velcro straps, and opening the lid of the laptop. This is convenient, since it makes a full development environment mobile, and allows for implementation and testing in the wild.

2.2.3 Networking

The latest networking and clustering software is based on Twisted, a powerful open source Python framework for event-driven networking (asynchronous). Interprocess communication is performed via sockets, even when the processes are on the same machine. Sockets are less efficient for communication on the same machine than methods such as shared memory, but the overhead is acceptable for the coarse grained parallelism used in our application. Sockets offer the benefit of a very general and well-supported method of interprocess communication, which significantly assists development. The four machine cluster is connected via a 100 Mbps switch. One of the four machines in the cluster contains two ethernet cards and serves as a gateway between the wired network and the wireless network. Currently, a DNS server and DHCP server run on the wired network in order to provide name resolution and IP addresses to the machines on both the wired and wireless network.

2.2.4 Development

We use emacs, ipython and the bash command line as our development environment. We originally wrote the clustering code and all of the wearable code in C++ along with a few bash scripts. Although the code was fast, development and maintenance was costly due to the coding inefficiency of C++ and the long code, compile, test loop. Consequently, we gradually integrated the use of a high-level-language into the code-base to facilitate rapid development and scripting. We chose Python because it promotes rapid development, is well-supported with a variety of libraries, is easily installed through Debian, interfaces well with C and C++, is open source, has a clean and readable syntax, and fits well with both functional and object-oriented programming styles. Moreover, with the interactive environment ipython and the matplotlib, scipy and numarray libraries for mathematical processing and visualization, Python does a good job providing much of the same functionality as Matlab in a manner that is suitable for both offline exploratory analysis, and real-time processing. We frequently use the same interpreted code for online real-time processing and offline analysis.

2.2.5 Libraries

Besides the standard libraries for C++ and Python development on GNU/Linux, our software makes use of a number of more specialized open source libraries. The vision system uses several efficient low-level operations provided by OpenCV, a computer vision library written in C and sponsored by Intel, [12]. The SDL libraries and Pygame interface to them are used for visualization of video and OpenGL models in C++ and Python. A number of components of the Boost C++ libraries have been helpful. The SWIG libraries were used to generate Python interfaces for our C++ code. Image conversion, including JPEG compression for wireless image transmission and data capture, uses the ImageMagick libraries and C++ API. The Python Imaging Library (PIL) has provided some basic image processing algorithms. The libdc1394 and libraw1394 libraries are used to interface with the Firewire camera. Pexpect has provided useful functionality for interfacing with interactive command-line programs. We make use of Numarray, SciPy, and Scientific Python for various numerical computations, and Matplotlib for Matlab style visualization. The Orange C++ and Python libraries have provided useful machine learning tools. Our C++ gui applications use GTKmm. For XML generation and parsing in C++ we use libxml++. The flite C API and command line application are used to convert text to speech. SQLite 3.0 is an embedded database for the system.

2.2.6 Wearable Servers

Three main servers run independently on the wearable laptop in order to provide compressed frames of video, samples from the orientation sensors, and speech services. All three servers provide their services upon request, rather than pushing their data out to their clients. We found that requiring an explicit request for each frame of video and orientation sample added negligible latency, since the request could be made by the client prior to when it actually is ready for new data. Requiring explicit requests has the benefit of controlling the flow of data and allowing the clients to make more refined requests, such as for subimages or lower-resolution images. In addition, requests for metadata, such as the current number of connections, or requests to change the server's parameter, such as requiring that each image be sent to all clients, are easy to add to the protocol. Each type of captured data also includes a time stamp, which is important upstream for registering multi-modal data,

properly handling time dependent signal processing operations, and searching the database. The angle server also throttles its output to be a maximum of 100Hz. The time stamps use the UNIX convention of seconds from the epoch. The speech server receives requests for speech and executes the request using the command-line version of flite, an open-source text to speech package.

The video and orientation servers make use of the basic Twisted infrastructure for clients and servers, which entails client factories and server factories that create a sender or receiver for each new connection. We created a generic server factory that takes a sample iterator and a parser as input. The sample iterator returns a sample and a sample name for each call to its "next" method, while the parser has a "write" method that when given a transport, a sample, and a sample name as input, sends the sample and sample name out on the transport in the appropriate format for the protocol. This generic definition allows us to easily create and maintain new servers. The video and orientation servers use this same generic server factory, but with different sample iterators and parsers. The video server uses an iterator that grabs frames, while the orientation server uses an iterator that grabs orientations. Moreover, we have created iterators that grab pre-captured images and orientation data from the disk or a database. The image server can also serve generic images from a directory. This allows us to easily run the same wearable application on previously captured data from the wearable system, or any image collection of interest. The speech server uses the Twisted distributed object model called Spread. Clients call the appropriate method on this remote object in order to have the application talk to the wearer.

2.2.7 Storage

Data capture clients can be run on the wearable laptop or the cluster. The angle capture client and image capture client run as separate processes. The angle capture client saves the angles and associated time stamps it receives to a single binary file by continually appending them. The format for the binary file is compact and is designed to be easily recovered in the event that the saving process is interrupted unexpectedly. The image capture client saves each raw JPEG encoded binary image it receives directly to a file without decoding the image, and uses the image's time stamp for the image's file name. The file name simply replaces the decimal point of the time stamp with an underscore to give a series of image names such as the three images 1082513949_110296.jpg, 1082513949_204026.jpg,

1082513949_297521.jpg captured on April 20th, 2004 at a rate of approximately 10Hz. We use the ReiserFS filesystem, which is particularly well-suited to efficiently storing large numbers of small files.

After a capture session, we connect the wearable to the wired network and run a script that transfers the captured data to a machine on the cluster, puts the data into a standard directory structure, and updates the SQLite database. Each capture session is stored under a subdirectory labeled by the start time and date of the capture session in a human readable format, such as the directory 200404202219 used to hold the session captured on April 20th, 2004. This naming scheme is sufficient for our purposes since there is only a single capture device. This global SQL database holds the captured orientations indexed by their time stamps and the fully qualified image file names indexed by their time stamps. This SQL database consists of at least three tables, one of which holds the captured angles from all sessions, another that holds all the image file names from all sessions, and another that holds session specific information. Although the angles are stored within the database, we keep the original capture files as backup. Only the file names of the images are stored within the database, and the ReiserFS filesystem handles the actual storage of the data. The database can be easily reconstructed using a script, which also looks for the presence of new session directories and the absence of previously existing session directories in order to help maintain the integrity of the database. If a session directory is missing, the script gives the user the option to delete the associated data from the database. If a new session directory is found, its data is added to the database.

Given the previously discussed bandwidth requirements, we can compute some approximate storage requirements for our system over various time scales relevant to human experience, see table 2.1 and 1.1.

2.3 The Future Duo

Duo started out as a platform for investigating real-time interaction between a kinematically perceptive wearable and the wearer. Given our success with fully autonomous, unsupervised, offline processing of captured first person experience, we believe that future versions of Duo should be optimized for data capture. By relinquishing the goal of real-time perceptual processing, future versions of Duo could begin to approach the ideals of wearable computing

	images 8hrs/day	angles 8 hrs/day
second	270KB	5.6KB
minute	16MB	340KB
hour	1GB	20MB
day	8GB	160MB
week	56GB	1GB
year	3TB	52GB
decade	20TB	520GB
lifetime (75 years)	1.4PB	4TB

Table 2.1: This table shows the approximate storage requirements for our system if run for 8 hours per day over various time scales that are relevant to human experience.

in terms of comfort, style, robustness, and low-power requirements. Reducing the computational requirements would lead to reduced battery requirements, which would both lead to reduced size and weight for the system. Our experience has shown that researchers in wearable computing have been correct to place such a premium on these design factors, since even the most devoted wearer will find himself deterred from wearing the device for long periods in public, if it is uncomfortable and unstylish. This quickly results in a habit of not wearing the system and a habit of looking for excuses not to wear the system, which is easy to do since the moments of interesting experience for a device such as Duo are often very sparse and to some extent unpredictable in detail. Pushing future versions of Duo to be seamlessly integrated into the life of the wearer would be likely to result in much more extensive data sets that would prove invaluable for offline analysis and learning related to the natural statistics of human behavior. Since a capture only version of Duo would require no display, and possibly no significant interface to the user at all, it should be possible to make a very compact computation system, potentially the size of an IPod since 25GB would be sufficient to capture 24 hours worth of data. The major discomfort for such a system would relate to the head-mounted camera and orientation sensors. Absolute orientation sensors and cameras, such as from cellphones, are both shrinking which bodes well for future designs. Wireless interfaces to the sensors would be especially beneficial, since no mechanical coupling is required between the various sensors except implicitly through their individual,

rigid couplings with the wearer's body. As previously mentioned, our autonomous methods for learning a kinematic model can free a designer or researcher to explore different system structures on a daily basis, without worrying about ruining the data, which would indicate that a versatile sensor mounting system would at least be appropriate in the short term while searching the design space for good system configurations.

Chapter 3

Adapting to the Body

We would like to interpret the images from the camera and the orientations from the orientation sensors in a consistent way across different wearers and across sessions of use by the same wearer. A three dimensional kinematic model that represents the major axes of the limbs and the relative position and orientation of the camera serves as a well-grounded representation for the body of the wearer and its relationship to the sensors. It can also serve as a useful representation for visualizing the output of the sensors. The methods described within this chapter use the streams of orientations and images to autonomously estimate a kinematic model that describes the wearer's body, the camera's relative orientation and position, and the relationship between the kinematic model and images captured from the camera. In addition, the system autonomously estimates the projected direction of gravity in the image for each pixel.

We would like the wearer to don the equipment without worrying about the details of sensor placement, or calibration. Similarly, we would like to give the designer of the equipment flexibility in sensor placement, so that he can put more emphasis on factors such as comfort and ease of use. The more the system is able to autonomously adapt to changes in sensor position, sensor alignment, and the body of the wearer, the more easily the system will be used without error. The methods we describe impose few constraints on the wearer and designer since they only require the use of a camera with known intrinsic parameters, sensors that are in rigid alignment with the body parts, and the visibility of a portion of the hand's workspace from the camera. Moreover, in practice the constraint for rigid alignment with the body parts can be weakened.

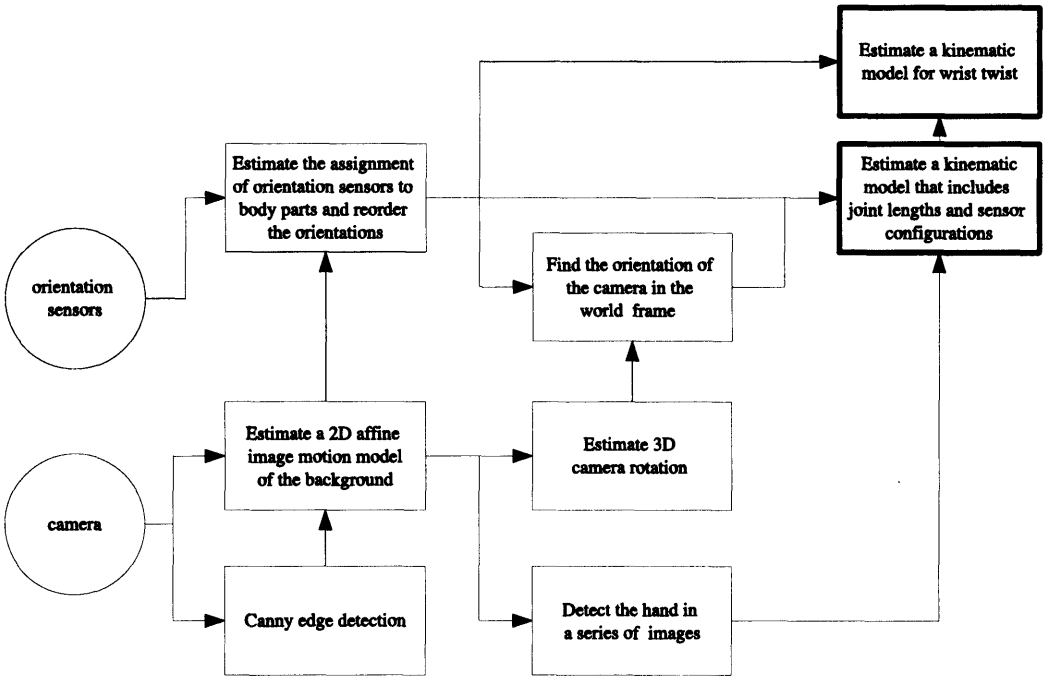


Figure 3-1: This figure gives an overview of the methods we describe within this chapter and their dependencies on one another. A kinematic model of the wearer is our ultimate goal within this chapter, as represented by the darker boxes in the upper-right corner of the diagram.

As shown in diagram 3-1, the automated model estimation system first determines which orientation sensor is attached to which body part by finding a kinematic chain that starts with the camera, passes through all of the orientation sensors, and maximizes the similarity between the motions of adjacent sensors in the chain, including the camera. The 3D rotation of the head-mounted camera, used when finding this assignment of sensors to body parts, is estimated using a fast and robust linear least squares method for finding a 2D affine model of background motion. Second, the system estimates the rotational offset between the head-mounted camera and the head-mounted orientation sensor by relating the frame to frame rotation of the camera to the frame to frame rotation of the head-mounted orientation sensor using linear least squares. Third, the system estimates the location of the hand within the images from the head-mounted camera, which can be done using motion processing or other methods. Finally, a nonlinear optimization is performed to find the kinematic model that minimizes the error between the detected hand locations from image processing, and the estimated hand locations from the kinematic model and the orientations of the camera, torso, upper-arm, and forearm.

3.1 Handmade Models

Much of the research behind this thesis started with handmade models of the wearer's body. These handmade models were based on measurements from the body of the wearer, careful sensor placement, and iterative adjustment of the parameters using a visualization of the 3D kinematic model, as shown in figure 3-2. These models have been useful for visualizing the orientation sensor data, generating coarse estimates of the hand's motion in the world, discretizing the hand's work space, detecting some kinematic activities, and more. These models represent the head, torso, upper-arm, and forearm as boxes with joints between boxes located at the center points of faces and edges. For example, the model shown in figure uses the following dimensions in meters for the body parts.

$$
\begin{aligned}
d_t = d_{torso} &= \begin{bmatrix} 0.39m & 0.18m & 0.47m \end{bmatrix} \\
d_u = d_{upperarm} &= \begin{bmatrix} 0.13m & 0.13m & 0.29m \end{bmatrix} \\
d_f = d_{forearm} &= \begin{bmatrix} 0.08m & 0.08m & 0.33m \end{bmatrix} \\
d_h = d_{head} &= \begin{bmatrix} 0.15m & 0.20m & 0.25m \end{bmatrix}
\end{aligned}
$$

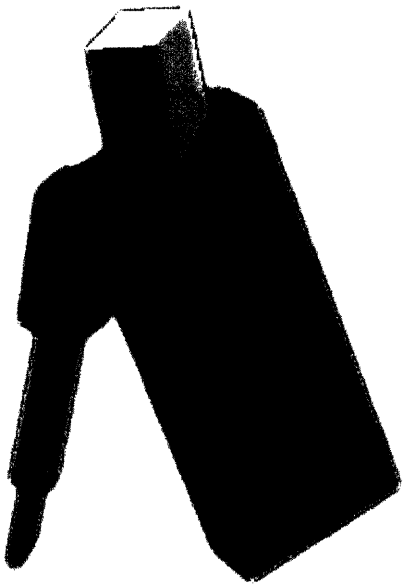


Figure 3-2: This figure shows a visualization with a handmade kinematic model of the wearer reaching into the environment.

The third dimension holds the longest dimension for each body part, which represents the measured major axis (length) of the body part. The first dimension of the torso holds the measured distance from shoulder joint to shoulder joint. The first two dimensions of the head hold loose approximations for the head's dimensions, which are primarily for visualization. The remaining dimensions for the forearm, the upper arm and the torso are for visualization and do not effect the kinematics of the model.

Each orientation sensor returns its orientation as Euler angles α, β, and γ, which the system immediately converts into a rotation matrix with the function $R(\alpha, \beta, \gamma)$.

$$R(\alpha, \beta, \gamma) = R_y(\alpha)R_x(\beta)R_z(\gamma)$$

Where R_y, R_x, and R_z represent the rotation matrices around the y-axis, x-axis and z-axis respectively. We then rotate each of the resulting orientations by a suitable offset, R_{offset}, that accounts for the particular placement of the orientation sensor on the body. So, for example, the final torso rotation matrix, R_{torso}, is computed as follows:

$$R_{torso} = R(\alpha_{torso}, \beta_{torso}, \gamma_{torso})R_{offset_{torso}}$$

Given the dimensions and orientations of the body parts, we can now calculate the positions of the parts in various coordinate systems. For example, the hand's position, x_h, with respect to the approximate pivot point on the torso is calculated in the following way.

$$x_h = R_{torso}\begin{bmatrix} \frac{1}{2}d_{t1} \\ 0 \\ d_{t3} \end{bmatrix} + R_{upperarm}\begin{bmatrix} 0 \\ 0 \\ d_{u3} \end{bmatrix} + R_{forearm}\begin{bmatrix} 0 \\ 0 \\ d_{f3} \end{bmatrix}$$

where d_{tn} represents the n^{th} element of the torso dimensions d_t.

As described within chapter 2, the placement of the head-mounted orientation sensor and the torso-mounted orientation sensor were changed several times during development of the platform. Although joint lengths did not need to be altered, the constant rotational offsets that compensated for these changes needed to be found each time and properly associated with the captured data sets. Besides these fundamental changes to the platform, variations in sensor placement occur between each session, which were typically ignored when using a handmade model. The handmade models also did not precisely or dependably characterize the relationship between the kinematic model and the images from the camera. The automated model estimation described within this chapter addresses these issues.

One disadvantage of the machine made models is that, unlike the handmade models, they do not include a measurement of the length of the major axis of the torso. This dimension does not play a role the kinematic chain from the camera to the wearer's hand, so it is not estimated by the automated methods described within this chapter. A method based on detecting the distance of the camera from the floor as a function of bending in the torso might be able to produce such an estimate, but we have not attempted to do this. When modeling motion of the hand with respect to the world, an estimate of the major axis of the torso with respect to a suitable pivot point can improve the estimate of the hand's motion, especially since the legs are often planted during manipulation. A properly constructed handmade model can also be better for visualization since it incorporates three dimensions for each body part, is hand-tuned while being visualized, and includes the major axis of the torso. So, for some tasks the handmade models may be superior to the purely machine made model. In these situations, augmenting the automated estimates with some handmade estimates can produce a better model.

3.2 Assigning Orientation Sensors to Body Parts

We have four orientation sensors and four body parts and we wish to automatically determine which sensor is mounted to which body part. This section describes a method that determines which orientation sensor is attached to which body part by finding a kinematic chain that starts with the camera, passes through all of the orientation sensors, and maximizes the similarity between the motions of adjacent sensors in the chain, including the camera.

On several occasions due to a software change or hardware change, such as rewiring or remounting, the ordering of the angular measurements from the orientation sensors has changed - sometimes unexpectedly. The following algorithm quickly determines the assignment of sensors to body parts for Duo, which has been helpful for both real-time processing, where it simplifies setup, and offline processing, where it simplifies the use of multiple data sets that may have different sensor configurations jumbled together. At minimum, the robustness afforded by the following algorithm gives a practical benefit for a research platform, since many hardware and software components tend to be in flux.

We assume that the four body parts form a single kinematic chain of body parts, and

that the measured motions of adjacent body parts will tend to be more similar than the measured motions of non-adjacent parts. With these assumptions, we define the problem as one of finding a kinematic chain whose path through the body parts minimizes the differences between the motions of adjacent parts, which we can reasonably formalize as solving

$$argmin(T(C(m_1, m_2), C(m_2, m_3), ..., C(m_{n-1}, m_n)))_m \tag{3.1}$$

where T is a function that combines the individual costs into a total cost, C is a symmetric function that returns the cost between the measured motions from two adjacent parts, and $m$is a sequence of measured motions from the n distinct parts $m = [m_1, m_2, ..., m_n]$. Notice that all n parts must be included in $m$once and only once. The same formalization holds if multiple sensors are attached to the same body part, although the ordering of these sensors in the resulting kinematic chain m will be arbitrary, and further processing or prior information would be required to determine the exact relationship between the chain of sensor measurements and the chain of distinct body parts. For motion capture data that included position information and no camera, some researchers have used a similar formulation, which they have solved with a minimum spanning tree and a cost function related to how strongly two parts appear to share a position describing a rotational joint [47]. Since we only have uncalibrated orientation information we must devise a different cost function. Likewise, our set of admissible solutions does not include trees, which alters the problem. [1]

For Duo, we optimize the following function which has the form of equation 3.1

$$argmin \left(\sum_i \frac{\sum_t |\theta_{i,t} - \theta_{i+1,t}|}{\sum_t 1} \right)_\Theta \tag{3.2}$$

where $\Theta = [\theta_{1,t}, \theta_{2,t}, \theta_{3,t}, \theta_{4,t}, \theta_{5,t}]$ and $\theta_{n,t}$ is the sequence of angular changes over time, t, measured for one of the five sensors and assigned to the n^{th} body part in the kinematic chain. The symmetric cost function, C, for this equation is

$$C(\theta_i, \theta_{i+1}) = \frac{\sum_t |\theta_{i,t} - \theta_{i+1,t}|}{\sum_t 1}$$

[1]The measured motion from sensors on the same body part might be similar enough to allow for their categorization as a single body part, or as with our situation, we might know ahead of time that two sensors are mounted to the same body part. Multiple kinematic chains would further complicate things, and require inference of trees from the adjacency information.

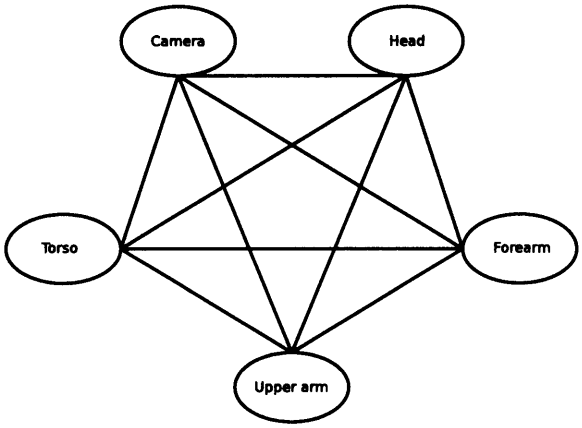


Figure 3-3: This figure shows the graph for the problem, which is equivalent in complexity to the Traveling Salesman Problem, and is used to assign body parts to sensors by finding the kinematic chain from the camera to the forearm. Each node represents the time series of orientation change estimates associated with a sensor. The head, torso, forearm, and upper arm estimates come from the body-mounted orientation sensors, while the camera orientation estimates are found by analyzing image motion. Each edge has a cost associated with it from the cost function, C, which represents the dissimilarity between the two time series the edge connects. The minimal cost path that starts from the camera and visits each node once and only defines the estimated kinematic chain through the sensors and their associated body parts. With proper data the minimal path should go from the camera to the head to the torso to the upper arm and terminate at the forearm.

which computes the average difference between the angular change associated with the two sensors. The angular change, θ_{t_2}, is the angle from the angle and axis form of the rotation $R_{t_2 t_1}$, which is the rotation of the sensor between sequential times t_1 and t_2. So,

$$R_{t_2 t_1} = R_{t_1}^T R_{t_2}$$

with R_{t_1} and R_{t_2} being the consecutive orientations of the sensor at times t_1 and t_2. The quantity θ is invariant to constant rotational offsets, so two sensors on the same body part should have very similar values of θ over time. Because the angular change between time steps is small we can get away with the absolute value of the difference in two angles without worrying about wrap-around effects. For equation 3.2, the combination function T is simply a summation.

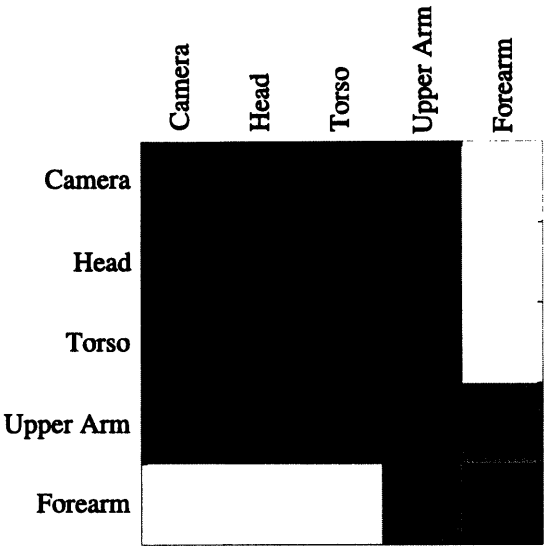


Figure 3-4: This figure shows the average final cost matrix for the tests shown in figure 3-5. The row indices from top to bottom represent the camera, head, torso, upper arm, and forearm respectively. As with all dissimilarity matrices, the diagonal is set to zero, since each body part's motion is identical to itself. The forearm's motion is clearly most similar to the upper arm's motion and very dissimilar to the other parts.

We include the camera in the chain by using its time series of estimated angular changes as found through the affine motion approximation described in section 3.5.

With this starting constraint, optimizing the objective function of equation 3.2 defined over the set of admissible sequence orderings Θ, is equivalent to solving the traveling salesman problem (TSP) [11] without the roundtrip constraint, which has equivalent complexity to standard TSP. The kinematic chain must visit each sensor once and only once, and must minimize the sum of the costs between the sensors. In the associated fully connected, undirected graph, each node represents a sensor and its motion data, θ_t, each edge represents the option of assigning two sensors to be on adjacent body parts, and the cost associated with traversing an edge is defined by the symmetric cost function C. Figure 3-3 shows the graph associated with the specific problem the system solves. The symmetric cost function C also defines a symmetric cost matrix for the fully connected graph, which we can use to visualize the cost function applied to the data as shown in figure 3-4.

The traveling salesman problem is NP-hard, but we have very few sensors, so the brute

force enumeration of the $n!$ possible paths is fast. We have sometimes restricted the second sensor in the chain to be the sensor with motion most similar to the camera, but this greedy step is unnecessary and can sometimes increase convergence time due to noisy camera estimates and the similarity between torso motion and the head. Without this greedy step we have $4! = 24$ paths over which to sum the costs and pick the minimum for our brute force solution. We initially tried an entirely greedy approach to finding the best path, but found that it converged significantly more slowly, partly because it did not take into account the forearm's tendency to have motion much more closely related to the upper arm's motion than any of the other body parts, which we would expect given that it sits at the other end of the kinematic chain. This property suggests that a more general kinematic structure discovery method for a more complicated kinematic model could detect the end points of the included kinematic chains in a similar way.

Once we have our ordering for the sensors in the kinematic chain, assigning them to body parts is trivial. The first two sensors in the optimal chain are the camera and head orientation sensor. The next three sensors are mounted to the torso, the upper arm, and the forearm, respectively.

The graph in figure 3-5 shows results for 20 tests over 400 frames each (approximately 30 seconds) from a data set 1, which consists of everyday activity within a kitchen, such as opening cabinets, picking up a cup, etc.. By 200 frames over 50% of the tests have found the optimum kinematic chain. By 400 frames 90% of the tests have found the optimum kinematic chain. Convergence to the solution relies on non-trivial arm motion. If the wearer is only walking with his arm by his side, the system will not necessarily result in the proper ordering, since during this activity the dominant motions of all the body parts are related. Once a non-trivial manipulation activity occurs, the solution is found quickly. We can gate the estimation based on coarse properties relating to the significance of the motions to compensate for this issue. For offline processing, we can increase the confidence in our estimate by using large portions of the captured data. For online processing, we can wait until a non-trivial action takes place or request that the wearer perform a non-trivial action, such as waving his hand around and looking around the room. Waiting for a non-trivial action is typically sufficient, since the system is likely to be uninterested in trivial activities anyway.

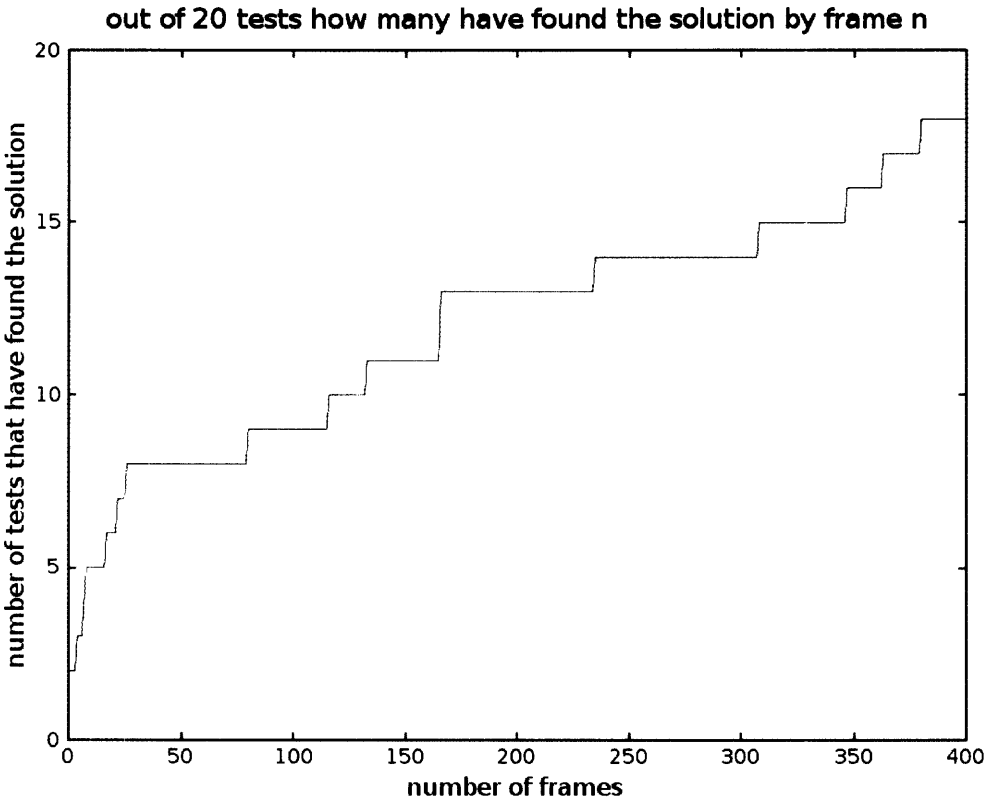


Figure 3-5: This graph shows results from 20 tests over 400 frames each, which is approximately 30 seconds. The test dataset consists of everyday activity within a kitchen, such as opening cabinets, picking up a cup, etc.. By 200 frames over 50% of the tests have found the optimum kinematic chain. By 400 frames 90% of the tests have found the optimum kinematic chain.

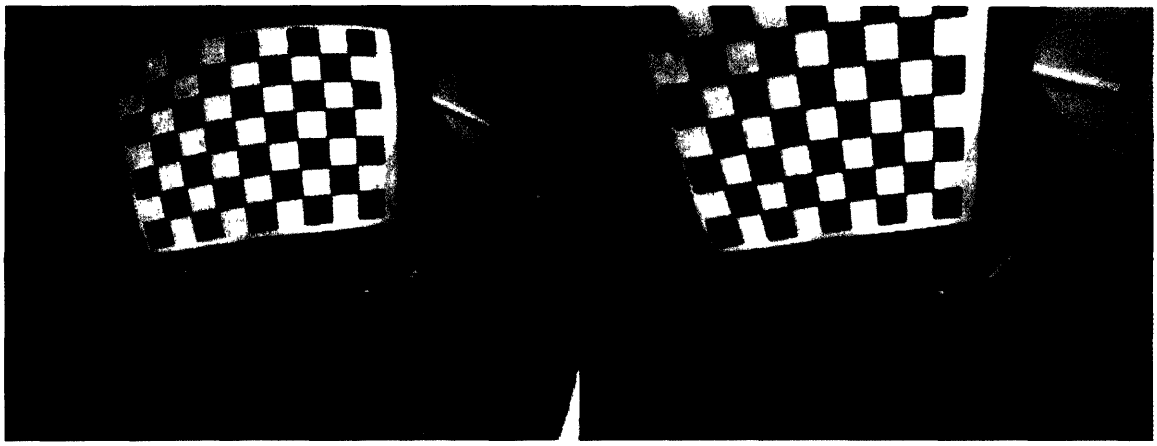


Figure 3-6: The left image was used during calibration with a standard calibration pattern. The right image shows the image after using the undistort function in OpenCV with the parameters determined by the Camera Calibration Toolbox for Matlab. The undistort function removes radial distortion and centers the image around the optical center. The elimination of a significant amount of the image around the periphery and the results from resolution changes lead to undesirable side effects of the process.

3.3 Camera Calibration

Some of the methods described within this chapter assume that we have a calibrated camera. The same camera captures all of the video, so we only need to calibrate the camera once in order to assist the analysis of many days worth of video. Since we have direct access to this fixed-lens, fixed-focus camera. we can calibrate it by hand using a calibration pattern and standard algorithms. For this very common calibration problem we used the open source Camera Calibration Toolbox for Matlab distributed by Jean-Yves Bouguet.

Several of the methods within this chapter make use of the optical center and the focal length of the camera. Besides determining the camera's intrinsic parameters, the calibration procedure finds parameters for radial distortion. Removing this radial distortion can benefit several of our machine vision algorithms including motion processing and shape processing by helping to preserve translation invariance across the image. We use a very wide angle camera with a horizontal field of view of approximately 90 degrees. The radial distortion and vignetting[2] for this camera are extreme, as can be seen on the left side of figure 3-6.

[2]Vignetting refers to the darkening around the periphery of the image that is especially noticeable in the corners.

Given the radial distortion parameters estimated during calibration, we use the undistort function in OpenCV, which centers the image, removes the radial distortion, and cuts out the periphery based on the estimated optical center and radial distortion parameters, [12]. The major drawbacks of undistorting the image prior to processing are the extra computation involved, the effects of space-variant interpolation across the image, and the reduction in visible area, all of which can be seen in figure 3-6.

3.4 A 2D Affine Model for Background Motion

The system uses a 2D affine motion model to quickly and robustly estimate the image motion resulting from the background environment. Subsequently, this motion model is used for a variety of tasks, such as attending to regions of interest and estimating the 3D rotation of the camera. A 2D affine motion model is defined by a $2x3$ affine matrix M that transforms an image position $p_1 = (x_1, y_1)$ at time step 1 into an image position $p_2 = (x_2, y_2)$ at time step 2,

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

This model can account for global changes in translation, scale, rotation, and shearing, but is unable to properly handle foreshortening and other perspective effects. Nevertheless, its fidelity is sufficient for modeling global background motion between two images that are close together in time. Its simplicity allows us to use weighted linear least squares to fit the model M to a set of estimated translations each of which has an associated covariance matrix that represents the estimate's error. Furthermore, after we have our model, M, we can compute the Mahalanobis distance between these translation estimates and the model to obtain a good measure of how likely it is that the translation was generated by the model, or in other words, the likelihood that a point in the image is part of the global background with motion modeled by M.

2D affine motion models are commonly used to model global image motion [26, 66, 63, 49, 58]. Many choices exist for fitting the 2D affine motion model to the image data, in terms of the cost function and optimization method applied. The method we have developed formulates the problem in terms of block matching with Gaussian measurement errors, and

a closed form, weighted linear least squares solution. While linear least squares fitting of 2D affine motion models to the best matches from block matching is a common approach in the current literature, the use of Gaussian measurement errors for computing optic flow from block matching appears to be uncommon. Ajit Singh was one of the first researchers to model measurement error from block matching using Gaussians [56]. His method, however, did not constrain the solution to be in the form of a parametric motion model as we do. He used iterative optimization as opposed to a direct solution, which implies less efficient processing. He performed block matching on the Laplacian of the intensity image, rather than the image itself, as we do. He modeled the measurement error distribution by applying an exponential to the raw errors, while we simply threshold the errors to form a binary error map. He modeled the error around the mean of the error distribution, while like Anandan, [1] , we represent the measurement error around the best matching translation. Finally, he used a dense map of block matching results, while we only use the results of block matching started at Canny edge points, [12]. Out of these differences, the most significant by far, is our efficient, yet robust, closed form weighted linear least squares solution to fitting the 2D affine motion model. This formulation can run in real-time, provides affine motion estimates that are robust enough to find the alignment of the camera with the head mounted orientation sensor, and has the added benefit of providing an edge segmentation map in terms of the Mahalanobis distance.

3.4.1 Overview

The major computational load for motion processing comes from high resolution estimation of the image motion model M from images I_1 and I_2. We take several steps in order to reduce the computation and achieve real-time performance. The frame rate of processing is especially important for motion since a larger time difference between frames leads to complicating factors such as larger between frame image displacements, increased surface occlusions, and smaller overlapping areas for the two fields of view. These factors reduce the set of valid correspondences between the frames and increases the search required to find legitimate correspondences.

First, we find a coarse low-resolution affine motion estimate in order to reduce the size of the search window required when matching a block from I_1 to blocks in I_2 at high resolution. The low resolution estimate can be misleading, but typically the majority of the

image motion is well modeled as 2D affine motion and low-resolution images give a useful hint as to the best area over which to search for block displacement.

Next we reduce the computation required by restricting the high resolution block matching to points on edges. Canny edge detection, [12], is efficient and several other algorithms used by our system make use of the same edge map, so the computational cost of finding edges is negligible and the measurements of edge motion can be useful elsewhere. We use our own code for Canny edge detection, which we have modified to simultaneously group edges that are strongly related, where two edge points are considered strongly related if a smooth, low-curvature path through other strongly related edge points connects them. Edges also form a good compromise between dense estimates performed on every pixel and sparse point tracking approaches that usually track corners or multi-scale blobs. Block matching should be successful along at least one degree of freedom because edges fall on blocks with high contrast elements. Ambiguity along the edge is more likely, but our error model can represent this uncertainty.

Fitting a 2D affine motion model with weighted linear least squares to the results of restricted block matching makes a useful compromise between fidelity, computation, and robustness.

Algorithm Overview

The estimation method performs the following steps, or a close variation of them :

1. Perform a low-resolution estimation of the 2D affine motion, M_L. The following steps describe one method we have used:

   (a) Average and down-sample the sequential gray scale images I_1 and I_2 to obtain scaled versions of the images I_{1s} and I_{2s}, which are $\frac{1}{4}$ of the size of the original images.

   (b) For each pixel, perform block matching on I_{1s} and I_{2s} to determine the best matching translation of the pixel's image block between the two images.

   (c) Find the linear least squares estimate of M_L given the estimated translations.

2. Perform high-resolution estimation of the 2D affine motion M:

   (a) Find edges in the images I_1 and I_2 using Canny edge detection.

(b) For each edge pixel, perform block matching between I_1 and I_2 with the search window centered on the properly scaled motion estimate provided by M_L. Find the best matching translation for each image block and a full 2D covariance matrix representing a Gaussian model of the match error around this best match.

(c) Find the linear least squares estimate of M using the translations and a matrix U that incorporates the covariance matrices.

(d) Throw away outliers by removing the 20% of the edges with the highest error. The estimated affine motion of these edges has the highest Mahalanobis distance from the measured motion as described by the best match translations and U.

(e) Reestimate M using the remaining edges (80% of the total).

(f) Throw out the 25% of all the edges with the highest error.

(g) Reestimate M using the remaining edges (75% of the total).

This particular description of low-resolution estimation serves as an example of one method that we have used to provide hints to the high-resolution estimation through M_L. Depending on the expected amount of image motion and the search window used during block matching, low-resolution motion estimation may not be useful, in which case we would set M_L to the identity transform, $M_L = [I|0]$. Likewise, we can produce M_L using the same steps as the high-resolution method, but applied to properly scaled low-resolution images. Once we have estimated the relationship between the head-mounted orientation sensor and the head-mounted camera, we could also use the head-mounted orientation sensor to generate M_L.

3.4.2 Block Matching

As illustrated within figure 3-7, the block matching algorithm we use outputs a Gaussian model of how well a block matches an image over the search area. More precisely, we estimate the motion of point p_1 in image I_1 to the corresponding point p_2 in image I_2 with the standard technique of block matching. This technique searches for point correspondences between I_1 and I_2 using image blocks as the point descriptor and a suitable comparison measure such as normalized correlation or the sum of absolute differences. For each edge point p, our block matching algorithm outputs the location of the best match, p_b, and a

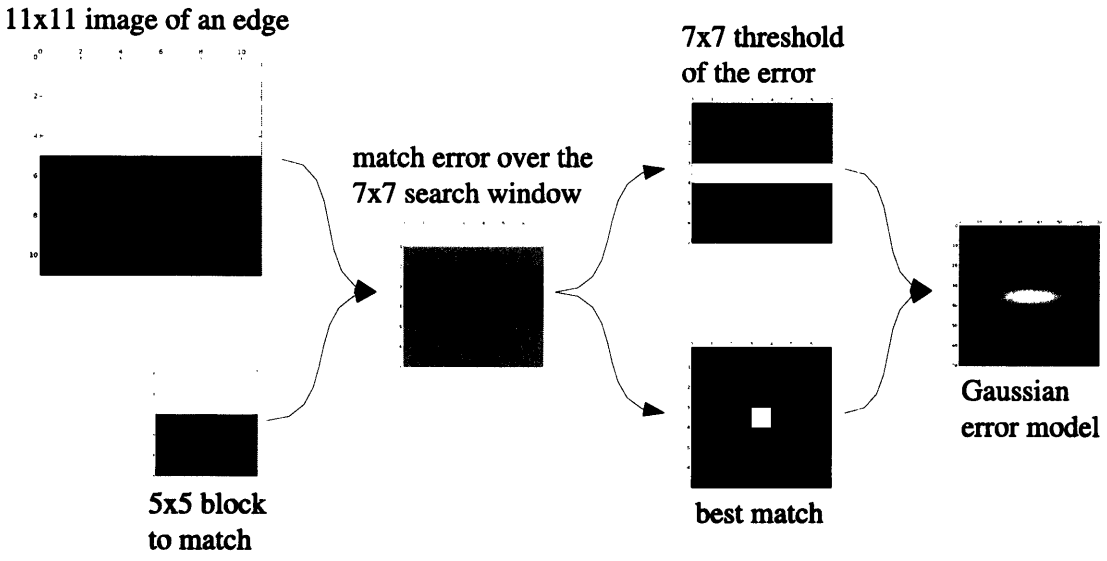


Figure 3-7: This figure illustrates the block matching algorithm by showing the results of matching a 5x5 block to an 11x11 image using a 7x7 search window.

covariance matrix, C, that models the matching error around this best match. The specific procedure used in our code for the affine motion estimate follows:

1. For each edge point p_1 in I_1

   (a) Select a square, B, from image I_1 centered around p_1.

   (b) Define a search window S within image I_2 centered around our current best estimate of where the corresponding point p_2 will be located. (For our high resolution processing this involves using the suitably scaled low resolution affine motion estimate to set the center of the search window, or making use of the 3D rotation of the camera as estimated from the head-mounted orientation sensor.)

   (c) Slide the block B to all points p_s contained in S

      i. Select a square B_s centered on p_s and of the same size as B.

      ii. Calculate a match value, h_s, by comparing B_s to B using the sum of absolute pixel differences $h_s = \sum_{x,y} |B_s(x,y) - B(x,y)|$

   (d) Set the best position estimate p_b and associated match value h_b to be the point p_s with the lowest h_s

(e) Set the best estimate for the translation t of p_1 `to be` $t = p_b - p_1$

(f) Compute a covariance matrix C that describes the matching uncertainty centered around p_b. With

$$C = \alpha I + \frac{1}{\sum w_s} \sum w_s (p_s - p_b)(p_s - p_b)^T \text{ where } w_s = \begin{matrix} 1 \text{ if } h_s < h_b + \tau \\ 0 \text{ if } h_s \geq h_b + \tau \end{matrix}$$

and

$$w_s = \begin{matrix} 1 \text{ if } h_s < h_b + \tau \\ 0 \text{ if } h_s \geq h_b + \tau \end{matrix}$$

where τ is our threshold for points that are sufficiently well matched to justify inclusion in C. For the blocks of size $5x5$ that we use during high resolution matching we set $\tau = 200$ which assumes that we should see no more than 4 units of additional error per pixel in a block that truly matches, since $5x5x4 = 200$. We set τ to this constant value by hand. It works sufficiently well for our purposes, but estimating τ from measured pixel errors, especially as a function of brightness and contrast, might improve the algorithm's performance. To avoid over-confidence in the estimated error distribution, we also add αI to C, which is equivalent to convolving the error Gaussian with a circular Gaussian with variance α. We have had success with $\alpha = \frac{1}{4}$.

For low resolution motion estimation, we have used a search window S with size $7x7$ and a block B of size $3x3$. For high resolution motion estimation, we have used a search window S with size $11x11$ and a block B of size $5x5$.

3.4.3 Low Resolution 2D Image Motion Estimation

This subsection describes a simple low-resolution method we have used to find a low-resolution affine motion model M_L that we can pass to a higher-resolution motion processing method. Given two images I_{1s} and I_{2s} we perform block matching on all points, not just edges, to find values for t_i, and we do not calculate the covariance matrices C. This leads to the following equation in the standard linear least squares form of $Ax = b$, [59]:

$$X_1 M_L^T = X_2$$

where

$$M_L = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \end{bmatrix}$$

$$X_1 = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} x_1 + t_{x1} & y_1 + t_{y1} \\ x_2 + t_{x2} & y_2 + t_{y2} \\ \vdots & \vdots \\ x_n + t_{xn} & y_n + t_{yn} \end{bmatrix}$$

with t_i being the translations associated with the best block matches for each of the n pixels indexed by i.

Since we are using all pixels to estimate M_L, X_1 is constant for every image pair of the same size and we only need to compute $K = (A^T A)^{-1} = (X_1^T X_1)^{-1}$ once upon starting the motion processing system, assuming the stream of images stays at a constant size. This results in the constant $3x3$ matrix K. Consequently, for each image pair the system receives, we only need to compute

$$M_L^T = K X_1^T X_2$$

Where $X_1^T X_2$ involves a low cost multiplication of a $3xn$ matrix with an $nx2$ matrix followed by a trivial multiplication of a $3x3$ matrix with the resulting $3x2$ matrix.

3.4.4 High Resolution 2D Image Motion Estimation

For high resolution motion estimates, we only perform block matching at edge points and we use weighted linear least squares to incorporate the error covariance matrices generated by the block matching process into the estimation of the motion model M.

Weighted Linear Least Squares

We know that the least squares solution for $Ax = b$ is $x = (A^T A)^{-1} A^T b$, which minimizes $\|Ax - b\|^2 = (Ax - b)^T (Ax - b)$ [59]. In order to better model the error in our translation estimates, we can use the standard practice of applying a weight matrix W and minimizing $\|(WAx - Wb)\|^2 = (Ax - b)^T W^T W (Ax - b)$ instead. Clearly, by substitution, this is still linear least squares and can be minimized with $x = (A^T W^T W A)^{-1} A^T W^T W b$. By defining $U = W^T W$ to be an inverse covariance matrix for a multi-dimensional Gaussian on the block translation estimation error, we can discount the influence of some types of block matching error, such as errors along the direction of the edge where there will typically be more block matching uncertainty. In this case, solving

$$x = (A^T U A)^{-1} A^T U b \tag{3.3}$$

minimizes

$$(Ax - b)^T U (Ax - b) \tag{3.4}$$

To see this better, we can consider the linear least squares solution to be maximum likelihood estimation of our model x where the error is Gaussian distributed with inverse covariance matrix U. Specifically, with $\mathcal{N}$ representing the equation for a d-dimensional Gaussian density on random vector y with covariance matrix Σ and mean vector μ we have

$$\mathcal{N}(\mu, \Sigma, y) = (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(y-\mu)^T \Sigma^{-1} (y-\mu)}$$

and we can thus model our block matching error with

$$\mathcal{N}(b, U^{-1}, Ax) = (2\pi)^{-\frac{d}{2}} \left|U^{-1}\right|^{-\frac{1}{2}} e^{-\frac{1}{2}(Ax-b)^T U (Ax-b)}$$

where we set the mean vector to the means of the translation estimates $\mu = b$, the covariance matrix $\Sigma = U^{-1}$, and we constrain the random vector y to obey our model Ax where x is the vector of free parameters defining M. The maximum likelihood solution is identical if we set $\mu = Ax$ and $y = b$, so the distinction is not important for our purposes. We can either think in terms of finding the affine motion model that generates the most likely set of samples given our error model, or we can consider ourselves to be finding the affine model that gives the mean vector that maximizes the probability of the samples.

Minimizing $(Ax-b)^T U(Ax-b)$ clearly maximizes the probability and hence gives us the maximum likelihood solution, as shown below in the standard way

$$\begin{aligned} argmax(\mathcal{N}(b, U^{-1}, Ax))_x &= argmax\left((2\pi)^{-\frac{d}{2}} \left|U^{-1}\right|^{-\frac{1}{2}} e^{-\frac{1}{2}(Ax-b)^T U(Ax-b)}\right)_x \\ &= argmax(\log(\mathcal{N}(b, U^{-1}, Ax)))_x \\ &= argmax(-\frac{1}{2}(Ax-b)^T U(Ax-b))_x \\ &= argmin((Ax-b)^T U(Ax-b))_x \end{aligned}$$

Defining the Matrices

To put our 2D affine model into a form that allows us to use weight matrix U we vectorize M to m, so that if we ignore U we have $X_1 m = X_2$, with

$$X_1 = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix}$$

$$m = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} x_1 + t_{x1} \\ y_1 + t_{y1} \\ x_2 + t_{x2} \\ y_2 + t_{y2} \\ \vdots \\ x_n + t_{xn} \\ y_n + t_{yn} \end{bmatrix}$$

where using the equalities $A = X_1$, $x = m$, and $b = X_2$ translates these matrices into the proper terms for the weighted least squares objective function of equation 3.4 . We now define U to be a sparse block diagonal matrix with $2x2$ matrices C_i^{-1} along the diagonal, where C_i^{-1} is the inverse covariance matrix for the 2D Gaussian that models the matching error around the best matching location for the block associated with edge point i.

$$U = \begin{bmatrix} C_1^{-1} & 0 & 0 & \cdots & 0 \\ 0 & C_2^{-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & C_n^{-1} \end{bmatrix}$$

Now, by equation 3.3, solving

$$m = (X_1^T U X_1)^{-1} X_1^T U b \tag{3.5}$$

will minimize the objective function of 3.4 and give us the solution we desire.

Simplifying The Solution

If we were to actually construct the full matrix U in order to solve equation 3.5, the computation would be daunting. Fortunately, due to the sparse block form of the matrices the computation is efficient enough for real-time processing. The details of the simplification are not worth showing here, but the result is worth seeing as evidence of the low computational requirements.

For the first part of equation 3.5, the computation of $X_1^T U X_1$ only requires that we sum n $6x6$ symmetric matrices, where n is the number of edge points i used in the estimation. The 21 distinct terms have the simple and redundant form shown below, where each .

represents the corresponding lower diagonal value and $C_i^{-1} = \begin{bmatrix} u_{1i} & u_{2i} \\ u_{3i} & u_{4i} \end{bmatrix}$ with $u_{2i} = u_{3i}$ due to symmetry.

$$X_1^T U X_1 = \sum_i \begin{bmatrix} x_i^2 u_{1i} & . & . & . & . & . \\ y_i x_i u_{1i} & y_i^2 u_{1i} & . & . & . & . \\ x_i u_{1i} & y_i u_{1i} & u_{1i} & . & . & . \\ x_i^2 u_{3i} & x_i y_i u_{3i} & x_i u_{3i} & x_i^2 u_{4i} & . & . \\ y_i x_i u_{3i} & y_i^2 u_{3i} & y_i u_{3i} & y_i x_i u_{4i} & y_i^2 u_{4i} & . \\ x_i u_{3i} & y_i u_{3i} & u_{3i} & x_i u_{4i} & y_i u_{4i} & u_{4i} \end{bmatrix}$$

The structure of the matrix is more clear when written in block form with $v_i^T = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}$

$$X_1^T U X_1 = \sum_i \begin{bmatrix} u_{1i} v_i v_i^T & u_{3i} v_i v_i^T \\ u_{3i} v_i v_i^T & u_{4i} v_i v_i^T \end{bmatrix}$$

The resulting symmetric $6x6$ matrix, $X_1^T U X_1$, will be positive definite except for extreme circumstances, such as when not enough edges are provided due to darkness. Consequently, we can compute $(X_1^T U X_1)^{-1}$ using fast specialized $6x6$ matrix inversion code. We adapted code from Fermilab by David Sachs available on the web, which he documents as using an algorithm from pages 138-139 of [22]. This code notifies the caller if an error is encountered when computing this inverse, so the system can recover on the rare occasions when the $6x6$ matrix is not positive definite or otherwise improper.

The second part of equation 3.5, $X_1^T U b$, is also computationally simple with

$$X_1^T U b = \sum_i \begin{bmatrix} (x_i + t_{xi}) x_i u_{1i} + (y_i + t_{yi}) x_i u_{3i} \\ (x_i + t_{xi}) y_i u_{1i} + (y_i + t_{yi}) y_i u_{3i} \\ (x_i + t_{xi}) u_{1i} + (y_i + t_{yi}) u_{3i} \\ (x_i + t_{xi}) x_i u_{3i} + (y_i + t_{yi}) x_i u_{4i} \\ (x_i + t_{xi}) y_i u_{3i} + (y_i + t_{yi}) y_i u_{4i} \\ (x_i + t_{xi}) u_{3i} + (y_i + t_{yi}) u_{4i} \end{bmatrix}$$

which can also be more clearly written in block form with v_i

$$X_1^T U b = \sum_i \begin{bmatrix} v_i u_{1i} & v_i u_{3i} \\ v_i u_{3i} & v_i u_{4i} \end{bmatrix} \begin{bmatrix} x_i + t_{xi} \\ y_i + t_{yi} \end{bmatrix}$$

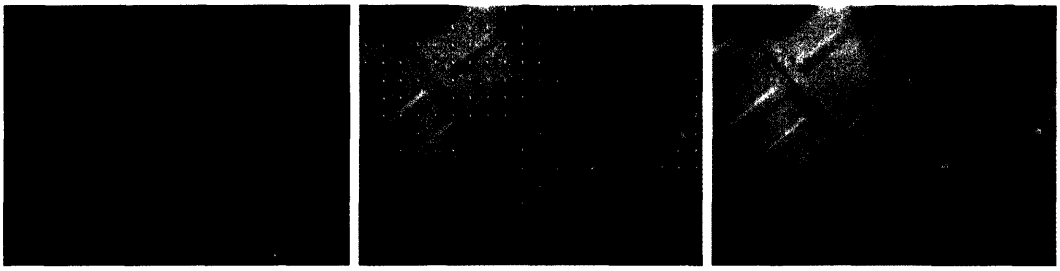


Figure 3-8: This figure shows an captured image of the motion backgrounding algorithm. The first frame shows the Mahalanobis distance for the motion edges, which in this case leads to strong responses from the hand, arm, and brim of the hat. The second image shows the first image in time with a vector field sampled from the 2D affine background motion model with respect to which the Mahalanobis distances were calculated. The image on the far right shows the second image in time to which the motion model maps edge locations in the first image.

Now, we only need to multiply these two parts to find our solution for m and the corresponding motion model matrix M.

Refitting And Motion Backgrounding

As described in the algorithm overview from subsection 3.4.1, we iterate the fitting process in order to remove the influence of edge points that are not likely to be part of the motion background. On each iteration we remove the worst fitting $n\%$ of the edge points and reestimate m, which is computationally reasonable since we only need to perform block matching once. We determine how well each edge point matches the model by calculating the Mahalanobis distance, d_i, between the best match translation vector, t_i, for edge point i and the translation predicted by the model M, Mv_i, relative to the edge point's error model defined by the covariance matrix C_i.

$$d_i = \left(\left(Mv_i - \begin{bmatrix} x_i + t_{xi} \\ y_i + t_{yi} \end{bmatrix} \right)^T C_i^{-1} \left(Mv_i - \begin{bmatrix} x_i + t_{xi} \\ y_i + t_{yi} \end{bmatrix} \right) \right)^{\frac{1}{2}}$$

The units of Mahalanobis distance, d_i, are image pixels, so working with these distances is intuitive. The Mahalanobis distances also rank the edge points, which allows us to throw out the points that fit the model poorly. For many environments the majority of

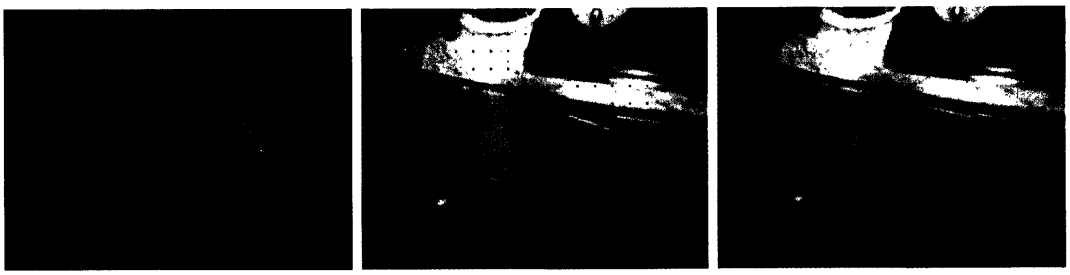


Figure 3-9: This figure shows an captured image of the motion backgrounding algorithm. The first frame shows the Mahalanobis distance for the motion edges, which in this case leads to strong responses from the arm, and cabinet door. The second image shows the first image in time with a vector field sampled from the 2D affine background motion model with respect to which the Mahalanobis distances were calculated. The image on the far right shows the second image in time to which the motion model maps edge locations in the first image.

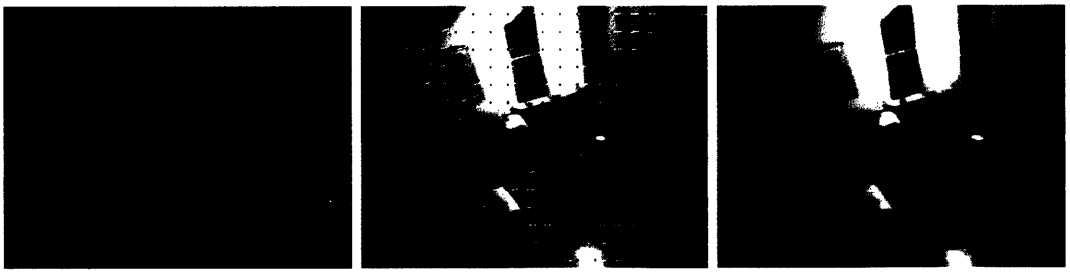


Figure 3-10: This figure shows an captured image of the motion backgrounding algorithm. The first frame shows the Mahalanobis distance for the motion edges, which in this case leads to low magnitude distances spread across the image, since the camera is rotating. The second image shows the first image in time with a vector field sampled from the 2D affine background motion model with respect to which the Mahalanobis distances were calculated. The image on the far right shows the second image in time to which the motion model maps edge locations in the first image.

the image motion relates to the background environment, and can be well-modeled with M. In environments where the majority of the image motion relates to the background, edge points weighted by d_i serve as a useful map for motion based attention, and motion backgrounding. Other segment [65]

3.5 Estimating 3D Camera Rotation from 2D Affine Image Motion

The methods of Section 3.4 give us a 2D affine motion model, M, for the background environment between sequential images I_1 and I_2. Within this section, we use a weak perspective camera model along with this estimated 2D affine transformation to coarsely estimate the 3D rotational motion of the camera between the images. We define the camera's local coordinate system to have its z axis pointing into the image, its x axis points along the width of the image to the right of the optical center, and its y axis pointing from the optical center to the bottom of the image.

Writing the 2D Affine Transform as a 3D Affine Transform

To emphasize that the motion model M is a 2D affine transform, we will define $A'' = M$ and use A'' instead of M for the rest of this section. We first find a 3D affine transform A that has the equivalent effect of A'' on image locations and incorporates camera parameters. A'' defines a 2D affine transform on raw pixel coordinates $p'' = (u'', v'', 1)$ that gives $p''_2 = Ap''_1$, where p''_1 and p''_2 are the pixel coordinates in I_1 and I_2 respectively. We convert A'' to an affine transform A on pixel coordinates $p = (u, v, f)$ with the constant focal length f as the 3^{rd} component, and with $p = (0, 0, f)$ positioned at the optical center of the image such that $p_2 = Ap_1$. We initially create A' that operates on $p' = (u', v', 1)$ with $p' = (0, 0, 1)$ positioned at the optical center of the image (c_x, c_y) so that $(u', v') = (u'' - c_x, u'' - c_y)$. With $c = (c_x, c_y, 0)$, $p''_1 = p'_1 + c$ and $(p'_2 + c) = A''(p'_1 + c)$ which implies that $p'_2 = A''p'_1 + A''c - c = A'p'_1$ so that with

$$A'' = \begin{bmatrix} a''_1 & a''_2 & a''_3 \\ a''_4 & a''_5 & a''_6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A' = \begin{bmatrix} a_1'' & a_2'' & a_3'' + (c_x a_1'' + c_y a_2'') - c_x \\ a_4'' & a_5'' & a_6'' + (c_x a_4'' + c_y a_5'') - c_y \\ 0 & 0 & 1 \end{bmatrix}$$

We then create A by scaling the 3^{rd} column of A' by $1/f$.

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_1' & a_2' & a_3'/f \\ a_4' & a_5' & a_6'/f \\ 0 & 0 & 1 \end{bmatrix}$$

Specifying the Camera Motion Model

We now define our camera model, and an equation that models the image motion resulting from the motion of this camera model within a static environment.

A 3D point in the world $X_w = (x_w, y_w, z_w, 1)$ is transformed into a point, X_c, in the camera's coordinate system by rotation R_c^T and translation $-t_c$, $X_c = [R_c^T | -t_{c2}]X_w$. We wish to estimate the between image rotation, $R_{c_2c_1}$, of the point X_w with respect to the camera's frame of reference. For I_1, $X_{c1} = [R_{c1}^T | -t_{c1}]X_w$, and for I_2, $X_{c2} = [R_{c2}^T | -t_{c2}]X_w$. $X_{c2} = [R_{c2c1}|t_{c2c1}]X_{c1}$, so by substitution $[R_{c2}^T | -t_{c2}] = [R_{c2c1}^T|t_{c2c1}][R_{c1}^T | -t_{c1}]$ and consequently $R_{c2c1}^T = R_{c2}^T R_{c1}$ and $t_{c2c1} = -t_{c2} + R_{c2}^T R_{c1} t_{c1}$. Since we are assuming that any radial distortion has already been removed from the images and since the pixel coordinates p are with respect to the optical center, the intrinsic camera model simply scales x_c and y_c by the focal length f and projects the resulting X_i onto the image plane:

$$X_i = PX_c = (fx_c, fy_c, z_c)$$

$$P = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$p = (u, v, f) = (\frac{x_i}{z_i}, \frac{y_i}{z_i}, f) = (\frac{fx_c}{z_c}, \frac{fy_c}{z_c}, f)$$

The non-affine form of X_c can then be simply expressed in terms of pixel coordinates p, [60],

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \frac{z_c}{f} \begin{bmatrix} u \\ v \\ f \end{bmatrix} = \frac{z_c}{f} p$$

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \frac{z_c}{f} \begin{bmatrix} u \\ v \\ f \\ \frac{f}{z_c} \end{bmatrix} = \frac{z_c}{f} p$$

We can now write $X_{c2} = [R_{c2c1}|t_{c2c1}]X_{c1}$ in terms of the image coordinates p with an additional affine element when necessary to arrive at:

$$\frac{z_{c2}}{f} p_2 = [R_{c2c1}|t_{c2c1}] \frac{z_{c1}}{f} p_1 \tag{3.6}$$

Equation 3.6 describes the image motion of stationary 3D points imaged through a perspective camera undergoing rotation and translation.

Approximating the Camera's Motion Model

At this point we introduce a number of approximations in order to arrive at a coarse estimate of the camera's 3D rotation in terms of our affine image motion model A.

First, we substitute $p_2 = Ap_1$ into equation 3.6 which will not be strictly equivalent since it is itself an approximation of the 2D image motion.

$$\frac{z_{c2}}{f} Ap_1 \approx [R_{c2c1}|t_{c2c1}] \frac{z_{c1}}{f} p_1 \tag{3.7}$$

Next, we approximate the full perspective camera with a weak perspective camera. This results in z_{c2} and z_{c1} being constant for all image points p. This approximation works well when the difference in depth between the points is small compared with the average depth of the points. We assume that the affine transformation A describes the image motion of 3D points in the background that correspond with a mostly stationary environment within which the camera is moving, and consequently that the weak perspective model is a reasonable approximation.

We then assume that the image motion resulting from camera translation, t_{c2c1}, is much smaller than the image motion resulting from camera rotation, R_{c2c1}, and that we can consequently set $t_{c2c1} = (0, 0, z_{c2} - z_{c1}) = (0, 0, z_{c1}(\frac{z_{c2}}{z_{c1}} - 1))$ (which can be multiplied by the affine component $\frac{f}{z_c}$ and changed to $t_{c2c1} = (0, 0, \frac{z_{c2}}{z_{c1}} - 1)$ and affine component f) with any actual influence from lateral translation being attributed to noise in our estimates. During everyday activities, head motion does tend to create much larger image motion than translation. We restrict the samples used in the alignment estimation based on the magnitude of the head motion recorded by the head-mounted orientation sensor in order to help ensure that this assumption holds. For example, filtering in this way helps the system avoid interpreting image motion from the floor as camera rotation.

Finally, we throw out the 3^{rd} row of the equation, since A does not give us any information about this row.

These assumptions result in the following approximation for the top two rows of equation 3.6:

$$\frac{z_{c2}}{z_{c1}} A p_1 \approx R_{c2c1} p_1$$

$$\frac{z_{c2}}{z_{c1}} \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{bmatrix} p_1 \approx \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \end{bmatrix} p_1 \tag{3.8}$$

Solving for Rotation

We wish to find values for $\frac{z_{c2}}{z_{c1}}$ and R_{c2c1} that minimize the difference between the two sides of equation 3.8 over the set of points $p_1 = (u, v, f)$, where u and v vary. The six parameters of R_{c2c1} should be constrained to be the top two rows of a rotation matrix and should consequently only have three free parameters describing the two orthogonal unit vectors.

The angular change between images should be small, so we introduce a further approximation and linearize R_{c2c1} around zero angular change, which corresponds to the identity matrix I. For this linearization we represent R using Euler angles α, β and γ, which are a suitable representation for small angular changes.

$$R(\alpha, \beta, \gamma) = R_x(\alpha) R_y(\beta) R_z(\gamma)$$

$$R(\alpha,\beta,\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\gamma) & sin(\gamma) & 0 \\ -sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We now take the gradient of R with respect to the Euler angles

$$\bigtriangledown R = \begin{bmatrix} D[R]_\alpha & D[R]_\beta & D[R]_\gamma \end{bmatrix} = \begin{bmatrix} D[R_x]_\alpha R_y R_z & R_x D[R_y]_\beta R_z & R_x R_y D[R_z]_\gamma \end{bmatrix}$$

$$\bigtriangledown R = \begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & -sin(\alpha) & cos(\alpha) \\ 0 & -cos(\alpha) & -sin(\alpha) \end{bmatrix} R_y R_z \\ R_x \begin{bmatrix} -sin(\beta) & 0 & -cos(\beta) \\ 0 & 0 & 0 \\ cos(\beta) & 0 & -sin(\beta) \end{bmatrix} R_z \\ R_x R_y \begin{bmatrix} -sin(\gamma) & cos(\gamma) & 0 \\ -cos(\gamma) & -sin(\gamma) & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}^T$$

and linearize around zero rotation $\omega = (\alpha, \beta, \gamma) = (0, 0, 0)$, which is simply a first order truncation of the Taylor expansion of R

$$R(\theta + \omega) \approx R(\theta) + \bigtriangledown R(\theta)\omega + o(\omega^2)$$

$$R(0 + \omega) \approx I + \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \omega + o(\omega^2)$$

$$R(0 + \omega) \approx \begin{bmatrix} 1 & \gamma & -\beta \\ -\gamma & 1 & \alpha \\ \beta & -\alpha & 1 \end{bmatrix} + o(\omega^2)$$

We now vectorize the equation and use linear least squares to minimize the distance between the vectorized matrices with respect to $\frac{z_{c2}}{z_{c1}}$ and ω, which corresponds to minimizing the Frobenius distance between the matrices

$$argmin \left(\left\| \frac{z_{c2}}{z_{c1}} \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{bmatrix} - \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \end{bmatrix} \right\| \right)_{\frac{z_{c2}}{z_{c1}}, \omega}$$

For which we replace R with our linear approximation

$$\begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \end{bmatrix} \approx I + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \alpha + \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \beta + \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \gamma$$

which after vectorizing the matrices gives us the following minimization formula

$$argmin \left(\left\| \begin{bmatrix} a_1 & 0 & 0 & 0 \\ a_2 & 0 & 0 & -1 \\ a_3 & 0 & 1 & 0 \\ a_4 & 0 & 0 & 1 \\ a_5 & 0 & 0 & 0 \\ a_6 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{z_{c2}}{z_{c1}} \\ \alpha \\ \beta \\ \gamma \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\| \right)_{\frac{z_{c2}}{z_{c1}}, \omega}$$

which linear least squares minimizes, since it's in the standard form $Ax = b$ with

$$A = \begin{bmatrix} a_1 & 0 & 0 & 0 \\ a_2 & 0 & 0 & -1 \\ a_3 & 0 & 1 & 0 \\ a_4 & 0 & 0 & 1 \\ a_5 & 0 & 0 & 0 \\ a_6 & -1 & 0 & 0 \end{bmatrix}, \; x = \begin{bmatrix} \frac{z_{c2}}{z_{c1}} \\ \alpha \\ \beta \\ \gamma \end{bmatrix} \; b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Finally, we construct a rotation matrix using our linear least squares estimate for ω. The linearized estimate for R_{c2c1} is not guaranteed to be a valid rotation matrix. We could plug ω into $R(\alpha, \beta, \gamma) = R_x(\alpha)R_y(\beta)R_z(\gamma)$, which would be a reasonable estimate given our small angles, but it unnecessarily imposes an ordering on the three rotations, which is not dictated by our linearization, and it will not necessarily be the closest rotation matrix to our linearized estimate. We use a standard SVD based method from [60] to construct the closest rotation matrix to the full linearized estimate $\widetilde{R_{c2c1}}$ of R_{c2c1}. More specifically we construct

$$\widetilde{R_{c2c1}} = I + \left[\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right]\omega\right]$$

compute its SVD

$$UDV^T = SVD(\widetilde{R_{c2c1}})$$

and then construct our final estimate for R_{c2c1}, where I is the identity matrix.

$$R_{c2c1} \approx UIV^T$$

Evaluation

This coarse estimate of camera motion is important to several later estimates. It is used to find the assignment from sensors to body parts. It is also used to estimate the orientation of the camera with respect to the head-mounted orientation sensor, which in turn plays an important role in estimating the kinematic model. Ultimately, these success of these later estimates serve as the most important evaluation of the success of these coarse estimates of camera motion. We can, however, get some sense of its isolated performance by comparing its estimates to those made by the head orientation sensor, see figures 3-11 and 3-12.

3.6 Aligning the Camera with the Head Orientation Sensor

This section describes a linear least squares method that estimates the orientation of the head-mounted camera with respect to the head-mounted orientation sensor. This estimation uses the series of orientations provided by the orientation sensor along with the series of camera rotation estimates provided by the methods of image motion analysis described within the previous two sections. Since the head-mounted orientation sensor measures its rotation with respect to world coordinates, this method allows us to estimate the orientation of the head-mounted camera with respect to world coordinates.

The head orientation sensor provides an orientation R_h that represents the orientation of the sensor with respect to world coordinates, where the world coordinates are in terms of gravity and the earth's magnetic field. We would like to know the orientation of the

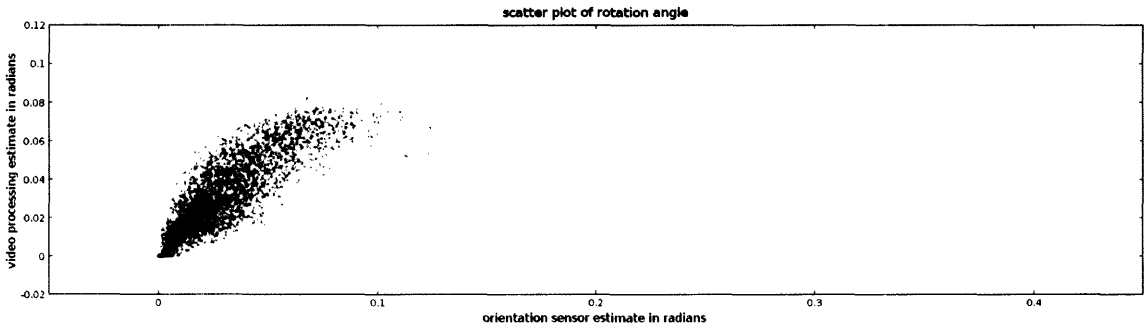


Figure 3-11: This figure shows a histogram of the relationship between the rotational change measured by the orientation sensor and the rotational change measured by the image motion method of this section. Ideally the two values would be perfectly correlated as a line of unit slope. The values are strongly correlated, which indicates that camera rotation estimate from image motion does usefully relate to the true camera rotation. Also note that above a change of approximately 0.12 radians between the orientation of the camera in the two images, the motion processing algorithm misestimates the magnitude of the motion. This outlier error relates to the constrained search window of the underlying block matching algorithm for motion estimation. For this run, we did not perform full low-resolution image motion first, which exacerbates the problem. Fortunately, we can reduce the impact of these outlier errors by throwing out our image based camera rotation estimates when the change measured by the head mounted sensor is above a threshold, such as 0.12 radians.

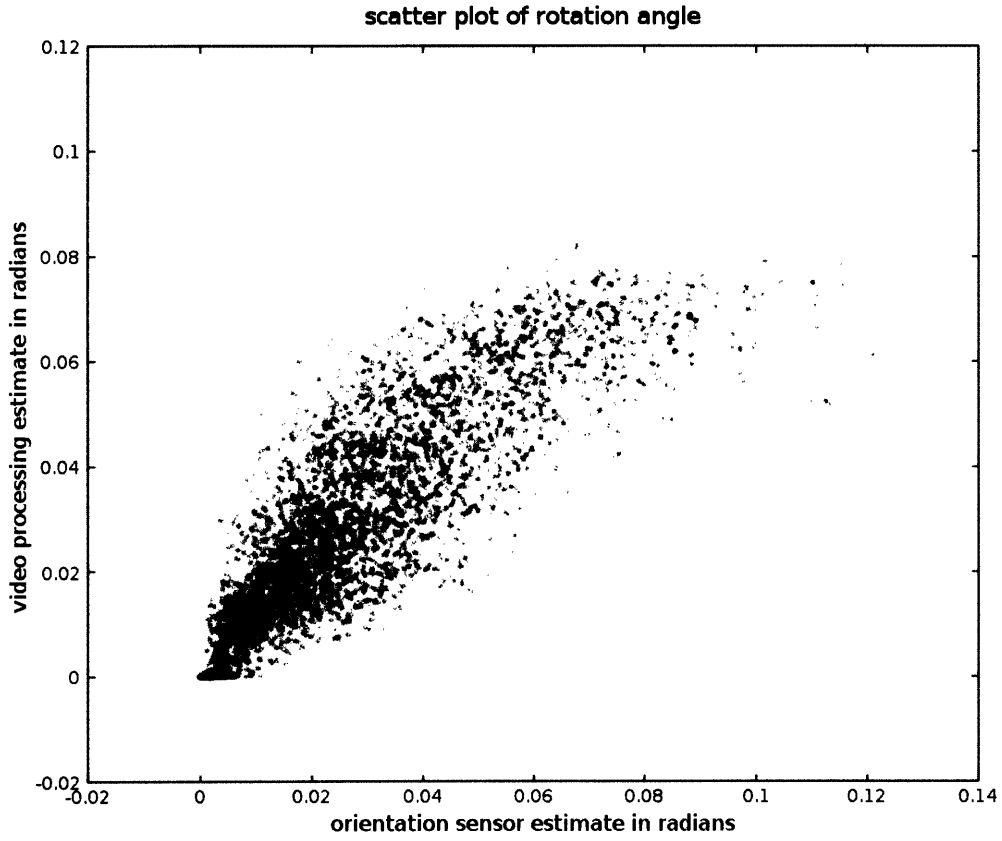


Figure 3-12: This figure shows the same histogram of figure 3-11 zoomed in to the area that would be relevant if we were to throw out outliers based on the head's angular velocity as measured by the head mounted orientation sensor.

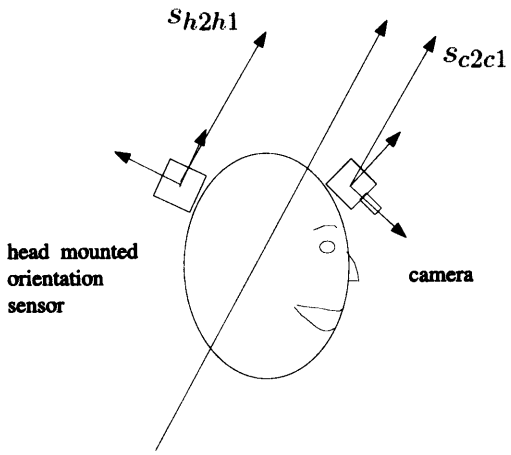


Figure 3-13: This diagram illustrates the geometry of the camera alignment problem. Because the head, the head orientation sensor, and the camera are rigidly attached to each other, they share the same axis of rotation. Consequently, $s_{c2c1} = R_{hc}s_{h2h1}$ where s_{c2c1} is the view of the shared axis of rotation in the camera's local coordinate system, and s_{h2h1} is the view of the shared axis of rotation in the head orientation sensor's local coordinate system.

camera R_c in the world coordinate system. We assume that for long periods of time there is a constant orientation offset R_{hc} in world coordinates that brings the head orientation sensor into alignment with the camera,

$$R_c = R_h R_{hc}$$

Between sessions, R_{hc} is likely to change. Occasionally during a session the alignment between the two sensors may also change due to comfort adjustments or collisions. These variations are the price we pay for having compliant head gear with a non-rigid coupling between the orientation sensor and the camera. For real-time applications we would like to estimate R_{hc} rapidly and in an online fashion so that the system can quickly adjust to a new wearer and monitor the alignment throughout a session in order to adapt to occasional changes.

In order to estimate R_{hc}, we relate the between image rotational motion of the camera to the between image rotation of the orientation sensor. The camera rotation estimate R_{c2c1} from section 3.5 estimates the rotation of the camera between images I_1 and I_2 with respect to the camera's own reference frame.

$$R_{c2c1} = R_{c1}^T R_{c2}$$

Similarly, measurements from the head-mounted orientation sensor for images I_1 and I_2 can be used to compute, R_{h2h1}, which describes the rotation of the sensor between images I_1 and I_2 with respect to the world reference frame.

$$R_{h2h1} = R_{h1}^T R_{h2}$$

We would like to estimate R_{hc} using our estimate of the camera rotation R_{c2c1} and the rotation of the orientation sensor R_{h2h1}. The most successful method we've found for estimating R_{hc} aligns the estimated axes of rotation in the local coordinate systems using linear least squares, see figure 3-13. Over most lengths of time in a particular episode, the camera and orientation sensor are well-modeled as being rigidly affixed to the head. Consequently, the camera, the orientation sensor and the head form a rigid body. When this rigid body rotates, the camera and the orientation sensor share the same axis of rotation. By aligning the local views of this shared axis of rotation, we can align the local coordinate systems of the camera and orientation sensor.

More precisely, when the rigid body undergoes a rotation, R_{21}, between two frames, I_1 and I_2, this rotation can be represented by an axis of rotation s_{12} and an angle α_{12}. The orientation sensor and the camera view this axis of rotation, s_{12}, in their local coordinate systems as s_{h2h1} and s_{c2c1} respectively. A rotation that brings s_{c2c1} into alignment with s_{h2h1} will globally align the local coordinate systems along two degrees of freedom.

$$s_{c2c1} = R_{hc} s_{h2h1} \tag{3.9}$$

The remaining degree of freedom corresponds with rotations around this shared axis of rotation and has indeterminate alignment, since the rotation α_{12} around the shared axis is relative to the current orientations of the local coordinate systems. We can align this remaining degree of freedom and compensate for noise in our estimates by combining rotation estimates from many frames due to the assumption that the aligning rotation R_{hc} is constant. By vectorizing R_{hc} in equation 3.9, we can create a linear least squares formulation that combines estimates from many sequential frames:

$$\begin{bmatrix} s_{h2h1}^T & 0 & 0 \\ 0 & s_{h2h1}^T & 0 \\ 0 & 0 & s_{h2h1}^T \\ s_{h3h2}^T & 0 & 0 \\ 0 & s_{h3h2}^T & 0 \\ 0 & 0 & s_{h3h2}^T \\ \vdots & \vdots & \vdots \\ s_{hnh(n-1)}^T & 0 & 0 \\ 0 & s_{hnh(n-1)}^T & 0 \\ 0 & 0 & s_{hnh(n-1)}^T \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \end{bmatrix} = \begin{bmatrix} s_{c2c1} \\ s_{c3c2} \\ \vdots \\ s_{cnc(n-1)} \end{bmatrix}$$

Where the entries r_i are the vectorized elements of the rotation matrix R_{hc} and $s_{h_i h_{(i-1)}}$ and $s_{c_i c_{(i-1)}}$ are unit vectors representing the estimated axis of rotation between frames $i-1$ and i in the local coordinate systems of the head and camera respectively. This equation is in the standard $Ax = b$ form and can be simplified and made computationally trivial by using the pseudo-inverse $\left(A^T A\right)^{-1} A^T$ to solve for x, with

$$x = \left(A^T A\right)^{-1} A^T b$$

and

$$A^T A = \begin{bmatrix} \sum_i s_{h_i h_{(i-1)}} s_{h_i h_{(i-1)}}^T & 0 & 0 \\ 0 & \sum_i s_{h_i h_{(i-1)}} s_{h_i h_{(i-1)}}^T & 0 \\ 0 & 0 & \sum_i s_{h_i h_{(i-1)}} s_{h_i h_{(i-1)}}^T \end{bmatrix} = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix}$$

where S is the sum of the outer products of the rotation axes with themselves, and

$$A^T b = \sum_i \begin{bmatrix} s_{h_i h_{(i-1)}} & 0 & 0 \\ 0 & s_{h_i h_{(i-1)}} & 0 \\ 0 & 0 & s_{h_i h_{(i-1)}} \end{bmatrix} s_{c_i c_{(i-1)}}$$

A key point to recognize with this solution is that the summations can be incremented by an online process that continually reestimates $A^T A$ and $A^T b$ by adding the new terms associated with the current frames i and $i-1$. After updating them we compute $\left(A^T A\right)^{-1}$ which, if we desire, can be further simplified by only finding S^{-1} and constructing the block inverse with S^{-1} along the diagonal. Finally, we multiply these easily updated terms to find x, $x = \left(A^T A\right)^{-1} A^T b$. Moreover, we can recursively weight the incoming estimates with the past estimates to have a fast online and adaptive estimate of R_{hc} that will compensate for occasional changes in alignment between the camera and the head orientation sensor.

Results

We have tested this method of estimating R_{hc} on both ideal data, shown in figure 3-14, and actual data from the system 3-16. Evaluating the quality of the solution is challenging, since we don't know the true orientation of the camera within the world. We can qualitatively assess the correctness of the solution within human environments by observing the edges within the image that point to the vanishing point associated with gravity, as predicted by the camera's global orientation, see figures 3-17, 3-18, and 3-19. In most urban indoor and outdoor human environments, a significant percentage of the edges are aligned with gravity. Besides serving as an efficient and intuitive way to qualitatively assess the estimated world orientation of the camera, selecting edges by how well they are aligned with gravity or other known world orientations - such as the forearm - can enhance perceptual processing, see appendix A. We have pursued, but not completed, a related method for quantitatively assessing the estimated camera orientation that sums up the responses of edges associated with vanishing points, finds locally maximal vanishing points, and then compares the estimated gravity vanishing point with the locally maximal vanishing points. A proper camera

orientation estimate should produce a gravity related vanishing point that is usually very close to a maximal vanishing point when within urban human environments.

One source of error is the different sampling rates used by the orientation sensing process, which samples at around $100Hz$, and the image capturing process, which samples at around $10-25Hz$. In this work, for each image we simply use the orientation sample that is closest in time to each image. For better accuracy we could interpolate between time stamped orientations to produce estimates for the orientation of the head at the time stamp associated with an image, for example using quaternions and SLERP.

3.7 Estimating Joint Lengths and Rotational Offsets

In our idealized handmade kinematic model from section 3.1, the orientations are independent from the joint lengths, such that given any set of orientations the lengths of the body parts could be longer or shorter. This lack of position information makes our estimation problem distinct from automated kinematic model estimation from traditional motion capture data, which almost always includes position information [47]. For real human bodies, the measured orientations are mostly invariant to the lengths of the body parts. Given a time series of orientation measurements, some process might be able to extract subtle information about the lengths of body parts from the dynamics of the body's motion, for example by finding natural modes of motion during walking and combining this information with the statistics of body dimensions from a large population study. Rather than resort to such challenging and subtle methods, we use additional information provided by the camera to directly estimate joint lengths for a kinematic model. Specifically, in addition to the orientations provided by the body-mounted orientation sensors, the methods within this chapter require the estimated orientation of the camera and the estimated position of the wearer's hand within the images. The previous sections presented the methods we use to estimate the orientation of the camera, R_c, from image motion, where, as required, R_c is specified with respect to the shared world coordinate system of the body-mounted orientation sensors. The required information about the hand's position with the images can be estimated in a number of ways, one of which we present within this section.

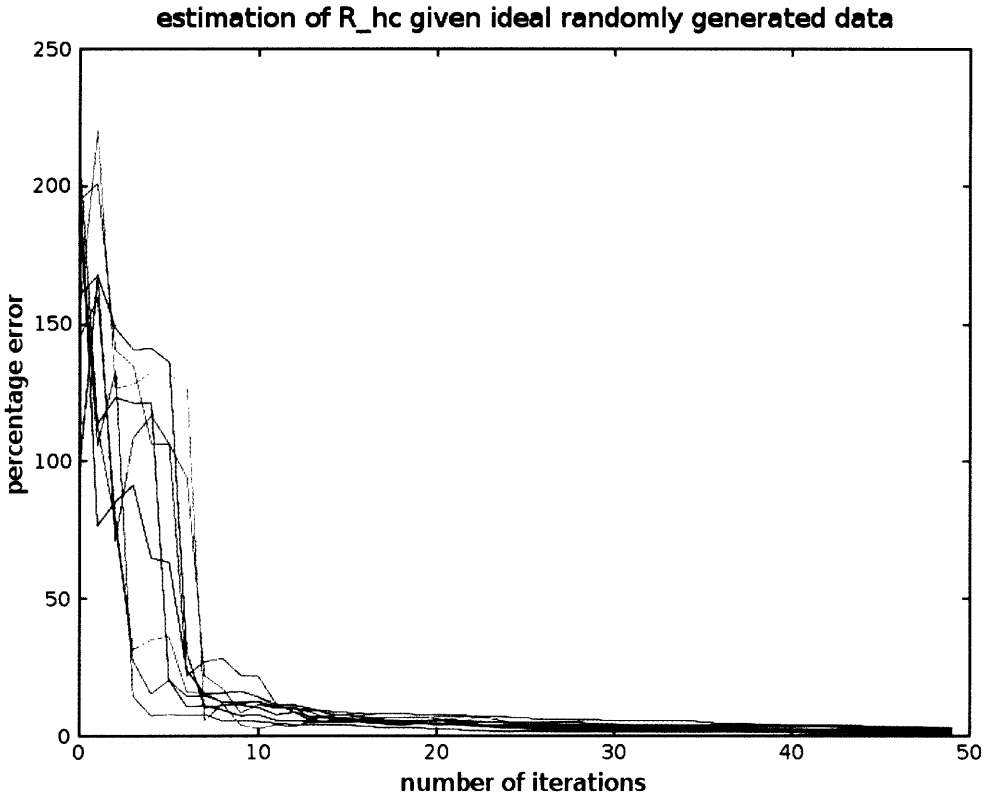


Figure 3-14: This figure shows characteristic results of estimating R_{hc} using the algorithm from this section with ideal randomly generated data. The data is generated by assuming an ideal unknown noiseless offset R_{hc}, and ideal known noiseless values for R_{h2h1} and R_{c2c1} on each frame. Between each frame an ideal rotation is randomly selected and applied to the camera and the head from which R_{h2h1} and R_{c2c1} are calculated. This test demonstrates that under ideal conditions the algorithm will quickly estimate R_{hc}. Under real world conditions the algorithm also works efficiently and accurately. The final average error for these ten test runs was approximately 2 after 50 iterations. The error shown is $100 * \frac{\|R_{hc} - \tilde{R_{hc}}\|}{\|R_{hc}\|}$ where the magnitude gives the Frobenius norm of the matrix. The error continues to drop rapidly given more iterations. These particular tests used randomly generated rotations specified in Euler angles from a uniform distribution of range $[-0.02\pi \ 0.02\pi]$ for each Euler angle, with a constant offset R_{hc} with an axis of [1 0 0] and angle of $\frac{1}{2}\pi$. Changing any of these parameters has little effect on the results.

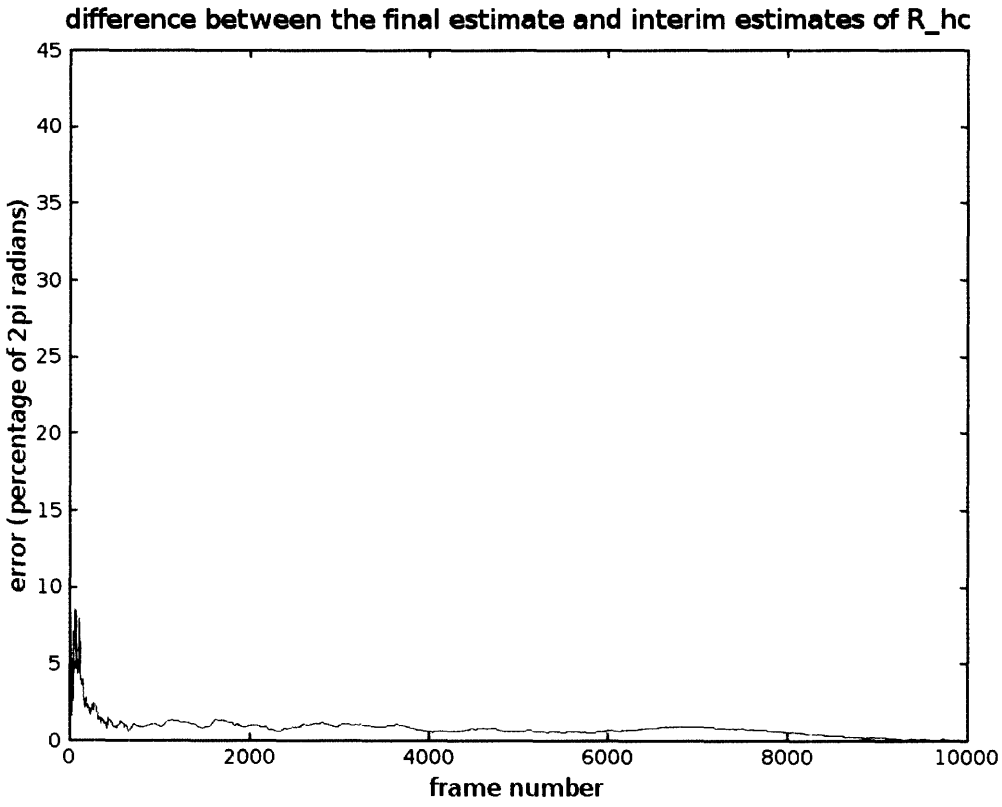


Figure 3-15: This figure shows the convergence graph of the estimate for R_{hc} on dataset 1. We can not be sure of the true solution, so we measure the difference between interim estimates and the final estimate to gain some insight into how stable the final estimate is and how quickly it is reached.

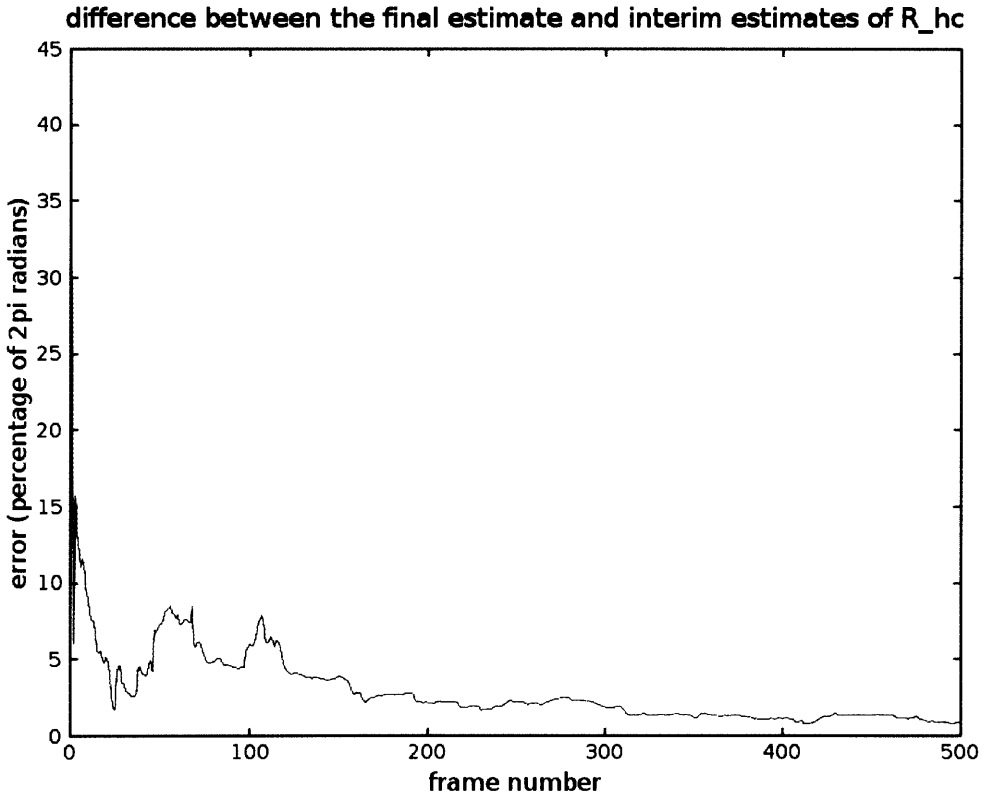


Figure 3-16: This figure is a zoomed in version of figure 3-15 to illustrate the details of the convergence speed. Notice that within these 500 frames the estimate for R_{hc} converges to a value very close to the value it maintains for the rest of the 10000 frames. It converges in less than a minute and gives useful values in just a few seconds.

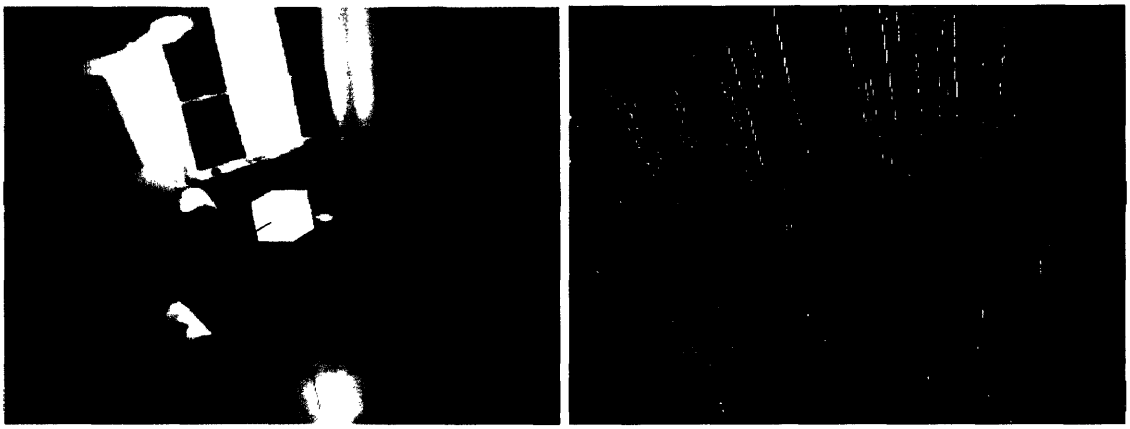


Figure 3-17: This figure and the three matching frames show qualitative results for the estimate of R_{hc}. The left frame shows an image of a cube and axes aligned with the global coordinate system. The image on the right shows the Canny edges weighted by how closely the point toward the vanishing point associated with gravity as estimated using R_{hc} and R_h.

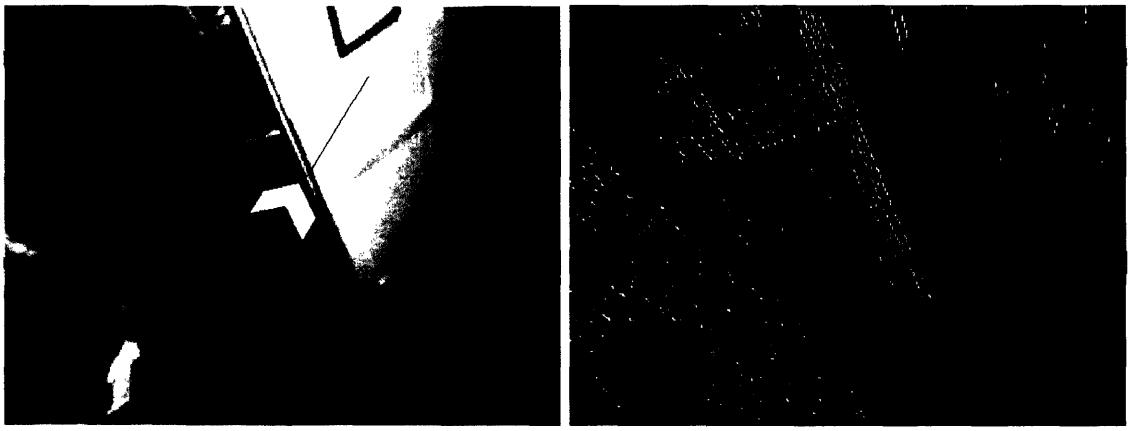


Figure 3-18: This figure and the three matching frames show qualitative results for the estimate of R_{hc}. The left frame shows an image of a cube and axes aligned with the global coordinate system. The image on the right shows the Canny edges weighted by how closely the point toward the vanishing point associated with gravity as estimated using R_{hc} and R_h.

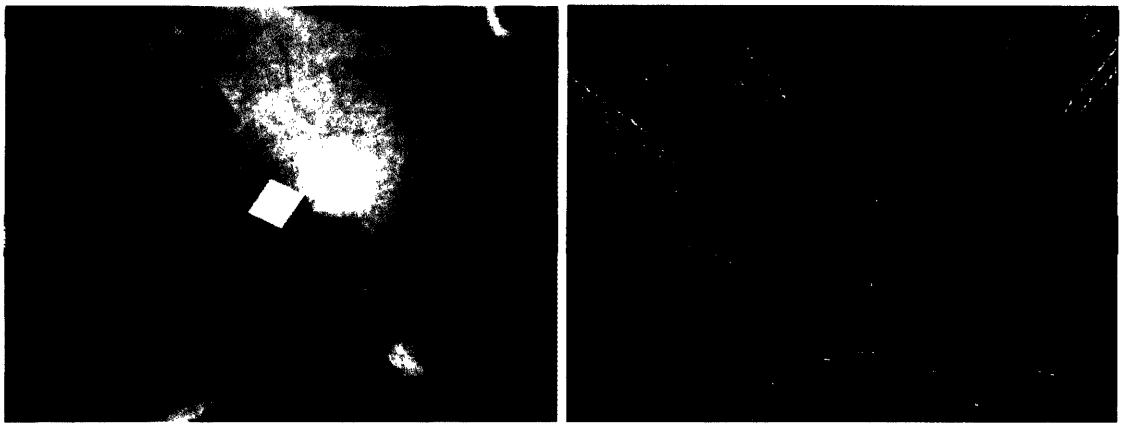


Figure 3-19: This figure and the three matching frames show qualitative results for the estimate of R_{hc}. The left frame shows an image of a cube and axes aligned with the global coordinate system. The image on the right shows the Canny edges weighted by how closely the point toward the vanishing point associated with gravity as estimated using R_{hc} and R_h.

3.7.1 A Linear Least Squares Estimate

We first present a method that uses linear least squares to estimate the kinematic model. Given only the hand's 2D position within images, this non-projective method is sufficient to estimate a model that predicts the hand's position in new images based solely on the measured orientation, but the resulting joint lengths are distorted due to projective effects. If the hand detection method is able to provide a reasonable estimate of the 3D position of the wearer's hand with respect to the camera, this method should be able to also generate good estimates of the joint lengths.

The Linear Least Squares Formulation

First, assume that we have samples of the 3D location of the hand X_h with respect to the camera's frame of reference and the associated world frame rotation matrices for the camera, torso, upper arm and lower arm, $\{R_c, R_t, R_u, R_f\}$. With the following linear least squares equation we can estimate the four 3D vectors $\{j_c, j_t, j_u, j_f\}$ that join the camera at the origin $X_c = (0, 0, 0)$ to the hand at X_h through the rotation matrices $\{R_c, R_t, R_u, R_f\}$.

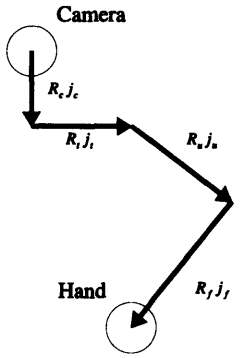


Figure 3-20: The kinematic model with respect to the world frame with the camera as the origin.

$$\begin{bmatrix} R_{c1}^T R_{c1} & R_{c1}^T R_{t1} & R_{c1}^T R_{u1} & R_{c1}^T R_{f1} \\ R_{c2}^T R_{c2} & R_{c2}^T R_{t2} & R_{c2}^T R_{u2} & R_{c2}^T R_{f2} \\ \vdots & \vdots & \vdots & \vdots \\ R_{cn}^T R_{cn} & R_{cn}^T R_{tn} & R_{cn}^T R_{un} & R_{cn}^T R_{fn} \end{bmatrix} \begin{bmatrix} j_c \\ j_t \\ j_u \\ j_f \end{bmatrix} = \begin{bmatrix} X_{h1} \\ X_{h2} \\ \vdots \\ X_{hn} \end{bmatrix} \tag{3.10}$$

$$R_{xyz} j = X_h \tag{3.11}$$

This is in the standard $Ax = b$ form that we expect for linear least squares. The entries of A are rotation matrices that rotate the body vectors $\{j_c, j_t, j_u, j_f\}$, first into the current orientation of the associated body part and then into the camera's reference frame. The first column of A is composed of identity matrices, since j_c is a constant vector within the camera's reference frame. The orientations of the body vectors can be used to partly align the rotation matrices into canonical orientations. However, they do leave rotations around each body vector's axis unnormalized. For example, wrist rotations around j_l will not be rotated into a canonical orientation. This is not a problem for predicting the location of the hand and estimating the major dimensions of the body parts, but it does mean that the resulting orientations are not directly comparable when found with distinct model fitting.

The vectors $\{j_c, j_t, j_u, j_f\}$ implicitly encode both the lengths the body parts, $\{l_c, l_t, l_u, l_f\}$, and appropriate orientation offsets, R_{offset}, of the body parts with respect to the sensors. The lengths and orientation offsets of the upper-arm and forearm, j_u and j_f, approximately

match the associated parameters that we determined by hand in section 3.1, while the lengths and orientation offsets of the torso and camera, j_t and j_c, do not directly map to our handmade model since they approximately define the linkage from the camera to the neck and from the neck to the shoulder joint. The lengths of the linkages are equal to the magnitudes of the vectors of matrix j.

$$\begin{bmatrix} l_c \\ l_t \\ l_u \\ l_f \end{bmatrix} = \begin{bmatrix} \|j_c\| \\ \|j_t\| \\ \|j_u\| \\ \|j_f\| \end{bmatrix}$$

Each implicitly defined orientation sensor offset rotates a unit vectors $\widehat{u}$ into the direction of the appropriate vector of j.

$$\begin{bmatrix} j_c \\ j_t \\ j_u \\ j_f \end{bmatrix} = \begin{bmatrix} R_{offset_c} & R_{offset_t} & R_{offset_u} & R_{offset_f} \end{bmatrix} \begin{bmatrix} l_c\widehat{u} \\ l_t\widehat{u} \\ l_u\widehat{u} \\ l_f\widehat{u} \end{bmatrix} \tag{3.12}$$

The rotational offsets $\{R_{offset_c}, R_{offset_t}, R_{offset_u}, R_{offset_f}\}$ are underspecifed, since rotations around the axis of $\widehat{u}$ will not effect the equality of equation 3.12. As mentioned above, this results from the estimation process. For example, rotations around the axis of the forearm from wrist twisting will not affect the location of the hand. If we wish to normalize this aspect of the measured orientations across different sessions and bodies, we could do so by monitoring the ranges of motion around the linkages during common activities and matching up these ranges of motion to a canonical range. For example, wrist twisting occurs over a fixed range due to joint limits and comfort - constraints on the distribution that we could use to pick a common rest orientation across different sessions and bodies.

2D Hand Detection

Equation 3.10 assumes that we have the 3D location of the hand in the camera's frame of reference. If our hand detection system is only able to output a 2D hand location in image coordinates, we can ignore projective effects, use a non-projective orthogonal camera model, and solve the resulting modified version of 3.10. This modified version removes the rows

that involve depth (every 3^{rd} row of the equation), and substitutes the non-projective image coordinates for the original projective x, y components of the hand positions X_h. Solving this modified form results in a useful kinematic model that gives the approximate angles of the body parts with respect to the sensors, predicts the hand location within images fairly well when the hand is not too close to the camera, and can be used to estimate whether or not the hand should be visible within an image given the angles of the body parts. Unfortunately, since the hand is a near-field object, this orthographic model performs poorly in many situations. The model's joint lengths are distorted due to projective effects, hand position estimates are poor when the hand is close to the camera, and, perhaps most significantly, the model does not successfully predict when the hand will not be visible in the image.

3D Hand Detection

Alternatively, we can attempt to create useful estimates of the 3D location of the hand. By a variety of methods we can detect the hand in an image and estimate its projected location and size. The projected location and size of the hand strongly relate to the hand's 3D location, but they do not give us exact estimates. Both vary based on the viewing angle and the configuration of the hand, which has many degrees of freedom, as one would expect from such a compact and highly articulate manipulator. Without additional information we cannot estimate the metric depth, z_c, of the hand with our monocular camera. We could use a distribution over expected hand size to help constrain the depth.

Fortunately, in order to make equation 3.10 useful, we only need to estimate the depth of the hand up to a constant scaling factor, since the equation holds for any constant scaling factor multiplied by $\{j_c, j_t, j_u, j_f\}$ and $X_{hi} = \frac{z_{hi}}{f} \begin{bmatrix} u & v & f \end{bmatrix}^T$. This allows us to solve for αz_{hi} and the relative body lengths $\{\alpha j_c \alpha, j_t \alpha, j_u \alpha, j_f\}$ and impose a metric scale later if we desire. For hand prediction we do not need to estimate a metric scaling factor since α divides out when computing (u, v). For our camera model, the projected area a_p of a 3D planar patch parallel to the image plane with visible surface area of a_s obeys the equation $a_p = \frac{f^2}{z^2} a_s$ so that $z = f\sqrt{\frac{a_s}{a_p}}$. We can solve for z up to an unknown constant scaling factor α by setting $f^2 \alpha^2 a_s = c$ where c is a constant and $c > 0$, which results in

$$\alpha z = \sqrt{\frac{c}{a_p}}$$

This is a good approximation for an object whose visible surface varies little and whose visible surface depth is much smaller than the distance to the image plane at $z = f$. Unfortunately, the visible surface of the hand can vary considerably by viewing angle and configuration. For example the visible surface of the knife edge of the hand differs from the palm of the hand with fingers outstretched by around a factor of 5. Also, the hand can come very close to the face and hence the camera. We can mitigate these problems by throwing out samples and performing more advanced area estimations that account for hand angle and configuration, but we should expect noise in our estimates of αz.

Much as we did for motion processing in section 3.4, we could reduce the effects of uncertainty by introducing a covariance matrix, U^{-1}, that describes the measurement error. U would be composed of covariance matrices and weighting terms in order to incorporate Gaussian models of the uncertainty in our estimates of X_h. We did not test this approach.

If we could confidently produce 3D estimates of the hand's position, this linear least squares model would be highly desireable, since it's estimate could be computed in real-time. As we will describe next, we did not pursue this solutions, and instead developed an offline, non-linear method. Finding suitable real-time 3D hand detection methods to couple with this linear least squares method for kinematic model estimation would be a worthwhile endeavor for future research into real-time wearable systems with kinematic sensing.

3.7.2 A Nonlinear Estimate

Rather than burden ourselves with estimating the 3D position of the hand up to a scaling factor α, we can instead reformulate the problem as a nonlinear optimization problem at the cost of additional computation.

We construct the the objective function, $c(j)$, using the cost function $f(j)$ and a penalty term $s_{penalty}(j)$.

$$c(j) = f(j) + \alpha s_{penalty}(j)$$

The scalar constant α determines the relative importance of the two terms.

The main cost function $f(j)$ measures the differences between the 2D visually estimated hand positions, x_h, and the hand positions predicted by the measured orientations, R, used with a kinematic model and a fully projective camera model. Specifically,

$$R = \begin{bmatrix} R_c^T R_c & R_c^T R_t & R_c^T R_u & R_c^T R_f \end{bmatrix}$$

$$f(j) = \sum_i g\left(\left\| f\frac{R_{xy_i}j}{R_{z_i}j} - x_{h_i} \right\|\right)$$

where R_{xy} is the $x$and $y$components (first and second rows) of the block matrix of full rotation matrices R, and R_z is the z component (third row). The division by R_z is an element-wise division, so that $f\frac{R_{xy_i}j}{R_{z_i}j}$ is simply the kinematic model's hand position estimate. The magnitude measures the Euclidean distance between this estimate and the visual estimate x_h in image pixels, and the cost function, g, can be used to make the total cost more robust to outliers. We make g constant beyond a threshold, d_{max}, so that the estimation can better handle outliers, such as false positive hand detections when the wearer's hand is not even visible due to the body's configuration.

$$g(d) = \begin{array}{l} d \text{ if } d < d_{max} \\ d_{max} \text{ otherwise} \end{array}$$

We found that making g robust to outliers helps significantly when optimizing $c(j)$ with real hand position estimates. Using an alternative robust cost function for g with some slope information when $d \geq d_{max}$ might help with optimization, but we have had success with simply clipping the maximum error.

The penalty term $s_{penalty}(j)$ encourages the sum of the joint lengths to stay close to 1,

$$s_{penalty}(j) = \text{abs}(1 - \|j\|)$$

since the objective function $c(j)$ is invariant to the length of j due to projection.

$$f\frac{R_{xy_i}\beta j}{R_{z_i}\beta j} = f\frac{R_{xy_i}j}{R_{z_i}j}$$

This term can improve the speed of convergence by removing an irrelevant degree of freedom. It also makes the exact results of the optimization more comparable.

A great variety of algorithms exist for optimizing a nonlinear objective function such as $c(j)$. We use the Nelder-Mead Simplex algorithm provided by the optimization module of SciPy, a numerical analysis package for Python [28]. This optimization algorithm gives us flexibility in defining the objective function since it only requires function evaluations and does not make use of the gradient or Hessian of the objective function. This ease of use, however, most likely comes at the cost of additional search and computation. We've had

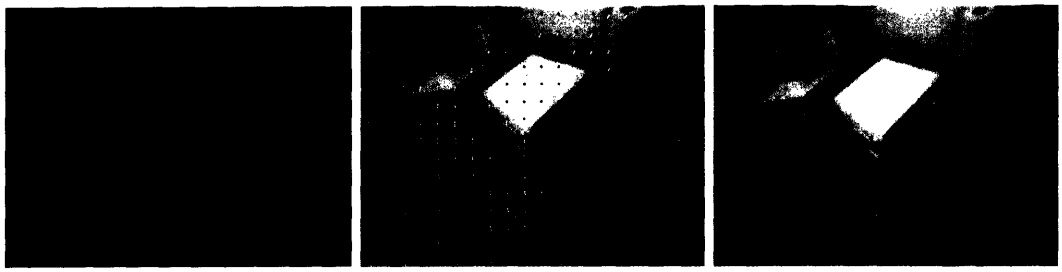


Figure 3-21: This frame shows an example that illustrates the motion signal used to generate hand estimates. Notice that points around the hand clearly have the strongest responses as their motion differs significantly from the projected motion of the environment due to head motion. The left image is the Mahalanobis distance between the measured edge motions and the 2D affine background motion model. The middle image shows the first image in time with a vector field sampled from the 2D affine motion model. The right image shows the second image in time to which the 2D affine motion model maps edge points in the first image.

success by running this optimization algorithm with the search initialized with a randomly selected 12 dimensional unit vector for j. To ensure that we find a more global optimum, we typically run the optimization around 100 times and select the lowest cost result. We have sometimes re-run the optimization after removing outliers, which can lead to modest improvements in the solution. Much more significant improvements could be obtained by using the hand predictions produced by a learned model to better detect the hand in the images, and then using these improved hand predictions to retrain the model.

3.8 Offline Hand Discovery

Both the linear and nonlinear methods for estimating the kinematic model require hand position estimates. For offline estimation we can provide suitable hand estimates by finding locations of high foreground motion within the image using the motion segmentation system we described in section 3.4.4. Within the workspace of the hand, the hand tends to be the fastest moving object. The human hand sits at the end of a long, high-powered, kinematic chain that moves it rapidly in 3D space. Due to our first person perspective, the hand is always near the camera, so these fast 3D motions project to fast motions within the

image. Other than the image motion resulting from head rotation, the image motion of the hand is often the largest. Consequently, positions corresponding with the largest foreground motion, as measured by the motion segmentation system, serve as noisy, but informative, estimates of the hand's position, see figure 3-21.

During any specific frame, the largest motion within the image may result from another object within the environment or even an object held by the wearer's hand. Likewise, the hand is not always moving and is not always visible to the camera. We address these issues by clustering the camera normalized orientation sensor data and then combining the motion estimates from images associated with the same cluster. For the images within the same cluster, the hand should be in approximately the same position within the image, since the orientations of the body-mounted sensors with respect to the camera are very similar, while other sources of motion should be more evenly distributed. Consequently, we can filter out many of the noise hypotheses and find body configurations for which the hand is not visible.

A Probabilistic Model

We can model this relationship in terms of probability, with a position of maximum foreground image motion generated by either the hand position distribution, p_h, or the noise distribution, p_n, as represented by the mixture distribution, p_x,

$$p_x = \alpha p_h + (1 - \alpha) p_n$$

We model p_h as being influenced by the random variable, c, which represents the configuration of the wearer's body with respect to the camera,

$$p_{h,c} = p_{h|c} p_c$$

and the noise p_n as being independent of c,

$$p_{n|c} = p_n$$

so that the joint distribution over x and c is

$$p_{x,c} = \alpha p_{h|c} p_c + (1 - \alpha) p_n p_c$$

which we can condition on c, which is known

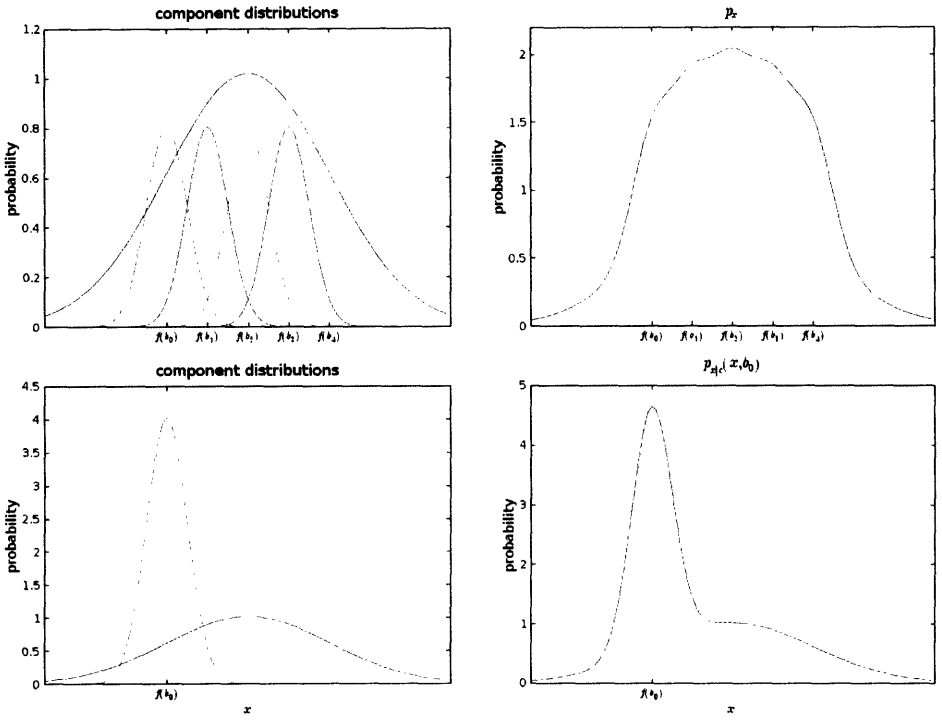


Figure 3-22: This figure illustrates the benefit of conditioning the hand position estimation on the body configuration c. The images on the right show the hidden component distributions that are mixed to give the full distributions that we can measure in the right column. The top row shows the unconditioned distribution p_x, which leads to the distributions associated with body configurations b_0, b_1, b_2, b_3, and b_4 being mixed with the background noise model p_n. The background noise model, p_n, is in blue in the left column. The bottom row shows $p_{x|c}(x, b_0)$, which is the result of conditioning the distribution, p_x, in the first row by body configuration b_0 associated with position $f(b_0)$. Since the noise model is independent of the body configuration the conditioned distribution has a strong peak around $f(b_0)$, which is the position we wish to estimate, and is consequently much easier to detect.

$$p_{x|c} = \frac{p_{x,c}}{p_c} = \alpha p_{h|c} + (1-\alpha)p_n$$

Additionally, we make the mixture probability α a function of the body configuration b

$$\alpha(b) = \begin{array}{l} \beta \text{ if the hand is visible given body configuration } b \\ 0 \text{ if the hand is not visible given body configuration } b \end{array}$$

Now we model $p_{h|c}$ as a unimodal distribution translated around the image plane based on the body configuration. Specifically u is a unimodal distribution with maximum value at $(0,0)$ that is translated by a 2D image position returned by f, which returns the ideal image position of the hand as a function of the body configuration, b.

$$p_{h|c}(m,b) = u\left(m - f(b)\right)$$

Assuming the translations by f are distributed over some area of the image and that $p_c(b)$ is not degenerate, it's clear that conditioning the distribution p_h on c increases the peak of the resulting distribution, since

$$p_h(m) = \int_b p_{h,c}(m,b) = \int_b p_c(b)p_{h|c}(m,b) = \int_b p_c(b)u\left(m - f(b)\right)$$

and by assumption $(0,0)$ is the maximum of u, so

$$u\left(f(b_1) - f(b)\right) < u\left(f(b_1) - f(b_1)\right) \;\forall\, b \neq b_1$$

which implies

$$\begin{aligned} \int_b p_c(b)u\left(f(b_1) - f(b)\right) &> u\left(f(b_1) - f(b_1)\right) \\ p_{h|c}(f(b_1), b_1) &> p_h(f(b_1)) \end{aligned}$$

since $f(b) \neq f(b_1)$ for some value of b and $E\left[u\left(f(b_1) - f(b)\right)\right]_c = \int_b p_c(b)u\left(f(b_1) - f(b)\right)$

Likewise, since the noise distribution p_n is independent of c, conditioning on c will increase this peak in the measurement model with respect to the noise when the hand is visible.

$$\begin{aligned} p_{x|c}(f(b_1), b_1) &> p_x(f(b_1)) \\ \beta p_{h|c}(f(b_1), b_1) + (1-\beta)p_n(f(b_1)) &> \beta p_h(f(b_1)) + (1-\beta)p_n(f(b_1)) \\ p_{h|c}(f(b_1), b_1) &> p_h(f(b_1)) \end{aligned}$$

Consequently, the signal to noise ratio is improved for the detection method we use. Similarly, the variance for the signal also decreases when conditioned on c. The increased peak of the signal relative to the noise also allows us to better determine when the hand is not visible for a configuration, since the absence of the peak can be more easily detected.

Estimation of the Hand's Position

Given our general probabilistic formulation, many estimation methods are applicable to find the ideal image positions $f(b)$ associated with body configurations b. We use histograms to nonparametrically estimate the distribution $p_{x|c}$ and find a maximum likelihood estimate for $f(b)$. Given the low dimensionality of the distribution and the large amount of training data, nonparametric estimation of the distribution is computationally feasible and results in a set of distributions that can be viewed as images to gain intuition and help debug the estimation process. A clear alternative would be parametric Gaussian distributions, since this estimation problem appears to be well-suited for the use of Gaussian distributions for the conditioned distributions $p_{h|c}$ and a Gaussian or uniform distribution for the global noise model p_n.

We first quantize the set of encountered kinematic configurations as represented by the orientations of the torso, upper arm and forearm with respect to the head orientation R_h.

$$\left[\begin{array}{ccc} R_h^T R_t & R_h^T R_u & R_h^T R_f \end{array} \right]$$

We use k-means [13] with the assumption that the set of encountered orientations occupies a small volume of the entire space of orientation values. For clustering we represent orientations using unit quaternions. The twelve dimensional feature vector consists of the four components of each of the three unit quaternions. The components of the quaternions function well with a Euclidean distance metric, since they represent rotations as a point on a 3-dimensional sphere and are easy to renormalize. Before performing k-means, we filter out the kinematic configurations for which the rate of change of the orientation of the forearm with respect to the head is below a threshold. We perform this prefiltering step because we intend to relate the k-means clusters to the output from our motion segmentation system, and image motion associated with the hand is more likely when the forearm is moving with respect to the camera's reference frame. In addition we filter out times when the rate of rotation of the head is above a threshold, since the background motion estimation performs

poorly during these large saccadic motions, as discussed in section 3.5.

For each of the k clusters, we initialize a 2D histogram, h_b, that we use to coarsely approximate $p_{x|c}$ for body configuration b. For each remaining image, i, there is an estimate of the hand's location, x_i, which corresponds with the location of maximal motion within the image. For the maximal motion location we originally used the position of the edge with the largest Mahalanobis distance from the background motion model, although more recently we have used the output of the interest point detector applied to the motion map as described in chapter 4. We partition these location estimates, x_i, into k sets where each set X_b contains the detected locations associated with members of cluster b. Finally, we coarsely approximate $p_{x|c}$ using the histograms h_b in the standard way.

$$h_b(v) = \frac{\sum_{X_b} \delta(g(x) - v)}{\sum_{X_b} 1}$$

where the function g transforms the image coordinates to the histogram's coordinates, which typically has a lower resolution than the image. We then smooth the resulting histogram.

For each body configuration b we now have a histogram h_b. We now find the bin of h_b with the maximum value, h_{max_b} and convert the bin index into an approximate image coordinate, h_{pos_b}. We also compute the variance of the histogram, h_{var_b}.

$$h_{pos_b} = \widetilde{g^{-1}}\left(\text{argmax}\left(h_b\right)\right)$$

$$h_{max_b} = \max\left(h_b\right)$$

Estimating the Joint Lengths

For each of the k body configurations we now have a hand position estimate h_{pos_b}. By itself, this nonlinear, data-driven, model can be used to predict the hand's location within an image given the configuration of the wearer's body as measured by the orientation sensors. However, it gives us no direct information about the placement of the sensors on the body, the orientations of the body parts in world coordinates, the positions of the forearm and upperarm with respect to the camera, or the lengths of the body parts. It is also unable to provide hand position predictions outside of the body configurations it has witnessed, and does not provide a clear method for interpolating the hand prediction given

a configuration in between the k means that coarsely discretize the the configurations of the body. In contrast, once they are fit to this data, the linear and nonlinear kinematic models do provide these benefits.

We first throw out estimates for which $h_{max_b} < p_{thresh}$ or $h_{var_b} > var_{thresh}$, in order to eliminate generally poor hand position estimates, as well as position estimates from images for body configurations for which the hand is not visible. With the remaining estimates, we generate pairs of body configurations and hand positions. For the body configuration, we normalize the mean vector of the associated cluster to be three unit quaternions. Originally, we would convert these three unit quaternions into rotation matrices $R_b = \begin{bmatrix} R_c^T R_t & R_c^T R_u & R_c^T R_f \end{bmatrix}$, so that they could be used with the kinematic estimation techniques of this chapter. More recently, we have used the body configuration closest to the mean in order to avoid the chance of the mean not matching a plausible body configuration. At this point we have sometimes used these pairs, (R_b, h_{pos_b}), to fit the 2D, non-projective, linear least squares model, which is very efficient. The resulting residuals and the 3D position estimates from this model could then be used to throw out significant outliers that correspond with very poor hand estimates prior to fitting the nonlinear model. More recently, we have directly fit the nonlinear model to the training pairs without this filtering step, which seems to lead to better solutions at the cost of more computation.

Summary

In summary, the system uses three tricks to discover the hand with very modest constraints on its appearance. This same approach should work if the wearer is wearing a glove or has an artificial hand. The first trick takes advantage of the fast 3D motion of the hand, and its proximity to the camera, which lead to the hand having the tendency to create the fastest foreground image motion. Second, the probabilistic model we described is able to amplify a hand detector signal by conditioning it on the body's configuration. In this case, conditioning on the body's configuration amplifies the weak hand signal provided by our maximal motion detection system. Finally, the form of the kinematic model, and robust nonlinear search for optimal parameters, finds hand hypotheses that are consistent with a kinematic model and throws out hand hypotheses that are not. Consequently, this last step is able to handle many bad hypotheses, such as the bad hypotheses giving a hand position in the image when the hand is not visible for the associated body configuration.

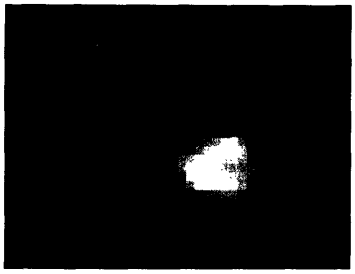


Figure 3-23: This image shows a smoothed histogram of maximal motion locations for all of the frames from the 11500 images of dataset 1. This histogram is not conditioned on the configuration of the body. With respect to our model, this histogram is a nonparametric empirical approximation of the density p_x. Notice that the lower-right third of the image has larger probability due to hand and arm motion, and the upper-left of the image has a strong response due to the brim of the hat, which was stationary with respect to the camera and hence moving with respect to the background. The approximate peak of the motion may correspond with preferred hand locations during manipulation and viewing by the wearer.

All three tricks work together to take a very weak hand detector based on maximal motion, and output useful 2D hand position predictions in the image, 3D hand positions relative to several different coordinate systems, and an explicit kinematic model with joint lengths and orientations relative to the sensors. One other point worth noting, is that using k-means with a fixed value of k to quantize the body configurations constrains the computational complexity of the nonlinear search by always using the same number of training examples, k.

3.9 Putting it All Together

We now combine all of the methods of this chapter to autonomously determine the assignment of the sensors to the wearer's body parts, find the relationship between the camera and the head-mounted orientation sensor, discover the position of the hand, and finally learn a kinematic model that includes the lengths of the major axes of the upper-arm and forearm, the relationship between the camera and the kinematic model, and the orientations of the body parts with respect to the sensors.

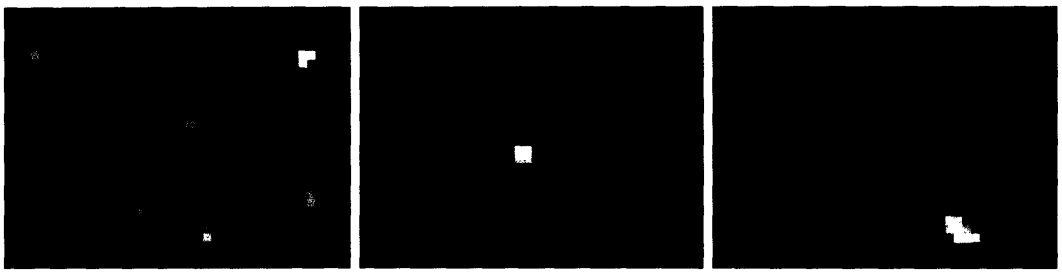


Figure 3-24: These three images show smoothed histograms of maximal motion locations conditioned on the configuration of the body. With respect to our model, these histograms are nonparametric empirical approximations of the conditional density $p_{x|c}(x, b)$ for three different quantized values of the body configuration, b. The left histogram corresponds with a body configuration for which the hand is not visible, which leads to widely distributed noise and a histogram that approximates p_n from our model. The center histogram corresponds with a body configuration for which the hand is positioned in the center of the camera's view. The right histogram corresponds with a body configuration for which the hand is in the lower right portion of the image. Notice that the signal to noise ratio is high for the right two histograms, which matches well with our model.

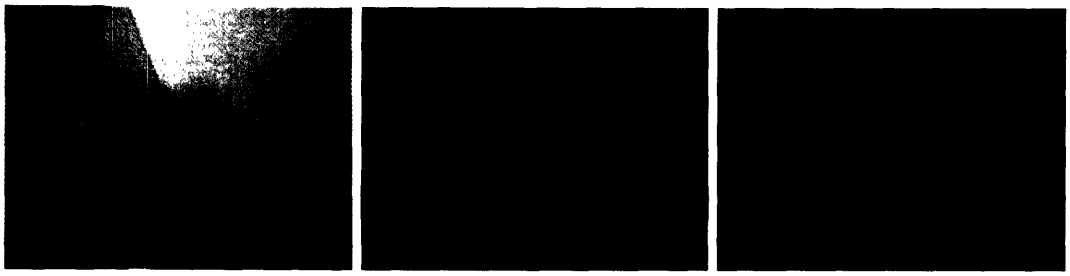


Figure 3-25: The left image shows the predicted hand location from the learned kinematic model. The center and right images show visualizations of the learned kinematic model from the camera's view point and the world coordinate system, respectively. These particular images illustrate a major advantage of combining wearable kinematic sensing with vision, since the hand would be difficult to detect visually in this common low-light situation. A trivial amount of computation produces the estimate of the hand's location using the measurements from the orientation sensor. Given this position estimate, visual processing related to the hand can be restricted to a small area of the image. The two modalities complement one another, with the kinematic processing usually requiring far less bandwidth, storage, and computation.

We successfully tested these methods on all three data sets. Within this section we present results from a kinematic model learned from data set 1. For this test, we used $k = 100$ body configuration clusters. Examples of the 100 corresponding histograms that serve as estimates for $p_{x|c}$ are shown in figure 3-24. Figure 3-23 shows the unconditioned global histogram p_x. As these examples indicate, for this data set the histograms match well with our probabilistic model.

3.10 Wrist Rotation

The kinematic model does not include an estimate of the twisting of the wrist around the forearm's axis. For manipulation tasks, however, this wrist rotation can be very informative. In this section, we describe a method that estimates wrist orientation based on the orientation sensor data and the learned kinematic model.

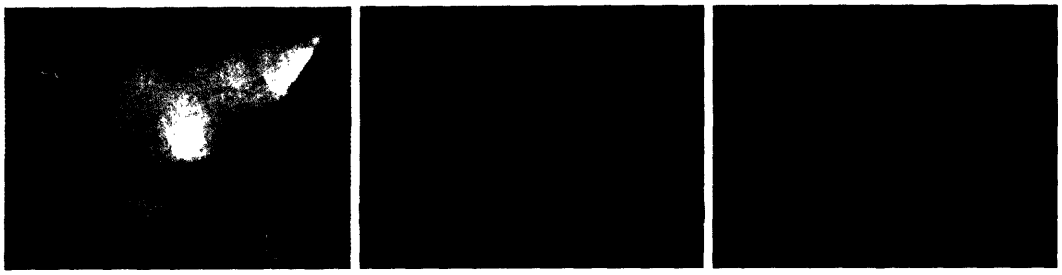


Figure 3-26: The left image shows the predicted hand location from the learned kinematic model. The center and right images show visualizations of the learned kinematic model from the camera's view point and the world coordinate system, respectively.

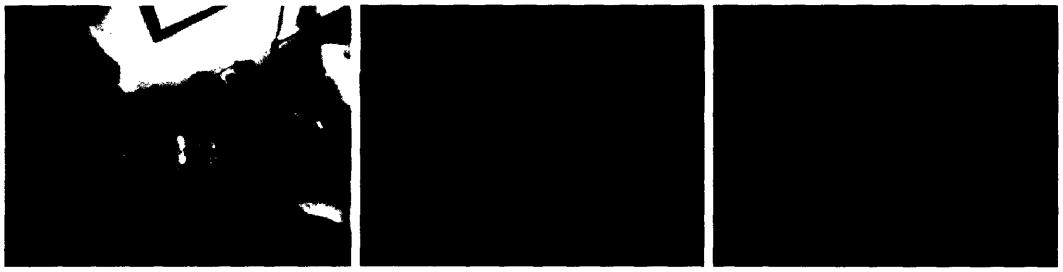


Figure 3-27: The left image shows the predicted hand location from the learned kinematic model. The center and right images show visualizations of the learned kinematic model from the camera's view point and the world coordinate system, respectively.

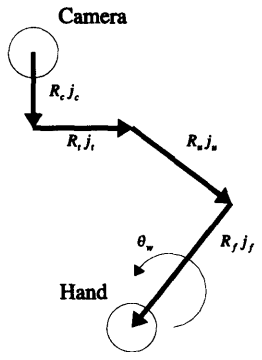


Figure 3-28: We can use the learned kinematic model to estimate the orientation of the wrist, θ_w.

Estimating Wrist Orientation

We can find the wrist's orientation using the absolute orientation information from the forearm along with the major axis of the forearm estimated by our learned kinematic model. One complexity that we must be careful to address is the influence of global rotations of the body on the wrist rotation estimation. For example, if a person were spinning in place with his arm by his side, the global spinning would be interpreted as wrist rotation with respect to the absolute orientation measurements from the forearm, R_f. In order to avoid this, we find the relative orientation of the forearm, R_{uf}, with respect to the absolute orientation of the upperarm, R_u, prior to finding the wrist orientation.

$$R_{uf} = R_u^T R_f$$

This works because the forearm and upper arm are only connected by a 1DOF rotational joint at the elbow followed by a rotational joint around the axis of the forearm. The major risk in doing this is the tendency of the upperarm mounted orientation sensor to be less stably mounted than the wrist sensor with respect to twisting around the arm. Despite this risk, in practice we've found the results to be of good quality. The learned kinematic model can be used to estimate the rotation around the elbow, R_e, and the difference between it and the relative rotation of the forearm must be approximately equal to the wrist rotation we desire, R_w. So,

$$\begin{aligned} R_w R_e &= R_{uf} \\ R_w &= R_{uf} R_e^T \end{aligned}$$

where the rotation around the elbow, R_e , has the following axis and angle of rotation, which is computed using the transformed forearm vector j_f from the kinematic model.

$$\begin{aligned} axis_{R_e} &= j_f \times R_{uf} j_f \\ angle_{R_e} &= \arccos\left(j_f, j_f \cdot R_{uf} j_f\right) \end{aligned}$$

We use the angle from the axis and angle form of R_w for our unnormalized wrist orientation, θ_{uw}. Notice that the axis of R_w simply points in the direction of j_f.

Normalizing Wrist Orientation

We wish to be able to compare wrist orientations across sessions with distinct sensor placement and different wearers, so we now find a normalized wrist orientation θ_w using the statistics of the observed angle, θ_{uw}. The orientation range of the wrist is approximately π. Two simple methods for normalizing the wrist orientations, would be to find the mean wrist orientation, or find the orientation range by looking for minimums and maximums. A risk of using the mean orientation, is that the specific activity of the wearer will unduly bias the measurement and require longer periods of averaging. Two risks of using the minimums and maximums are that there may be noise in the measurements that would distort the range, or the wearer might not rotate the wrist to the extremes. A histogram of the orientations from data set 1 samples, shown in figure 3-29, suggests an alternative method of normalizing the wrist orientation and lends credence to our concerns about using the mean or minimum and maximum values. The distribution is very peaked and falls off much faster than a Gaussian, which indicates that there is a special orientation at which the wrist spends most of its time. We also see the highly tapered tails of the distribution as well as some rare, but extreme, noise that would make estimations based on the minimum and maximum values poor, if not useless. We estimate this rest point angle non-parametrically by finding the maximum of a histogram of θ_{uw} fmod 2π, which has the advantage of avoiding any problems we might otherwise encounter due to orientation wrap-around.

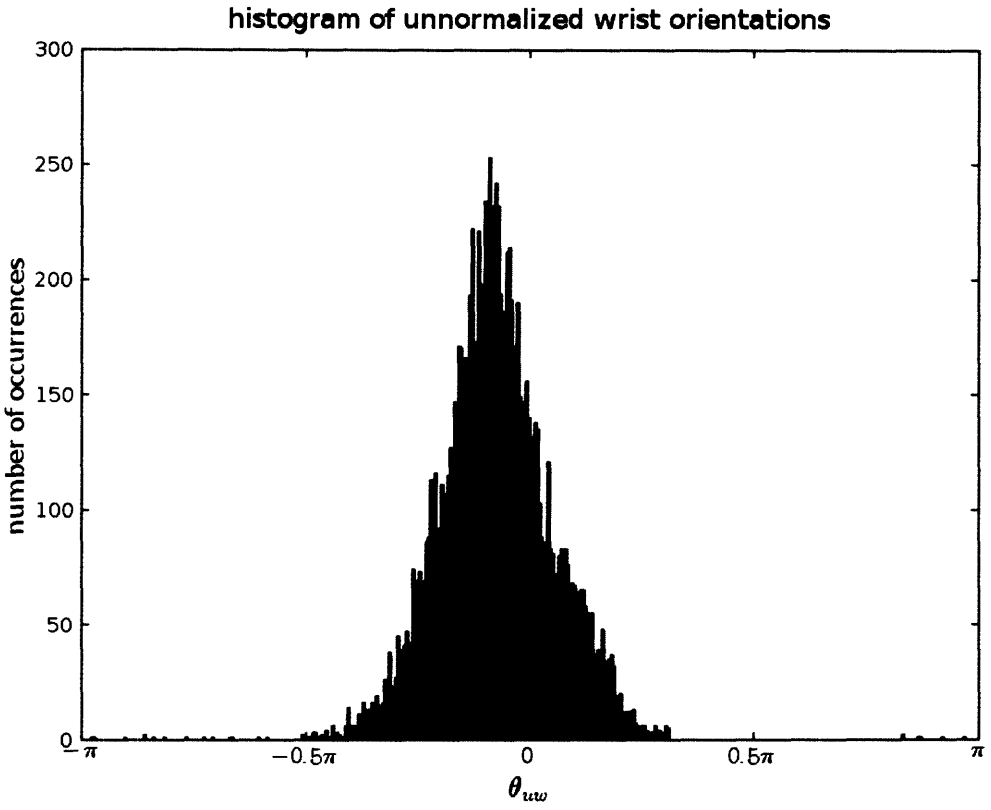


Figure 3-29: This figure shows a histogram of the unnormalized wrist orientations, θ_{uw} estimated from dataset 1. Notice that the range for wrist motion is approximately π radians, as one would expect from inspection of his own wrist movement. Also, notice that the distribution falls off much faster than a Gaussian, which indicates that during the activities the wrist had a natural orientation to which it would return. This also matches our intuition due to the comfort of various wrist angles and the orientation of the wrist when it hangs relaxed by one's side. Finally, one should notice that there is noise that occasionally appears and sits well outside of the true wrist orientation. We have not investigated the source of this noise.

Chapter 4

Attention & Segmentation

Duo's perceptual systems efficiently find salient segments of kinematic and visual stimuli. For the kinematic perceptual system, a kinematic segmentation method uses the learned kinematic model to find moments in time that are likely to border significant hand activity and correspond with important 3D positions of the hand. For the visual system, the learned kinematic model and a visual interest point operator select salient locations and sizes at which to find visual segments. This staged, attention-based filtering used by the visual system is important for practical applications, since our method of producing visual segments runs at well-below frame rate due to its large computational requirements.

Within this chapter, we first present our method for kinematic segmentation. Next, we describe the attention mechanisms used for the visual system, including a new visual interest point operator with associated shape descriptors. We then conclude by describing our image segmentation algorithm, which produces local, approximate, parts-based segments.

4.1 Kinematic Segmentation

The kinematic segmentation algorithm for hand activity splits kinematic activity at local minima of multi-scale, low-pass filtered estimates of the hand's velocity. Hand velocity serves as a useful measure of hand activity. The hand must make physical contact with items in the world in order to influence them, so the body propels the hand from position to position in order to physically interact with items of interest. Most of this propulsion is provided by the arm, which moves the hand very rapidly between positions with a unimodal velocity profile that tends to peak near the center of the reach and scale linearly with

distance [29]. Consequently, acts of manipulation can usually be broken into three parts: propelling the hand to the appropriate position and matching the velocity of the item of interest, interacting with the item, and propelling the hand back to the body or another item in the world. In real life manipulations, the transitions between these parts of the hand's activity tend to be blurred, but the velocity before and after interacting with an item does momentarily diminish relative to the high velocities used to propel the hand between positions. As always, there are exceptions, such as with ball contact in volleyball, which may result in a very small velocity change relative to the hand's motion.

If we can find these points of diminished velocity, we can better interpret captured video and kinematic data. For example, for machine augmented browsing and annotation as we describe in chapter 6, we can summarize video based on these special moments in time, see figure 4-1. Similarly, as we will explore in chapter 5, the hand positions associated with these special moments in time have strong structure, which we can discover autonomously.

Within the literature, researchers have often used multi-scale local minima or zero-crossings to segment motion capture data [27, 7, 21, 50]. However, these methods typically use joint angles, as opposed to the estimated motion of the hand. As discussed above, hand velocity is a well-founded method for segmenting the natural manipulation activities in which we are interested. Many motion capture researchers process free-form motions, such as dancing and martial arts, as opposed to natural manipulation activities. Furthermore, motion capture data for people performing everyday tasks within unaltered home environments is rare, if not unique, due to the challenges of capturing this type of activity. Finally, combining motion capture data with first person video is uncommon.

4.1.1 Detection Overview

We wish to detect these transition points using our kinematic model. Local minima are an appropriate feature for detecting these transition points due to three reasons: the typically smooth unimodal hand velocity of reaching, the tendency for the hand velocity to scale linearly with reaching distance, and our lack of knowledge about the two frames of reference between which the hand is transitioning. Likewise, velocity profiles for reaching tasks over different distances can be coarsely modeled as being scaled versions of one another, so a multi-scale filter bank with scaled unimodal filters is appropriate for finding activity across different distances. In general, long distance motions tend to correspond with more

Figure 4-1: This figure shows an automatically summarized 120 frame sequence of the wearer getting a drink out the refrigerator. Frames were selected that corresponded with local minima detected by the kinematic segmentation system, which in this case used the hand-tuned kinematic model. The automatically selected synopsis frames correspond well to reaching for the refrigerator door, opening the refrigerator door, reaching into the refrigerator to grab a drink, reaching for the refrigerator door, and closing the refrigerator door.

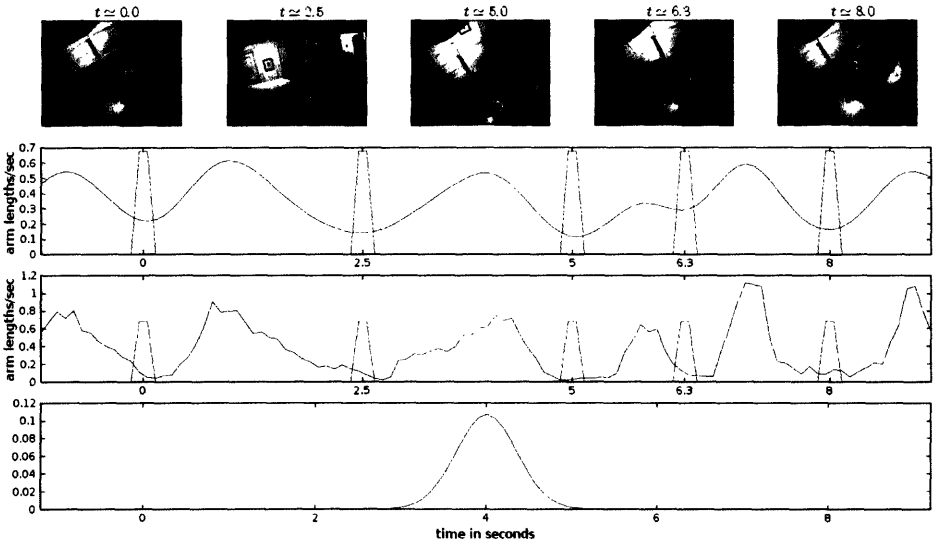


Figure 4-2: This figure shows the results of the kinematic segmentation algorithm applied to a sequence with the wearer drinking from a cup. The top row shows the images that are closest in time to the kinematic segmentation times. The second row down shows the smoothed hand velocity in units of arm lengths per second. The red spikes indicate local minima detected by the kinematic segmentation system. The third row down shows the raw hand velocity estimates based on the hand position estimates provided by the learned kinematic model over time. In general these estimates are noisy. The bottom row shows the Gaussian smoothing filtered used for these segmentations.

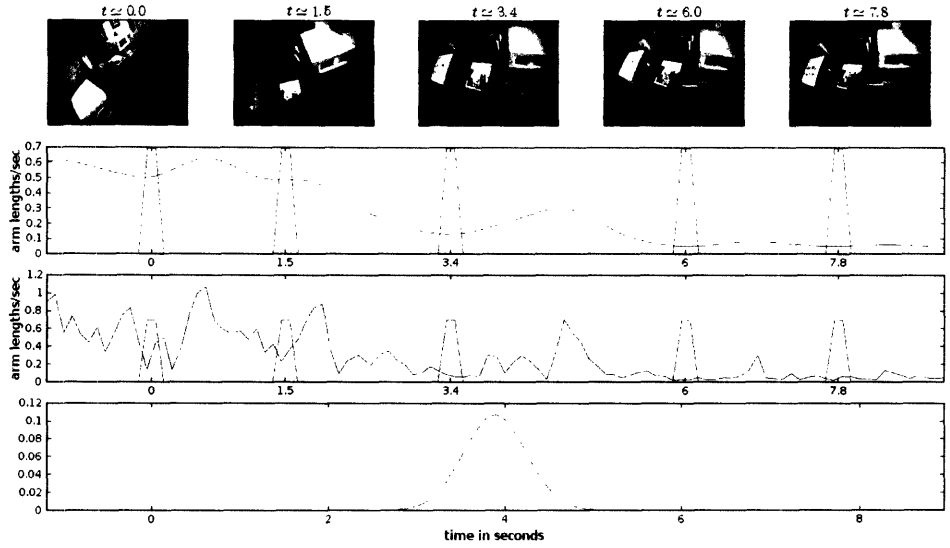


Figure 4-3: This figure shows an example of the kinematic segmentation algorithm being run on a sequence in which the wearer orients towards a writing pad on a table, reaches and grabs the notepad, manipulates the top page, and begins to write on the notepad.

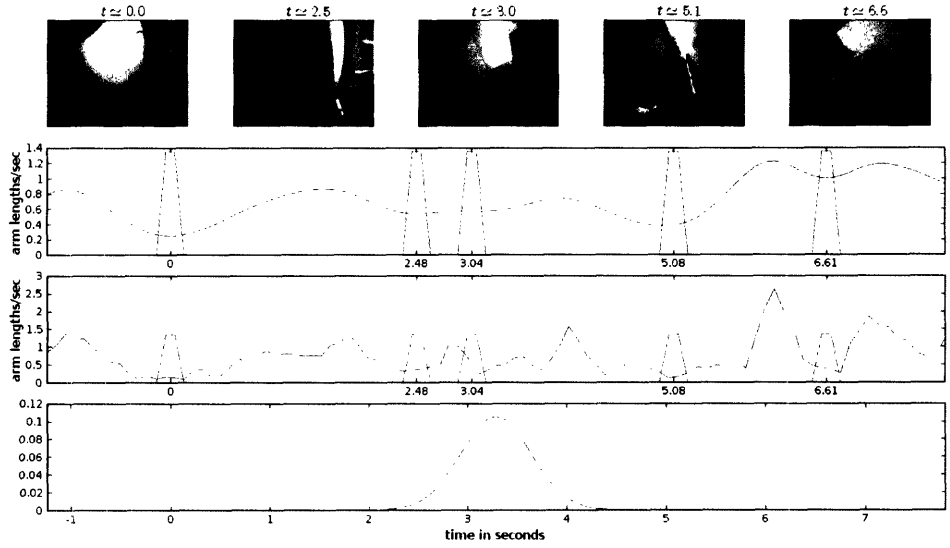


Figure 4-4: This figure shows another sequence of kinematic segmentations using the same form as figure. The sequence shows the wearer reaching for the door knob, opening the door, moving through the doorway, closing the door, and bringing his hand back to his side.

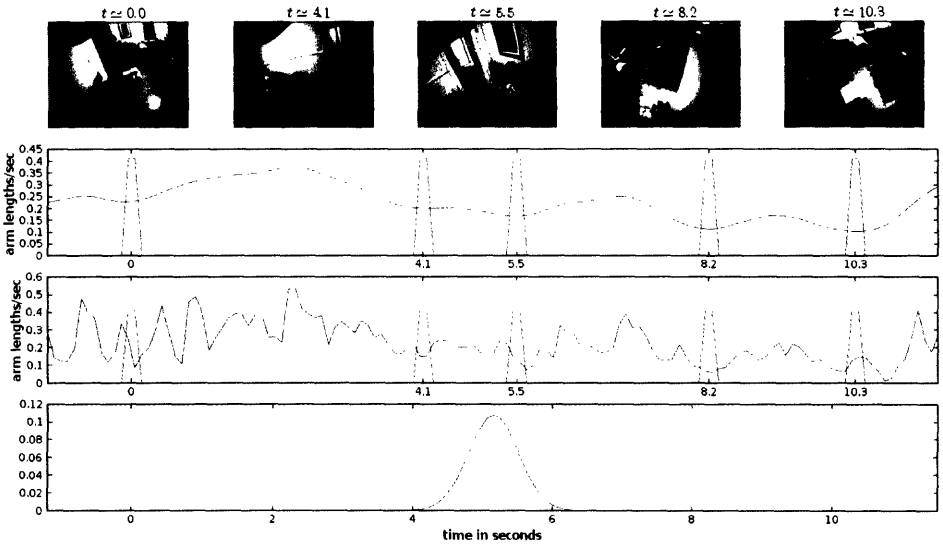


Figure 4-5: This figure shows the results of kinematic segmentation while the wearer walks around a room. The graphs are as described in figure. The segmentations occur because we are using the hand's velocity relative to the position of the torso and the orientation of the world, so that spinning in place results in estimated hand velocities. If we are interested in manipulation events we may want to filter out these segmentations by detecting walking. For other tasks related to navigation these segmentations may be useful, though we do not explore this possibility. We measure velocity with respect to the world's orientation in order to better account for hand velocity due to torso rotation, such as when twisting to open a door.

significant transitions. Figures 4-2, 4-3, 4-4, and 4-5 show examples of this signal processing performed at a single scale that we have used in this thesis.

We now need to select a frame of reference from which to measure the velocity of the hand. Ideally, we would know the frames of reference for the start and the end of the hand's journey and in some way interpolate our velocity measures between those two frames. We can safely assume that most of the manipulation activities in which we are interested will either be performed relative to a body related frame of reference or with respect to the world frame, since most objects are stationary with respect to the world prior to being grabbed for manipulation. Given that our kinematic model is based solely on absolute orientation measurements, it does not give us a direct way of measuring the hand's velocity with respect to the world frame. We can, however, approximate this velocity since the position of the torso tends to move slowly within the world relative to the hand's rapid motion. In many manipulation tasks the torso tends to be a stable frame of reference from which the arms and head move, observe, and manipulate the world. Using our kinematic models, we can measure the velocity of the hand with respect to a position on the torso using the world orientations of the body parts to find a good approximation of the world velocity of the hand. In contrast to the world frame, we can directly estimate the velocity of the hand with respect to the parts of the body. The torso is the most useful frame of reference for detecting these transitions, since many manipulation tasks occur in a nearly constant frame of reference with respect to the torso, which we will discuss further in section. For this frame of reference we normalize the rotation matrices of the kinematic model by the absolute world orientation of the torso.

$$\begin{bmatrix} R_t^T R_t & R_t^T R_u & R_t^T R_f \end{bmatrix}$$

The torso's orientation becomes I, so any position on the torso will result in the same hand velocity. The head's frame of reference could be useful for segmenting some activities that involve moving objects to the head, such as objects for eating, drinking, and listening. However, head rotations are common and outside of these can generally lead to irrelevant local minima in the hand velocity.

Fortunately, for many hand activities, the hand's velocity is high enough that multi-scale local minima with respect to the local torso frame are similar to the local minima measured with respect to the world torso frame. For these kinematic segmentation results, we use the

world torso frame.

4.1.2 The Specifics

With the hand-tuned model, we have used hand velocity estimates with respect to a centered position at the base of the torso using absolute world orientations. The 3D position of the hand, x, at time step i is a function of the orientations of the torso, upper arm and forearm (R_t, R_u, and R_f) at time step i with respect to the world, and the vectors that represent the torso length, upper-arm length, and forearm length (j_t, j_u, j_f).

$$x_i = \begin{bmatrix} R_{ti} & R_{ui} & R_{fi} \end{bmatrix} \begin{bmatrix} j_t \\ j_u \\ j_f \end{bmatrix}$$

The automatically adapted kinematic model only provides a length of the torso from the base of the neck to the shoulder, since it does not estimate a length for the major axis of the torso. We can either use this value for j_t and estimate the velocity of the hand with respect to the position at the base of the neck, or we can estimate the velocity of the hand with respect to the base of the torso by approximating the length of the torso's major axis, and its orientation with respect to the torso orientation sensor. For example, we could potentially estimate the torso's length by using the relationships between forearm length, upperarm length, and torso length from population statistics. Possibly due to a failing sensor, the torso dimension estimate from the learned kinematic model is poor. For this work, we simply use the estimated velocity with respect to the position of the base of the neck on the learned kinematic model and the orientations with respect to the world.

Given these hand position estimates, we compute the first difference of the positions and the associated time stamps, t, to obtain a linear estimate of the hand velocity, v, at the center time, $(t_{i+1} + t_i)/2$.

$$v_i = \frac{x_{i+1} - x_i}{t_{i+1} - t_i}$$

These estimated samples of the hand velocity are now measured and calculated at around 100Hz, although some of our results from older data sets use samples at 10 to 15Hz. Even with the newest system, the actual frequency varies based on a number of factors, not the least of which is the computational load for the system. Since the system uses Linux without

a hard real-time task scheduler the frequency and phase of the samples can show significant jitter. 100Hz is a relatively high frequency with respect to the rate of the significant units of kinematic motion in everyday activity, and is sufficient for segmenting the majority of hand activity.

We now filter these velocity estimates with low-pass filters, f_j, at a series of scales j. The specifics of the smoothing, low-pass filters do not seem to be too important for the tests we have performed, although given the common velocity profiles of the hand, smooth unimodal filters are sensible from a matched filter perspective. For very efficient processing, a simple block filter of various lengths can be sufficient, which only requires an addition and subtraction for each new sample. Given more processing power, multi-scale Gaussians can be used, possibly with downsampling. For this section we use multi-scale Gaussians to filter the velocities. Gaussians have the advantage of being used in a large body of work on the scale-space analysis of signals. Other types of wavelet type processing would most likely be effective too. We normalize the filter elsewhere, so we drop the multiplicative Gaussian normalization to give us the following set of filters:

$$f_j(t) = e^{-\frac{t^2}{2\sigma_j^2}}$$

We perform the following computation to compute the smoothed velocity estimates, s, at each scale, j. The equation is essentially a correlation (or convolution assuming symmetry of the filter) between the filters and the velocity signal. To compensate for jitter and frequency changes in the velocity samples, the filter is evaluated at appropriately shifted times and normalized to sum to one. Potentially superior compensation methods exist, for example interpolating and resampling the velocity signal at a uniform frequency and then performing standard convolution, but this method for computing s works sufficiently well.

$$s_j(t) = \frac{\sum_i f_j\left(\frac{t_{i+1}+t_i}{2} - t\right) v_i}{\sum_i f_j\left(\frac{t_{i+1}+t_i}{2} - t\right)}$$

Finally, after computing $s_j$at a sequence of times, t_k, we find local minima in these smoothed hand velocity signals. A local minima is detected at $t_k$if

$$s(t_k) > s_j(t_{k-1}) \; and \; s_j(t_k) > s_j(t_{k+1})$$

These detected local minima, m_n, now serve as hypothesized breaks between significant units of hand motion at different scales of time and space.

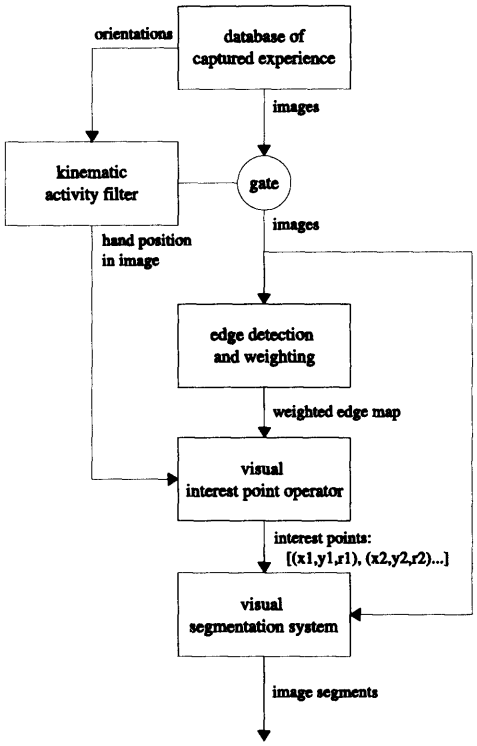


Figure 4-6: This diagram depicts the attention and segmentation system used to find image segments of interesting hand activity.

4.1.3 Results

We show qualitative results of this segmentation method in figures 4-2, 4-3, 4-4, and 4-5. Furthermore, in chapter 5, we show that the hand positions associated with these special moments in time have strong structure, which we can discover autonomously.

4.2 Visual System Overview

As shown in figure 4-6, for visual processing the learned kinematic model is used to find times and image locations worthy of further processing. From the filtered stream of kinematically interesting images, weighted edge maps are computed with weights that indicate how important an edge is to the particular situation. After this, a visual interest point operator takes the weighted edge map as input and finds salient image positions with associated scales, represented as radii, $[(x_1, y_1, r_1), (x_2, y_2, r_2)...]$. These interest points can then be further filtered based on their relationship to the kinematically predicted hand location.

Finally, given the set of remaining interest points, the visual segmentation system finds a coherent visual segment for each position and scale tuple.

Our visual segmentation system, which we describe later in this chapter, performs computationally intensive image segmentations at a rate of around 5 to 10Hz on a 3GHz AMD machine, which necessitates significant filtering of the vast number of possible positions, scales, and times by the attention system in order to allow for practical computation times. For example, to collect the segments that we will analyze in chapter 5, Duo's visual attention system uses the following measurements to efficiently direct Duo's computational resources toward the wearer's hand and the objects the hand is manipulating:

1. Select images that have high kinematically estimated visibility of the hand.
2. Select images that have high kinematically estimated motion of the hand.
3. Compute edge maps weighted by the edge's amount of foreground motion, where the edge point's foreground motion is the Mahalonbis distance between its motion and the global 2D affine motion model, as described in chapter 3.
4. Apply the visual interest point operator we describe in the next section to these motion weighted edge maps in order to produce a list of interest points.
5. Select the resulting interest points that are near the kinematically estimated position of the hand.
6. Use the visual log-polar image segmentation method we describe later in this chapter to find an image segment for each of the remaining interest points.

By kinematically selecting images in which the hand is highly visible, and moving rapidly, the visual attention system selects moments in time at which motion based interest points are likely to do a good job of selecting interest points associated with the hand and any object it might be manipulating. Consequently, as we show in chapter 5, the resulting visual segments tend to be of the hand, and objects the hand is manipulating.

4.3 From Visual Edges to Points of Interest

Our motion processing system only provides motion information at strong edges within the image. More generally, important image information is often concentrated around strong

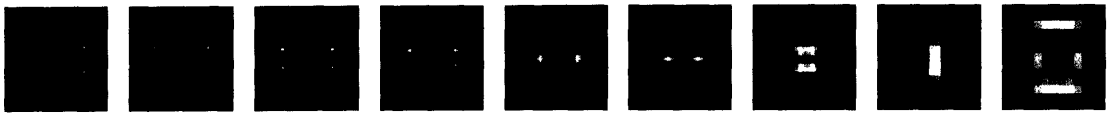


Figure 4-7: This figure shows an unfiltered series of interest point salience maps produced by this algorithm when given a white rectangle as input. The scale associated with the maps increases from left to right. Strong responses in the maps correspond with corners, parallel lines, and the ends of the rectangle at various scales. The output of the algorithm has similarities to the output from classic image processing techniques such as the distance transform and medial axis transform, which can be viewed in terms of wave fronts that start at the edges and propagate away from the edges, intersecting one another at significant locations.

edges. We would like to combine the information distributed across the edges in a way that helps the attention system. Multi-scale methods for finding interest points within images with distinct regions have become popular within the machine vision literature. Interest point detectors have been used as a precursor to a variety of recent object recognition systems [36, 37, 38, 64, 24]. David Lowe's interest point methods, which are especially popular, are supported by long developed formalizations of continuous scale-space representations of images [35]. They are particularly effective when the shape of the regions of interest are roughly circular and the interiors of the regions of interest have an average pixel value that is significantly different from the exterior's average pixel value. Due to these characteristics, these methods have been described as blob detectors. These methods are not well-suited to regions of interest that are primarily defined by varying contrast strongly localized at a region's border, nor are they appropriate for weighted edge maps as provided by our motion processing system. Within this section we describe our interest point method that serves as a complementary, edge-based approach to these region based interest point methods.

The algorithm we present is computationally efficient and suitable for real-time processing. An example of the raw output maps is show in figure 4-7. In addition to finding interest points in an image, we present a computationally efficient variation of this algorithm that produces shape descriptors for each interest point that characterize an interest point as corresponding to a corner, parallel lines, or a circle and provides an orientation for

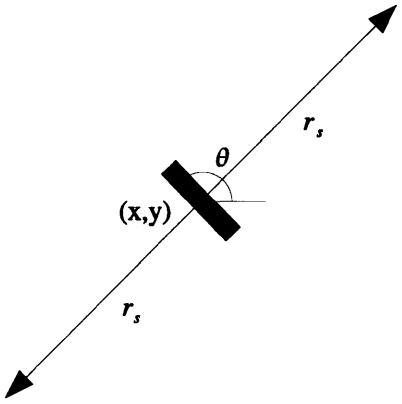


Figure 4-8: This figure depicts the approximate locations of the two votes at scale s cast by an edge with orientation θ and position (x, y).

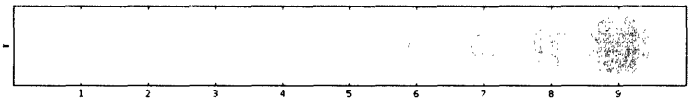


Figure 4-9: This figure shows the radius, r_s, as a function of the scale index, s, for parameters $r_{min} = 1.1$, $r_{max} = 20.0$, and $c = 9$.

this shape, if it is not too circular. A single, non-circular shape feature defines a transform in position, orientation, and scale relative to a corresponding shape feature, which is useful for finding alignments between sets of feature points. These features serve as a promising compromise between highly complex point descriptors [3, 43, 4, 15, 5], and overly generic point descriptors, such as classic corner detection methods [17, 12].

4.3.1 Edge Based Interest Points

The input to the interest point detector consists of a set of weighted edges, e_i, where each edge i consists of a weight, w_i, an image location, (x_i, y_i), and an angle, θ_i. In a manner similar to a Hough transform for circles [17], each edge votes on locations in scale-space that correspond with the centers of the coarse circular regions the edge borders.

For each edge, we add two weighted votes to the appropriate bin locations (b_x, b_y) at each integer scale s. As depicted in figure 4-8, within the original image coordinates the two votes are approximately at a distance r_s from the edge's location and are located in positions orthogonal to the edge's length. For this section, we assume that the angle θ_i describes the direction of the edge's length and that θ_i is in the range $[0, \pi)$, so that no distinction is made between the two sides of the edge.[1]We define r_s to be scale-invariant with respect to the integer scale index s with r_{s+1} a constant multiple of r_s. Given the constants h and g, r_s is defined as follows.

$$r_s = \exp\left(h(s-1) + g\right)$$

$$\frac{r_{s+1}}{r_s} = \frac{\exp\left(hs + g\right)}{\exp\left(h(s-1) + g\right)} = \exp(h)$$

We choose h and g so that r_s is between r_{max} and r_{min} inclusive, and s is an integer that ranges from 1 to c inclusive. The exponent of constant growth is determined by r_{max}, r_{min}, and the number of scales c, see figure 4-9.

$$r_{min} = r_1 = \exp\left(g\right)$$
$$r_{max} = r_c = \exp\left(h(c-1) + g\right)$$

which implies

$$g = \log(r_{min})$$

$$h = \frac{\log(r_{max}) - \log(r_{min})}{c-1}$$

which can be easily verified.

For each scale s there is a 2D histogram that accumulates votes for interest points. The discretization of these histograms is determined by the integer bin length, l_s, which scales linearly with the scalar parameter β. Higher β values result in larger bins and lower resolution histograms. For convenience, l_s can be specified in terms of the approximate length of the radial dimension of the log-polar discretization

$$l_s = \text{ceil}\left(\beta\left(r_{s+0.5} - r_{s-0.5}\right)\right)$$

[1]For some applications, such as finding bright regions, this algorithm can be adapted to use a full range of angles.

or the arc-length of this discretization,

$$l_s = \text{ceil}\left(\beta \frac{2\pi}{n} r_s\right)$$

Vote accumulation uses discretized angles a.

$$a(\theta) = \frac{2\pi}{n}\text{round}\left(\frac{n}{2\pi}\left(\theta + \frac{\pi}{2}\right)\right)$$

In order to calculate the bin indices, (b_x, b_y), for the 2D histogram at scale s, we first quantize the displacement vector, d,

$$d_x(\theta, s) = \text{round}\left(\frac{r_s}{l_s}\cos(a(\theta))\right)$$
$$d_y(\theta, s) = \text{round}\left(\frac{r_s}{l_s}\sin(a(\theta))\right)$$

which results in the displacement vector being constant across all edges with a given angle regardless of position, see figures 4-10 and 4-11. Given this quantization, a look-up table of displacement vectors can be precomputed for efficiency. We then add this displacement vector to the scaled and quantized position of the edge.

$$b_x(x, \theta, s) = \text{round}\left(\frac{x}{l_s}\right) + d_x(\theta, s)$$
$$b_y(y, \theta, s) = \text{round}\left(\frac{y}{l_s}\right) + d_y(\theta, s) = b_x(y, \theta - \tfrac{\pi}{2}, s)$$

Now that we have the bin indices, we can write the following equation for the resulting interest point salience map, map, for scale s using delta functions, δ

$$\delta(x, y) = \begin{array}{l} 1 \text{ if } (x = 0) \wedge (y = 0) \\ 0 \text{ otherwise} \end{array}$$

$$\begin{aligned} map_s(u, v) = \sum_i w_i(\; & \delta(u - b_x(x_i, \theta_i, s), v - b_y(y_i, \theta_i, s)) + \\ & \delta(u - b_x(x_i, \theta_i + \pi, s), v - b_y(y_i, \theta_i + \pi, s))) \end{aligned}$$

Finally, in order to soften the effects of our block discretization, we low-pass filter map_s with a separable, clipped, FIR Gaussian, for example a $3x3$, $5x5$, or $7x7$ filter.

$$smoothedmap_s = G \star map_s$$

which is equivalent to giving each edge a Gaussian vote distribution with

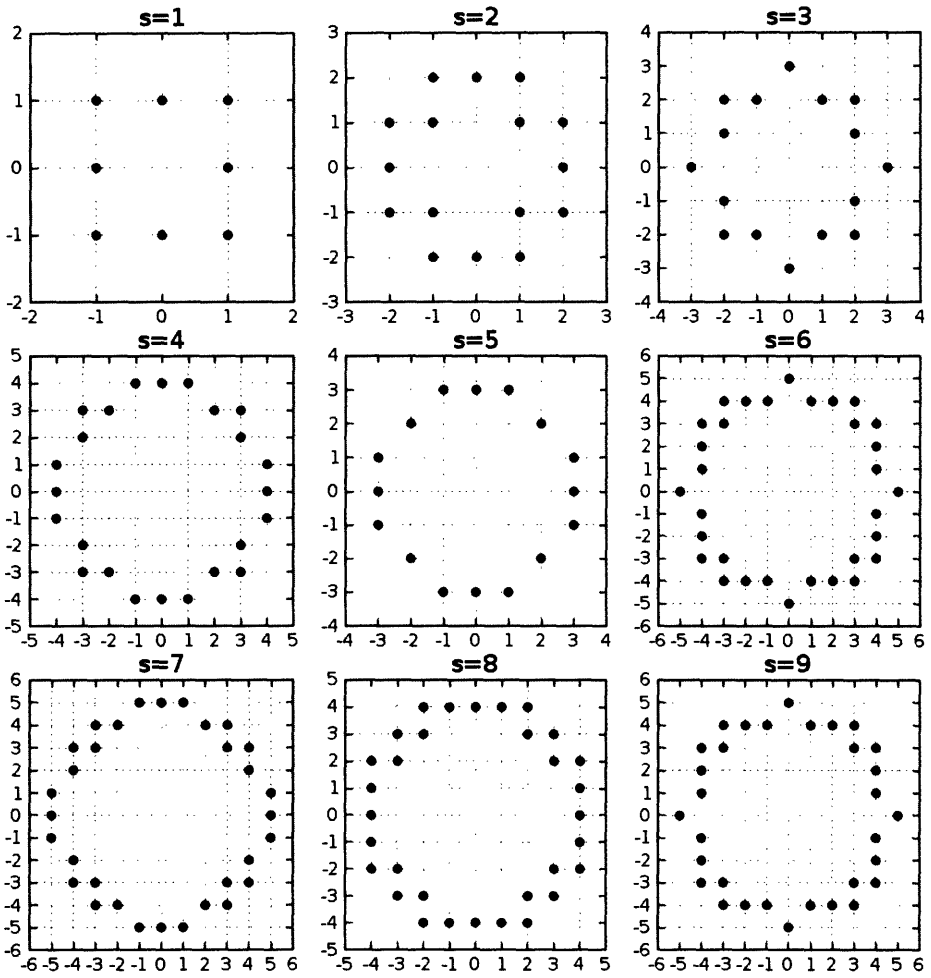


Figure 4-10: This figure shows the histogram bins that can receive a vote from an edge at position $(0,0)$ for some angle θ. Each of the nine diagrams represents the 2D histogram for a different scale s. These graphs were generated by placing a dot on a position if $(b_x(0,\theta,s), b_y(0,\theta,s))$ equals that position for some value of θ, notice that $(b_x(0,\theta,s), b_y(0,\theta,s)) = (d_x(\theta,s), d_y(\theta,s))$. The specific quantization of the circle varies by scale. For this visualization $r_{min} = 1.1$, $r_{max} = 20.0$, $c = 9$, $\beta = 0.5$, and $n = 32$.

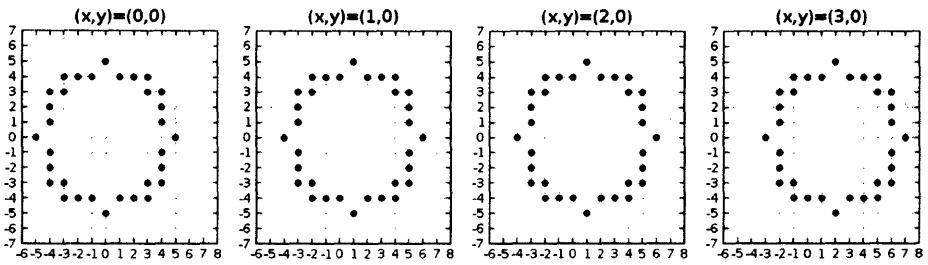


Figure 4-11: This figure shows how the 2D histogram at scale, $s = 6$, from figure 4-10 translates as the position of the voting edges translates along x. These graphs were generated by placing a dot on a position if $(b_x(x, \theta, 6), b_y(0, \theta, 6))$ equals that position for some value of θ. Due to the quantization of (d_x, d_y), the shape of the circle remains constant with translation. For this scale, the edges need to translate 2 units in order for the histogram to translate 1 unit. For this visualization $r_{min} = 1.1$, $r_{max} = 20.0$, $c = 9$, $\beta = 0.5$, and $n = 32$.

$$smoothedmap_s(u,v) = \sum_i w_i(\; G(b_x(x_i,\theta_i,s) - u, b_y(y_i,\theta_i,s) - v) + \\ G(b_x(x_i,\theta_i+\pi,s) - u, b_y(y_i,\theta_i+\pi,s) - v))$$

This is also approximately equal to blurring the weighted edge map by scale varying Gaussians.

4.3.2 Calibration

Ideally, the interest points resulting from a shape that is rotating and changing in size would all have the same value. We introduce two scalar functions $norm_s$ and $norm_\theta$ to reduce scale dependent variations and angle dependent variations respectively. The values for these two functions are determined empirically using two calibration steps. They are used in the computation of the interest point salience maps as follows.

$$map_s(u,v) = norm_s(s) \sum_i norm_\theta(\theta_i) w_i(\; \delta(u - b_x(x_i,\theta_i,s), v - b_y(y_i,\theta_i,s)) + \\ \delta(u - b_x(x_i,\theta_i+\pi,s), v - b_y(y_i,\theta_i+\pi,s)))$$

These two calibration steps help the algorithm successfully compare interest points across scales and reduce variations due to rotation. The Cartesian discretization of the

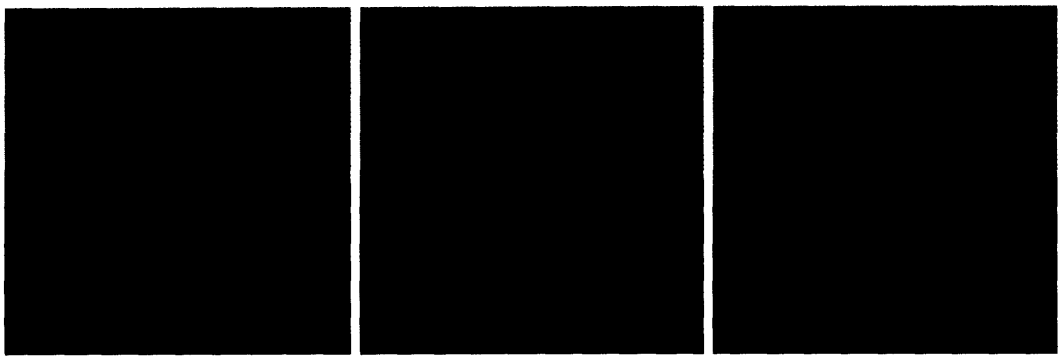


Figure 4-12: This figure shows three calibration images we use to help normalize the interest point values that would result from a shape as it varies in orientation and scale. For both normalizations in orientation and scale, we use this rotating half plane as input, which results in the scale-invariant shape of a straight edge. The angle of square calibration images increases from left to right. For the results within this thesis, the calibration steps use 360 rotated calibration images.

binary edges used as input for interest point processing leads to variation over rotation. Vertical and horizontal edges have more edge pixels per unit length than diagonal edges, which gives them more weight in the interest point algorithm. The specifics of the edge detection algorithm can also introduce variations over rotation. Antialiasing could reduce these sources of variation by representing each edge pixel with several properly weighted edge pixels. Although effective, this approach would substantially increase the processing required, since the computational complexity increases linearly with the total number of edge pixels. We instead mitigate the effect of these variations by performing additional edge thinning, scaling the edge weights with an empirically determined angle dependent value, and performing Gaussian smoothing on the resulting histograms.

We use the rotating edge, shown in figure 4-12, as the rotation calibration pattern. The mean value of a circular region of the resulting $smoothedmap_s(u, v)$ at some scale s is computed for each orientation of the rotating input image. These mean values are used to pick a scalar value for each orientation that would lead to a constant total, as shown in figure 4-13. These normalized orientation values would be likely to vary some with scale as well, which we could normalize during processing without much additional computation, but would require a single time consuming calibration of greater complexity. For the work we

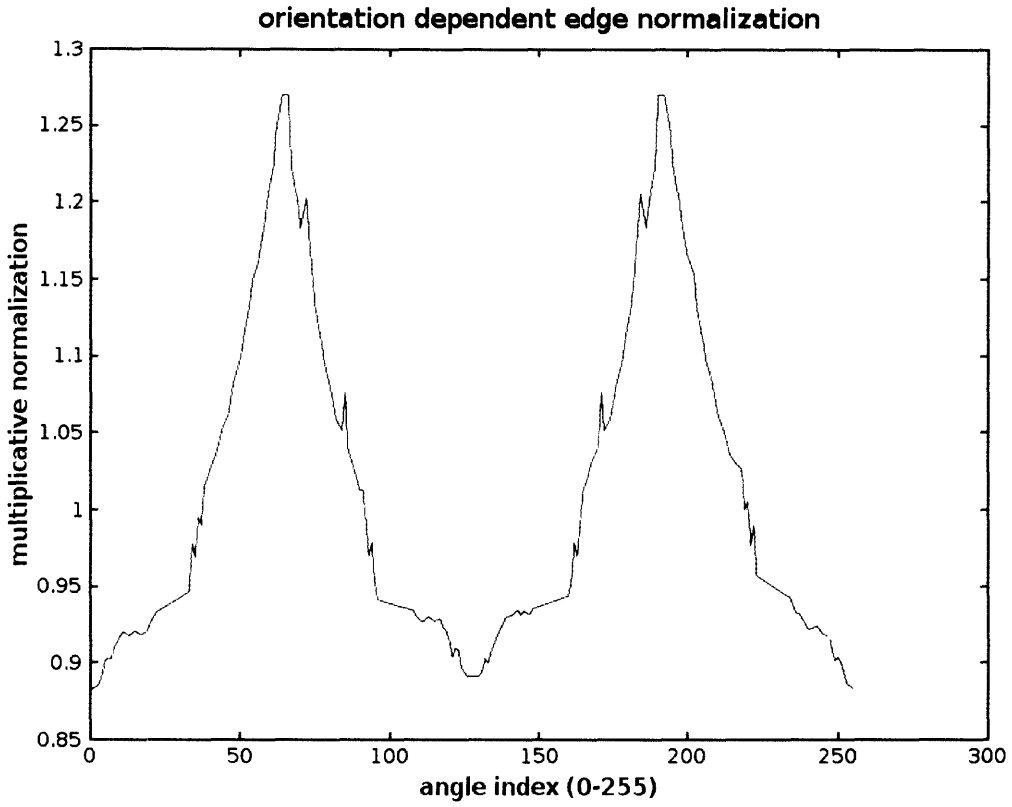


Figure 4-13: This figure shows an example of the resulting angle dependent normalization weights from calibration. The graph shows that edges of the same image length contribute different totals of votes due to their angle. The graph shows that the vote total, v_h, from a horizontal edge will tend to be related to the vote total, v_d, from a diagonal edge of the same length by $0.88v_h \approx 1.28v_d$. These empirically measured values make sense since $\frac{v_h}{v_d} \approx \frac{1.28}{0.88} \approx \sqrt{2}$ is the ratio we would expect given edges with Cartesian sampling and square pixel. A pixel with sides of length 1 will have a diagonal of length $\sqrt{2}$ and consequently an ideal diagonal edge must be multiplied by $\sqrt{2}$ to match the unit weight per unit length of a horizontal edge. It is also worth pointing out that this is not an insignificant difference in voting weight.

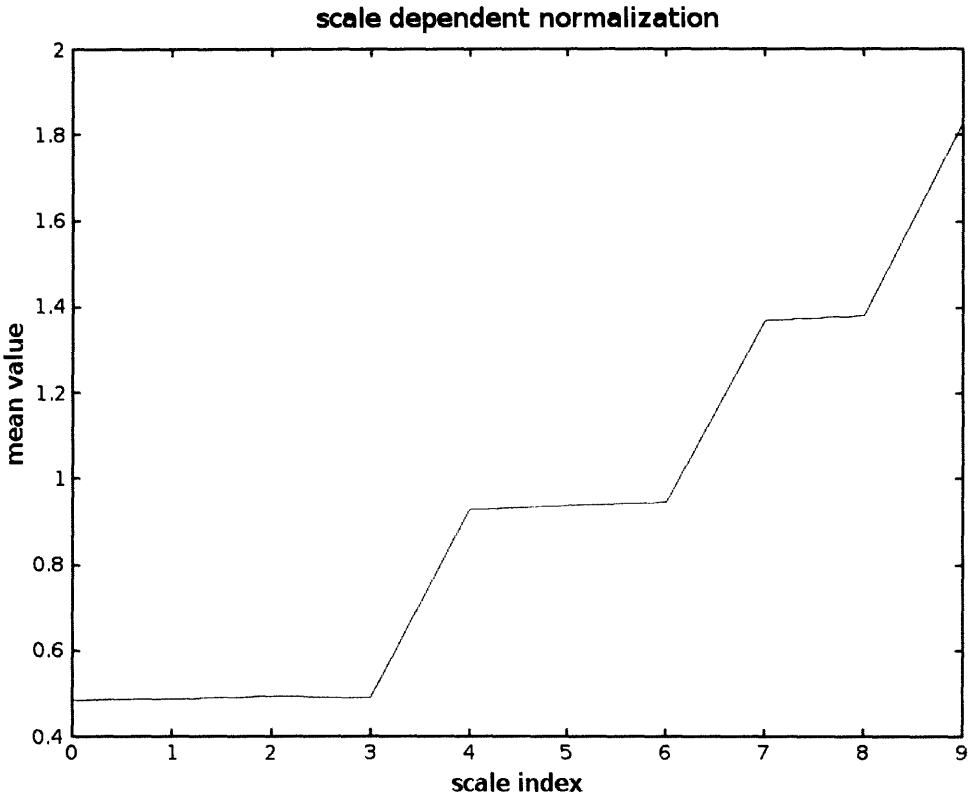


Figure 4-14: This figure shows the mean values of interest points indexed by scale that result from the input of the edge calibration image of figure 4-12. We normalize the maps so that these mean values become equivalent, since an edge viewed from any scale is still an edge and should produce interest points with the same weight. As we would expect, these empirically determined mean values increase with scale, since interest points at larger scales sum the votes from more edges.

present here, we take the simpler approach of picking a single scale and using the orientation normalization values from that single scale for all scales.

After normalizing variations due to rotations, we normalize variations over scale. Once again, we empirically determine the values for normalization rather than attempting to model the sources of variation in detail, see figure 4-14. Clearly some form of normalization is required, since as a shape is enlarged the total number of edge pixels representing its border increases, which causes corresponding interest points for the shape to have larger values. Additionally, some scale dependent variation results from the details of the quantization across scales.

4.3.3 Fourier Shape Features

By weighting the votes by functions of the edge's angle, f_1 and f_2, we can compute useful shape related features for each interest point.

$$\begin{aligned} maps_s(u,v) = \quad norm_s \textstyle\sum_i norm_{\theta_i} w_i (\quad & f_1(\theta)\delta(u - b_x(x_i, \theta_i, s), v - b_y(y_i, \theta_i, s)) + \\ & f_2(\theta)\delta(u - b_x(x_i, \theta_i + \pi, s), v - b_y(y_i, \theta_i + \pi, s))) \end{aligned}$$

The effectiveness of having two functions, f_1 and f_2, requires that the edge angle, θ, be restricted to a non-redundant angular range, such as $[0, \pi)$, which is sufficient to describe all the possible edge angles. Without this restriction, the application of one of the functions over another would not be distinct.

Setting f_1 and f_2 to Fourier basis functions can be used to detect parallel edges and corners, in contrast to the entire circles detected when they are constant. We can detect corners using

$$f_1 = \cos(\theta)$$
$$f_2 = -\cos(\theta)$$

and

$$f_1 = \sin(\theta)$$
$$f_2 = -\sin(\theta)$$

as shown in figure 4-15, to give us two sets of c maps each. The magnitude of these two corner maps responds strongly to rounded corners and arcs that subtend π radians, see

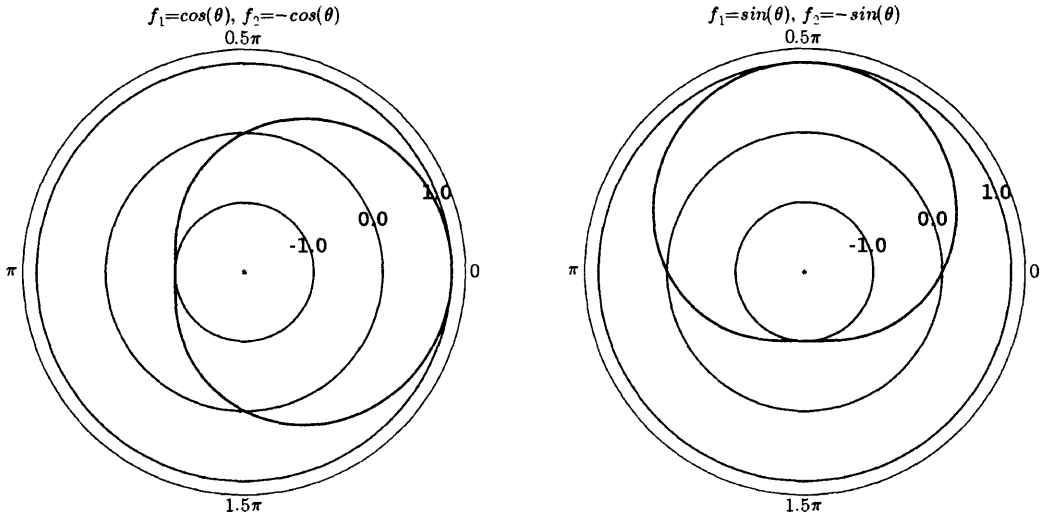


Figure 4-15: This figure shows the weighting functions used to generate complementary maps whose magnitude responds strongly to rounded corners and arcs, but weakly to parallel lines and circles, and whose angle points towards the interior of arc-like shapes. Two functions f_1 and f_2 are required, since each edge has two sides that vote. The result, however, is a single continuous weighting function for each vote as a function of the vote's angle θ from the edge, which is displayed in these graphs. In this case, the weighting functions are $\cos(\theta)$ and $\sin(\theta)$. The two values from a location on the resulting maps approximately correspond to the result of multiplying the edge distribution by these two functions. Notice how the weights from two sides of a full circle would cancel each other to zero. Likewise, notice that as an arc begins to subtend greater than π radians, the magnitude of its response begins to decrease. Finally, it's worth noting that perfectly parallel arcs will cancel each other out.

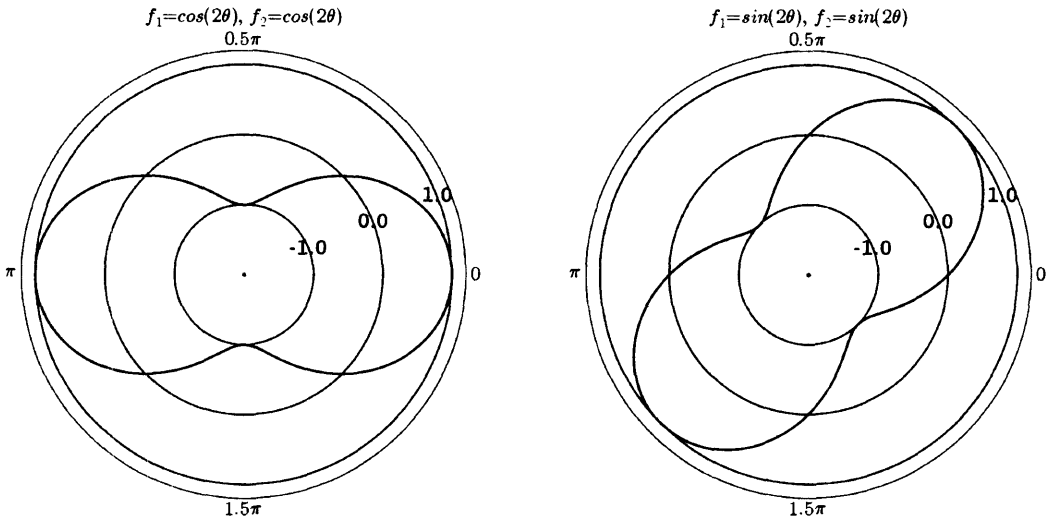


Figure 4-16: This figure shows the weighting functions used to generate complementary maps whose magnitude responds strongly to parallel arcs, but weakly to circles, and whose angle can be interpreted as pointing parallel to the parallel arcs. Two functions f_1 and f_2 are required, since each edge has two sides that vote. The result, however, is a single continuous weighting function for each vote as a function of the vote's angle θ from the edge, which is displayed in these graphs. In this case, the weighting functions are the Fourier basis functions $\cos(2\theta)$ and $\sin(2\theta)$. The two values from a location on the resulting maps approximately correspond to the result of multiplying the edge distribution by these two functions. Notice how the weights from two sides of a full circle would cancel each other to zero. Likewise, notice that an ideal arc subtending π radians would have zero magnitude. As an arc starts to subtend more than $\frac{1}{4}\pi$, the magnitude of its response begins to decrease. Finally, it's worth noting that perfectly parallel arcs will give a maximal response.

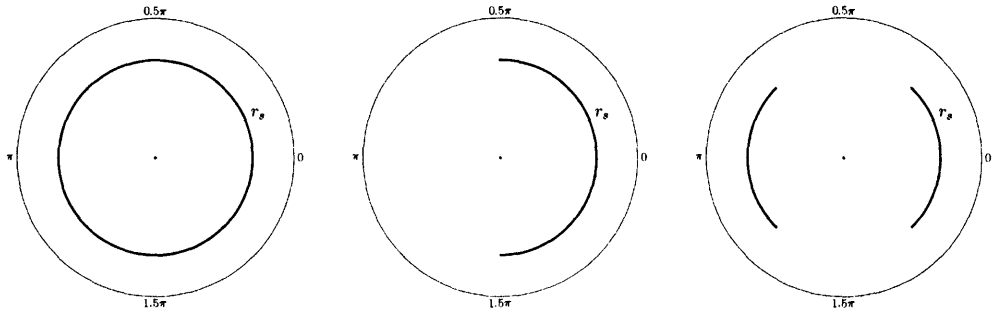


Figure 4-17: Given ideal edges with unit weight, and ideal continuous sampling of the scale-space, these three graphs show the edge input that would result in the largest possible magnitude for the three shape feature maps. From left to right, a circle would give the maximal response for the circle feature map, a half circle would give the maximal response for the corner feature map, and parallel arcs would give the maximal magnitude response for the parallel feature map. Please note that the orientation of these inputs will not affect the magnitude of the response, only the angle. These shapes correspond with the rectified Fourier components for frequencies of 0, 1, and 2.

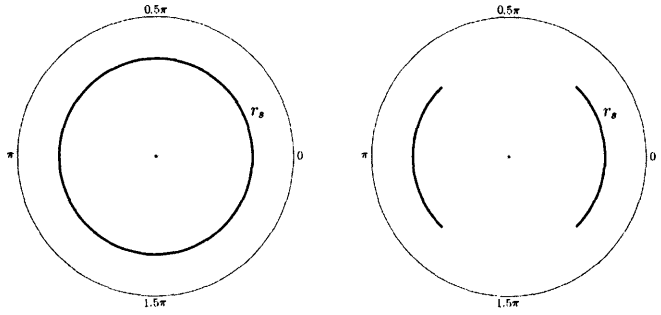


Figure 4-18: Given ideal edges with unit weight, and ideal continuous sampling of the scale-space, these two graphs show edge inputs that would result in a zero magnitude response from the corner shape map. Please note that the orientation of these inputs will not affect the magnitude of the response, only the angle. These shapes relate to the orthogonality of the Fourier components of frequency 0, 1, and 2.

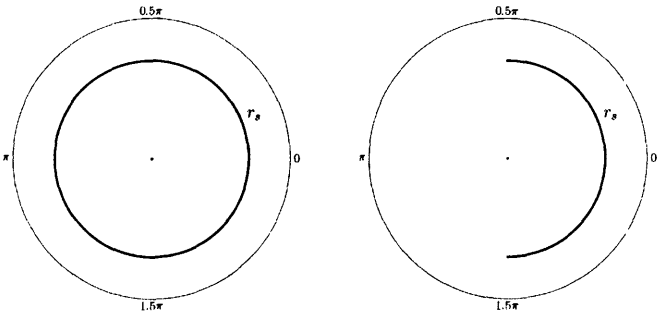


Figure 4-19: Given ideal edges with unit weight, and ideal continuous sampling of the scale-space, these two graphs show edge inputs that would result in zero magnitude for the parallel shape map. Please note that the orientation of these inputs will not affect the magnitude of the response, only the angle. These shapes relate to the orthogonality of the Fourier components of frequency 0, 1, and 2.

figure 4-17, but responds very weakly to parallel lines and circles, see figure 4-18. The angle of these two corner maps points toward the interior of the shape.

We can detect parallel edges by doubling the frequency

$$f_1 = \cos(2\theta)$$
$$f_2 = \cos(2\theta)$$

and

$$f_1 = \sin(2\theta)$$
$$f_2 = \sin(2\theta)$$

as shown in figure 4-16, to give us two more sets of c maps each. The magnitude of these two parallel maps responds strongly to parallel arcs, see figure 4-17, but responds very weakly to corners and circles, see figure 4-19. The angle of these two parallel maps can point parallel to the parallel arcs.

Clearly we could continue to find the responses to higher frequencies, until we could reconstruct a function that represents the summed input votes as a function of angle, but we've had success with these three lowest frequency components that do a reasonable job of balancing fidelity, generality, and computation. In addition, higher frequencies are likely to correspond with uninteresting noise in the input. Example magnitude maps resulting from an ideal rectangle are shown in figure 4-20.

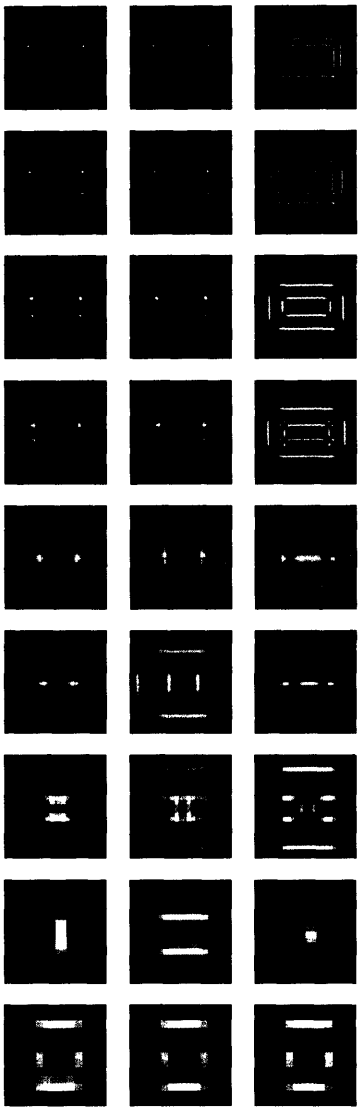


Figure 4-20: From left to right, these three columns of images show the magnitude of the circle map, the corner map, and the parallel map when given an input of a white rectangle. Scale increases from top to bottom, and each row shows the aligned responses for a given scale. The figure shows that features map locations that correspond with the corner's of the rectangle, the parallel lines of the rectangle, and the ends of the rectangle respond as we would expect from the discussion within this section.

4.3.4 Computational Efficiency

The complexity for computing the feature maps map_s is linear with respect to the number of edge points. For a single set of feature maps, map_s, over scale each of the n edge points must update two bins at each of the c scales, resulting in $2cn$ bin updates. For the five sets of shape feature maps this gives $10cn$ bin updates, which in practice is small enough for real-time video processing.

4.3.5 Filtering the Interest Points

Once we have these various interest point maps, we would like to filter out points that are uninteresting and select points that are likely to correspond with a salient region. We do this by selecting points that are local maxima in position and scale. If the circular magnitude is less than a threshold, we also filter out the interest points. If we calculate the circular magnitude map from unweighted edges, then after calibration we should set this threshold to a least remove any interest points that primarily correspond with a single edge point. More generally, the values of the circular map can rank the interest points across scale. We use this method to select the top 10 interest points to obtain the segments we process in chapter 5. We can also select desirable points based on the magnitudes of the shape maps. Two powerful measurements are the ratio of the parallel magnitude map to the circular magnitude map and the ratio of the corner magnitude map to the circular magnitude map. These ratios characterize the shape at the interest points, but also normalize out fluctuations due to edge weighting. If the parallel ratio is high, it means that the majority of the edges contributing to the interest point form a parallel shape. Likewise, if the corner ratio is high, it means that the majority of edges contributing to the interest point form a corner shape.

4.3.6 Integral Region Features

After filtering the interest points based on their Fourier features, the strength of their response, and non-maximum suppression, the system can quickly compute a variety of region based features for each of the remaining interest points using the "integral images" of various feature maps, as popularized by Viola and Jones in [61]. A conservative region would use the inscribed square for the circle associated with each interest point. We've experimented with several types of distinctive integral image features. Most of these region

features use the standard method of efficiently creating an "integral image" , N, from a feature image, I, with the same dimensions as the original edge image using

$$N(x,y) = \sum_{v=[0,y]} \sum_{u=[0,x]} I(u,v)$$

and then efficiently computing the sum, S, of the original feature image, I, over a rectangular region defined by $[x_{left}, x_{right}, y_{top}, y_{bottom}]$ with

$$S(x_{left}, x_{right}, y_{top}, y_{bottom}) = N(x_{right}, y_{bottom}) - N(x_{right}, y_{top}) - N(x_{left}, y_{bottom}) + N(x_{left}, y_{top})$$

For a given interest point at bin location (x_b, y_b) and scale s the square region would be defined in terms of the circular region of I with center $(m_s x_b, m_s y_b)$ and radius r_s, which can sensibly be translated into the square circumscribed by this circle

$$x_{left} = m_s x_b - \frac{r_s}{\sqrt{2}}$$
$$x_{right} = m_s x_b + \frac{r_s}{\sqrt{2}}$$
$$y_{top} = m_s y_b - \frac{r_s}{\sqrt{2}}$$
$$x_{bottom} = m_s y_b + \frac{r_s}{\sqrt{2}}$$

or the square inscribed with this circle

$$x_{left} = m_s x_b - r_s$$
$$x_{right} = m_s x_b + r_s$$
$$y_{top} = m_s y_b - r_s$$
$$x_{bottom} = m_s y_b + r_s$$

In order to better compare these region related features across scales, we typically use the average value, A, across the region.

$$A(x_{left}, x_{right}, y_{top}, y_{bottom}) = \frac{S(x_{left}, x_{right}, y_{top}, y_{bottom})}{(x_{right} - x_{left})(y_{bottom} - y_{top})}$$

We have used this method for a variety of feature types including color components and the Laplacian of color components. For color components we've used RGB and saturation and value from HSV, as well as grayscale values.

4.3.7 Fourier Integral Features

For cyclic quantities, such as hue and gradient angle, the sum or average value of the quantity has the undesirable effect of neglecting the cyclic nature of the feature, and hence

misrepresenting the distances between values. Moreover, sometimes we would like to take advantage of the power of histograms for representing distributions of feature values over a region, rather than relying on the average value. For some features, such as orientation and brightness, we may also wish to introduce some invariance to additive change.

For cyclic quantities, we could use two feature images in order to represent the quantities as unit vectors with angles that represent the value of the quantity. Averaging these vectors gives a measurement that represents the cyclic nature of the features at the cost of additional computation. We can extend this approach by applying a function f to the feature values prior to computation of the integral image.

$$N_f(x,y) = \sum_{v=[0,y]} \sum_{u=[0,x]} f(I(u,v))$$

In which case N_{sin} and N_{cos} would give us the unit vector representation. By using additional Fourier basis functions for f we can represent the distribution of feature values over the region with increasing levels of resolution in frequency. Moreover, the magnitude of these paired components, such as $(N_{cos}^2 + N_{sin}^2)^{1/2}$, can serve as shift invariant features describing this distribution.

4.3.8 Results

We have performed a variety of informal tests with this interest point operator, associated shape descriptors, and integral features for tracking and object correspondence. However, we do not report these results here. Within this thesis, we do, however, use the basic interest point operator provided by the circular map to select interest points based on a motion weighted edge map. The top interest point is used to select the maximal motion position used for the weak hand signal of chapter 3, which is crucial for learning the kinematic model. The top 10 interest points within a threshold distance of the estimated hand position are used to initialize the image segments collected for chapter 5, which results in segments of the wearer's hand and the manipulated objects. The interest point position and radius serve as a good way to initialize the log-polar segmentation method of the next section. For intuition about the parallel shape features we show two example images in figure 4-21.

Figure 4-21: These two images show the top 8 shape features at a single scale for these two images of beavers from the Caltech 101 database [16]. Blue circles mean that the interest point at the corresponding scale has a strong parallel magnitude, and the blue line in the middle of the circle shows the angle of the parallel shape. Green circles mean that the interest point at the corresponding scale has a strong corner magnitude, and the green ray in the middle of the circle shows the direction of the corner shape. The two beavers have strong parallel shape along the axes of their bodies.

4.4 Log-Polar Image Segmentation

We present an iterative algorithm for image segmentation that efficiently evolves a closed curve expressed as a radial function of angle. The curve evolution consists of two steps that are repeated until a convergence criterion is met. For the first step, given the current center of the shape, the algorithm finds a closed curve that approximately optimizes a parameterized cost function. To accomplish this, hypothesized borders are found along a set of rays that originate from the current center. Then, with respect to a rotation and scale-invariant cost function, an efficient shortest path algorithm finds a near optimal closed curve composed of hypothesized borders and illusory borders. Illusory borders are naturally incorporated into the optimization and relate to an explicit penalty term in the cost function. For the second step, a new center is found using the current closed curve. The new center is estimated based on the center of mass of the region enclosed by the curve.

By modeling the image segmentation problem using a graph within which a closed path must be found that optimizes some cost function, our approach is similar to algorithms such as normalized cuts [55] that partition a graph that represents the image in order to find a segmentation. The main difference with our work is that by using a simplified and approximate shape model with iterative optimization and a log-polar coordinate system, we are able to create a simplified graph that allows for rapid optimization via a shortest path algorithm rather than computationally intensive optimization methods, such as the spectral methods employed to find approximately optimal cuts on 2D graphs [46]. The form of our graph is essentially 1D instead of 2D and is specifically a topologically sorted, directed acyclic graph. By iteratively evolving the curve, our segmentation algorithm shares similarities to the vast number of curve evolution methods in the literature [51]. However, we do not use level-sets, or gradient based hill-climbing to find the segment, and instead use discrete, graph-based optimization methods. Likewise, our iterations involve two distinct steps, the first of which adapts the location of the segment's center, and the second of which adapts the shape of the segment with respect to the center. Many segmentation methods use clustering in various feature spaces to find regions with similar appearance [8]. Although our method makes use of texture features and a statistical appearance model, it is significantly different from most methods that use clustering in that it finds each segment independently instead of finding all segments simultaneously. It's interesting to note that

Figure 4-22: Results of using the log-polar segmentation algorithm on the wearer's arm.

evidence indicates that humans do use some form of visual segmentation early in their visual processing system [52, 2].

4.4.1 Task-oriented Segmentation

Image segmentation is a long extant sub-field of machine vision for which an enormous number of approaches have been proposed. Our approach relates to region growing, curve evolution, and graph-cut methods. By requiring segments to fall within the set of shapes described by our shape function, our approach also relates to work on shape representation. Complicated shapes outside of this set must be approximated or represented by multiple segments.

The greatest risk of using visual segments for visual perceptual processing, is that the segmentation algorithm may not produce the segments that are actually useful for the current vision task. One way to mitigate this problem is to allow a task-oriented vision system to bias the segmentation algorithm, thereby increasing the chances of selecting useful segments. Moreover, if each segment is found in succession rather than simultaneously, the results of each segmentation in the series can bias the next segmentation. From a biological perspective, one can consider this style of segmentation to be akin to an active vision system that saccades around the image segmenting regions in succession.

Rather than attempt to represent complex regions with a single segment, our algorithm uses many simpler segments in succession, thereby simplifying the computation of each segment and giving opportunities for the task to influence the segmentation process. Each

segmentation can be biased in a number of ways by the task-oriented attention system. For example, the attention system can bias the process by location, scale, textured region appearance, edge appearance, importance of edge features versus region features, curve smoothness, and curve continuity. Instead of looking for a single global segmentation for a given image, we iteratively look for simple segments as a function of several biasing parameters, which include a location and scale that can be provided by the visual interest point operator of the previous section. In order to produce segments across an entire image, the algorithm must be run for many settings of the biasing parameters, which may not be tractable over the entire space of parameter settings. This iterative formulation of global segmentation can have advantages for task-oriented image processing, since, for example, each segmentation in a search task can be biased by the previously found segments. For example, our algorithm might first segment a part of a tree trunk and then follow the trunk along its axis for further segmentations. One can think of this as allowing saccade-based strategies for vision, such as those used in active vision systems modeled after the human visual system. For this thesis, we use the output of the attention system in the form of a list of interest points to direct the process of visually segmenting the image.

4.4.2 Invariance

We require that the iterative curve evolution we use be invariant to position, scale, and rotation. More precisely, if we apply a transformation T, consisting of rotation, scaling and translation to the original input image I and the initialization parameters P, the result of applying the segmentation algorithm S should be the original series of evolved curves $\{C_1, C_2, ...C_n\}$ transformed by T. So, if

$$S(I, P) \rightarrow \{C_1, C_2, ...C_n\}$$

then

$$S(T(I), T(P)) \rightarrow \{T(C_1), T(C_2), ...T(C_n)\}$$

Consequently, we require that both the curve estimation step and the center estimation step be invariant to rotation, scaling, and translation, as this will result in the overall curve evolution be invariant to the transformations.

4.4.3 The Segmentation Shape

We restrict the segmentation shape to be a closed curve described by an image location $X = (x, y)$ and a polar function $d(\theta)$, which returns the distance, d, to the segmented region's border from the center, X, along each angle, θ. This set of shapes has been used by many researchers, since it includes useful shapes and has attractive computational properties. In this work, we sample the function $d(\theta)$ for n equally spaced values of θ, and hence represent the function as a vector D of length n, so that the entire segmentation can be represented as a vector $S = \{X, D\}$ of length $n + 2$.

For complex shapes that cannot be fully represented by a single segmentation S_1 we can represent the shape as the union of a set of segments, $S_{complex} = \{S_1, S_2, ...S_m\}$. Note that if we ignore sampling issues, any segment describes a unique shape, but that without further constraints a shape can be described by many different segments due to the use of approximate shapes.

4.4.4 The Appearance Model

Normalized histograms model the appearance of the interior and exterior of the closed curve. The segmentation algorithm takes as input a texture feature image that has been computed from the image to be segmented. The texture image has the same height and width as the original image, but each location holds an n-dimensional feature vector. The algorithm and the actual C++ code allow for any number of feature dimensions. Likewise, the actual structure of the normalized histogram is a design choice that can be tailored to the specific application. We have experimented with a number of different feature vectors and histogram models. For most of the work presented here, we use a six dimensional feature vector computed at a single scale, which includes y,u,v for brightness and color, and the magnitude of the gradients from the red plane, the blue plane, and the green plane of the original image. For the histogram, we use six independent one-dimensional histograms, each with 256 bin, so that the probability value associated with a given feature vector would be computed by multiplying together the six normalized values from these six histograms. The large number of bins works because these histograms are used to discriminate between interior and exterior membership, rather than to generate reasonable probability values.

The two histogram models, one for the interior and the other for the exterior of the

closed curve, are used to produce an interior versus exterior discriminant function. A feature vector is categorized as coming from the interior, if the probability value from the interior histogram is greater than the value from the exterior histogram. This is a maximum likelihood categorization of the feature vector with respect to the histogram models.

Rather than actually maintain and update an interior and an exterior histogram, we instead use a histogram for the interior and a histogram for the union of the interior and exterior areas. The advantage of this approach is that only the interior histogram needs to be updated upon a change in the closed curve. We could easily generate the exterior histogram by subtracting the unnormalized values of the interior histogram from the unnormalized values of the histogram of the union, so we know that these two histograms contain the same information. Depending on how many feature vectors we need to evaluate, we may choose to never explicitly compute the exterior histogram, and instead directly use the interior and union histograms. For example, we can determine the membership of a texture feature vector v using the following inequality:

$$\frac{\prod_i interior_i(v_i)}{\prod_i \sum_j interior_i(j)} > \frac{\prod_i (union_i(v_i) - interior_i(v_i))}{\prod_i \sum_j (union_i(j) - interior_i(j))}$$

where $interior_i$ represents independent interior feature histogram i, $union_i$ represents independent total image feature histogram i, v_i represents feature component i of the texture feature vector v, and summing over j sums all of the bin counts in the histogram, which facilitates proper normalization. If this inequality is true, then the feature vector v is marked as having interior appearance, otherwise it is marked as having exterior appearance. For efficiency, the denominators only need to be computed once given a particular set of histograms. For the rest of this chapter, we refer to the interior appearance model as the foreground appearance model, and the exterior appearance model as the background appearance model.

4.4.5 The Edge Model

The edge model is used to generate border hypotheses and to weight these border hypotheses based on their edge characteristics. Currently, we use a Canny edge detector with low thresholds to generate the set of hypothesized border points. The Canny edge detector serves two important roles when generating border hypotheses. First, with its thresholds, it throws out border hypotheses that sit within extremely smooth regions, which helps reduce

the chances of the closed curve passing through these locations. Without this filtering, smooth gradients such as those found on smooth walls and the sky can result in arbitrary boundaries along the gradient. Second, non-maximal suppression avoids overly redundant representations of the contours by thinning the edges. Additionally, our implementation of the Canny edge detector labels edge pixels that are strongly related to one another which is used later in the segmentation process. In addition to the Canny edges, border hypotheses can also be inserted at every transition detected by the appearance model, an approach that we have used successfully at times, but requires more computation due to the additional border hypotheses.

Texture edges and threshold adjustment are two common problems with using Canny edge detectors. We've found the segmentation algorithm to be somewhat robust with respect to these problems, since it also makes use of the appearance model and the border graph. The appearance model is able to reduce the influence of texture edges by classifying them as foreground or background points, while optimization of the border graph allows for missing edges to be interpolated in a reasonable way. If a class of images results in too many spurious edge hypotheses, the user has a variety of options including smoothing the image prior to processing, increasing the threshold used by the Canny detector, and a vast literature on filtering border points. The important aspect of this segmentation algorithm is that it can perform well over a wide range of edge maps, including situations where the edges are sparse with gaps in the ideal borders, and situations where the edges are too dense with respect to the ideal borders, due to noise and texture edges.

4.4.6 The Graph

Using the border hypotheses, we construct a graph to represent all the hypothesized closed contours, see figure 4-23. Each vertex in the graph corresponds with a border hypothesis and each edge in the graph corresponds with a hypothesized connection between two border hypotheses. Given a center X_1 and an image I, we wish to find an appropriate distance vector D. We assume that we have a method that produces a set of hypothesized borders, $H(\theta) = \{b_1, b_2, ...b_k\}$, for each angle θ, which in our implementation involves traversing the edge map from the center X_1 along a ray at each angle θ and inserting the border hypotheses for each edge encountered by the ray. We then create a directed graph, for which each hypothesized border is a vertex $v_{(\theta,d)}$ and each vertex at $\theta = i$ is connected to

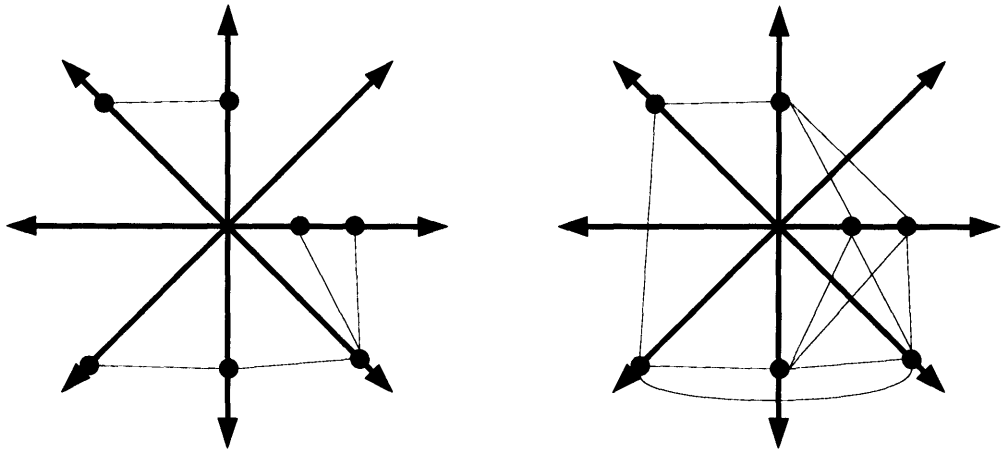


Figure 4-23: The diagram on the left shows a log-polar graph of border hypotheses with single connectivity and no skipping allowed, so that $c = 1$. The graph is improper and does not have a solution due to too few border hypotheses and no allowed skipping. The diagram on the right shows the same border hypotheses in a graph that allows skipping over a single ray of border hypotheses, so that $c = 2$.

all the vertices $\theta = i + j$ where $j = \{1, 2, ...c\}$ and θ is quantized to be one of a angles with $\frac{2\pi}{a}$ radians in between each ray. The integer value c dictates how many rays can be skipped when finding an optimal path, see figure 4-23. Each skipped angle results in an illusory border and a penalty, since the path will not travel through any of the hypothesized borders for these skipped angles. A closed path through this graph that starts and ends at the same vertex v and only loops through the angles once, corresponds with a distance vector D and a closed segment on the image with center X_1.

4.4.7 The Cost Function

We now wish to create a cost function for the graph that will lead to useful segmentation shapes. In order to efficiently find an optimal path using shortest path techniques, we restrict our cost function $Cost(path)$ to a form that can be calculated by summing the edge costs along a path. Since any cost associated with a vertex can be added to the cost associated with each incoming edge, we include vertex based costs for convenience.

$$path = \{v_1, v_2...v_n\} \; where \; (v_1 = v_n) \; and \; (\theta_i < \theta_{i+1} \forall i \neq n - 1)$$

$$Cost(path) = \sum_{i=[1,n-1]} EdgeCost(v_i, v_{i+1}) + VertexCost(v_i)$$

The optimal path should be invariant to scale, rotation and translation. Since we will handle translation invariance when we estimate the center of the segmentation, we only need to worry about scale and rotation here.

A vertex v has data associated with it, including the index of the ray on which it sits v_θ, the Cartesian position of the border v_x, the index of the edge group to which it belongs v_e, the number of locations categorized as having foreground appearance v_{fg} between it and the current center X_i, and the number of locations categorized as having background appearance v_{bg} between it and the current center X_i. Each ray Θ also has data associated with it, including the total number of locations categorized as having background appearance on the ray Θ_{bg}, and the total number of locations categorized as having foreground appearance on the ray Θ_{fg}.

For convenience, we restrict the *EdgeCost* function and the *VertexCost* function to be linear combinations of functions that can return values from zero to one, inclusive.

First we introduce a term for the cost of vertex to vertex distance in order to promote smooth paths and bias the segmentation to circular shapes. We calculate this distance function in log-polar coordinates which makes it invariant to rotation and scale

$$r(v) = \frac{a}{2\pi} log\left(\|v_x - X_i\|\right)$$

$$dist(v1, v2) = \left((r(v2) - r(v1))^2 + (v2_\theta - v1_\theta)^2\right)^{\frac{1}{2}}$$

Second, we introduce a term that penalizes skipping over a ray, which produces illusory edges.

$$skip(v1, v2) = \begin{array}{l} (v2_\theta - v1_\theta) - 1 \text{ if neither } v1 \text{ nor } v2 \text{ is a virtual border} \\ (v2_\theta - v1_\theta) - 0.5 \text{ otherwise} \end{array}$$

This is also invariant to scale and rotation, since it only uses the orientation of the two vertices. With only the *dist* and *skip* terms, the lowest cost shape is a circle of any scale that passes through a hypothesized border at each and every angle.

Third, we create a term that penalizes edges that are not in the same edge group e. Our modified Canny edge detector assigns the same group number e to edges that are likely to be part of the same continuous edge.

$$edgegroup(v1, v2) = \begin{array}{l} 1 \text{ if } v1_e \neq v2_e \\ 0 \text{ if } v1_e = v2_e \end{array}$$

This edge membership term encourages the segmentation system to find curves that use edges that are clearly related to each other. It's a coarse binary signal, but it is very efficient to compute and use within the graph.

Fourth, we add a term that allows a penalty to be associated with border hypotheses based on their salience, *bordersalience*. This is a vertex cost and is a general way of rewarding curves that use strong edges as measured by some other system. For example, they might be edges that have strong motion, or very high gradient magnitude.

Fifth, we add a term that penalizes vertices depending on how well they separate pixels in the foreground and background appearance models. While traversing the a ray, points are classified as being from the foreground model or the background model based on the current appearance models. While traversing the ray from the center outwards, a count of the number of pixels classified as foreground and a count of the number pixels classified as background are updated and assigned to border hypotheses as they are encountered.

$$b(v, \Theta) = \frac{(v_{fg} - v_{bg}) + ((\Theta_{bg} - v_{bg}) - (\Theta_{fg} - v_{fg}))}{\Theta_{bg} + \Theta_{fg}}$$

$$regionstrength(v) = 1 - \frac{b(v, \Theta_{v_\theta}) + 1}{2}$$

Out of these possible terms, we have had success using just the following five terms, and associated cost function:

$$EdgeCost = \alpha_1\, dist + \alpha_2\, skip + \alpha_3\, edgegroup + \alpha_4\, regionstrength$$

$$VertexCost = \alpha_5\, bordersalience$$

Setting the weights α_i biases the segmentation system to emphasize different properties that define a region, such as the border versus the foreground texture. For our work, we've

set these values by hand to values that have lead to generally useful performance. For example we've used the following settings for most of the work described within this thesis:

$$\vec{\alpha} = \begin{bmatrix} 1 & 3 & 1 & 3 & 0 \end{bmatrix}$$

These cost settings ignore the border salience term, give strong and equal weighting to the skip and region strength terms and give the same smaller weight to the distance and edge group terms.

It's not difficult to think of ways to automatically set these weights to constants given training data. It's less clear what methods one would use to set them dynamically, other than making use of contextual data to decide what settings might be more effective. An attention system could potentially set these weights given the task.

4.4.8 Finding the Shortest Path

We have constructed a directed graph with cycles. By converting it into an topologically sorted, acyclic graph with approximately the same optimal path, we can efficiently find a good approximate solution.

We first pick an angle with a maximally distant vertex to be our start and end angle. Then, before we construct the graph, we add a number of virtual borders to the list of hypothesized borders associated with this start/end angle. If we were sure that the optimal path would pass through one of the hypothesized border vertices at our start/end angle we would not need to add these virtual vertices. However, the optimal path may in fact skip over this angle and therefore have an illusory border, which would not correspond with any hypothesized border vertex. By adding vertices we can decrease the distance between a vertex and any possible illusory border, therefore bounding the error. We can make this error arbitrarily small, by adding more virtual border vertices, but each additional vertex increases the required computation. We place each new vertex at the center of the largest gap between all of the current vertices.

Now, after having added our virtual borders, we construct the graph described earlier in this section. We then take each vertex at this start/end angle, and substitute two new vertices, one of which has all of the incoming edges and is labeled as a start vertex, and the other of which has all of the outgoing edges and is labeled as an end vertex. Next, we remove all edges which jump over the gap we have inserted into the graph. More formally, if

we were to shift the vertex angles so that the start vertices have angle 0 and the end vertices have angle 2π, we would then remove any edge that ends at a vertex with an angle less than the angle of the vertex at which it starts. This construction results in a topologically sorted, directed acyclic graph.

Given this topologically sorted, directed acyclic graph, we can efficiently find the shortest path through by the traditional techniques of shortest paths [11]. We simply traverse the graph from the start vertices to the end vertices, angle by angle. For each vertex we maintain a list of pairs, each of which corresponds to the shortest path found from the associated starting vertex, and includes a path length and the preceding vertex for the associate path. The final result is the optimal path through the graph, which defines a path through the edge points associated with the border hypotheses at the vertices of the graph. We now create a new curve D by linearly interpolating between these edge points in the image.

4.4.9 Estimating the Center

Given the current center X_i and curve D_i we wish to estimate a new center X_{i+1}. At minimum we require that the update function be invariant to translation, rotation, and scale as represented by transform T. An update function that sets X_{i+1} to the center of mass, M, of the region defined by the curve D_i will be invariant to T, since, as is commonly known, the center of mass of a shape is invariant to T. This is easily shown using integrals over the region enclosed by D to define the center of mass M

$$M(D) = \frac{\int x}{\int 1}$$

which is clearly invariant to T, since T is a linear transform of position

$$T(M(D)) = T\left(\frac{\int x}{\int 1}\right) = \frac{\int T(x)}{\int 1} = M(T(D))$$

Likewise setting $X_{i+1} = X_i + \alpha(M - X_i)$ for a scalar α will be invariant since

$$T(X_{i+1}) = T(X_i + \alpha(M(D) - X_i)) = T(X_i) + \alpha(T(M(D)) - T(X_i))$$

which allows us to reduce the convergence rate by setting $0.0 < \alpha < 1.0$. We can define other reasonable invariant families of update equations. For example, one can define a set of update equations that are invariant to T, restrict X_{i+1} to be inside the convex hull of D_i, and require that the influence of a border on X_{i+1} vary monotonically with the border's distance from the current center, X_i.

4.4.10 Convergence

For this thesis we iterate the two steps of finding an optimal curve D and finding the center X until the following inequality is true:

$$\frac{(a_1 - a_i) + (a_2 - a_{i'})}{a_1 + a_2} < \tau$$

where a_1 is the area of the segment in the previous time step, a_2 is the area of the segment in the current time step, a_i is the area of the intersection of the two segments, and τ is a threshold below which the two segments are considered similar enough to stop the iterations and declare the segmentation converged. Notice that if the two areas are equal and fully overlap, then the left side of the inequality equals 0. Also notice that if the two areas are equal but do not overlap at all, the left side of the inequality equals 1. Alternatively, if a threshold for the maximum number of iterations is exceeded, the segmentation is stopped and the last segment returned.

4.4.11 Results

Despite impressive efforts to quantify the quality of visual segmentation algorithms, [39], evaluating a visual segmentation system in isolation is a challenging and potentially ill-posed problem. The appropriate visual segments can be task dependent, the number of potentially valid segmentations of a natural scene can be very high, and the significance of differences between an approximate shape and some ideal shape is unknown. Since our visual segmentation system produces sub-segments, that have approximate shape, evaluation is further complicated. In contrast, task dependent segmentation can be evaluated using standard methods, although often in these situations one ends up evaluating segmentation in terms of detection and recognition. Since our emphasis is on autonomous learning and discovery, we let the results of the next chapter serve as evidence of the efficacy of this visual segmentation algorithm. Within the next chapter we show results from the visual segmentation system when used to collect segments of hands and manipulated objects, along with examples of visual segments autonomously created and tracked through time.

Chapter 5

Associating Visual & Kinematic Segments

Using the attention and segmentation methods from the previous chapter, we can efficiently collect significant visual and kinematic segments from the database of captured experience. Within this chapter we use unsupervised clustering and tracking to autonomously find structure within autonomously collected sets of segments. The hand segments we analyze are 3D hand positions that correspond with the local minima of hand speed from the previous chapter. Through k-means and visualization, we show that with respect to the head and the torso these positions are well-represented by three to four intuitive clusters. We then use k-means to cluster appearance feature vectors that correspond with visual segments that were generated by a visual attention system tuned to segment the hand and manipulated objects. As we would expect, this process results in clusters that strongly represent the hand, and clusters that strongly represent objects manipulated by the hand. Next, we show example results from tracking visual segments over time. We then conclude by discussing the implications of these results for future research on machines that autonomously learn about everyday human manipulation.

5.1 Clustering Significant Hand States

Our kinematic segmentation method from section 4.1 uses the learned kinematic model to give us a set of times representing hypothesized transitions between significant hand-related activity. By clustering the 3D position of the hand at these transition points, we can rep-

resent the locations between which most significant hand motions take place. We find that for data set 1, which consists of a large number of distinct activities, these positions can be well modeled as being in one of three or four states. These hand states are sensible and correspond with intuitive interpretations and are informative for the recognition of kinematic activity. The results also have an appealing analogy with "place cells" from neuroscience. Experiments have demonstrated the existence of neurons that respond strongly and selectively to the presence of a stimulus in a particular position relative to an animal's body, and have further found neurons that are especially sensitive to the animal's hand [29, 45, 44, 23].

We use both the head frame and the torso frame to measure the positions for clustering. Each frame offers a coordinate system that is informative about particular activities. For example, the head coordinate system can show at what locations the hand is typically observed. The clusters demonstrate that the torso tends to serve as a relatively stable and consistent coordinate system during common activities.

Clustering with respect to the hand positions associated with segment points has several advantages to clustering over all positions. First and foremost, as one can see from the figures of section 4.1, the hand spends a significant percentage of its time in transit during many common activities. By using the locations associated with these segmentation points, we can bias the distribution towards the starting points and destinations of the hand, which allows us to model hand motions in terms of making transitions between these states. We can also use these clusters, which model the likely locations of action transitions, to improve the segmentations by using position as well as velocity. Finally, the number of transition points as measured by the segmentation algorithm is far smaller than the number of total positions. For example, the approximately 11000 captured configurations of the body in data set 1 result in approximately 550 segmented positions, which is a factor of 20 smaller. This directly results in very fast processing of the data set, including the clustering we perform in this section. With around '550 positions, we are able to run our k-means algorithm for many values of k, and many random initializations, in just a couple of minutes.

We cluster these positions using k-means, which is fast and effective for this data set. As can be seen in the figures, the high density areas of the distribution are compact and somewhat spherical. For the examples within this section, we clustered the data with k set to 1 to twelve. For each value of k we fit 10 randomly initialized k-means models and keep the model with the lowest error. A standard challenge with a model such as k-means, is how

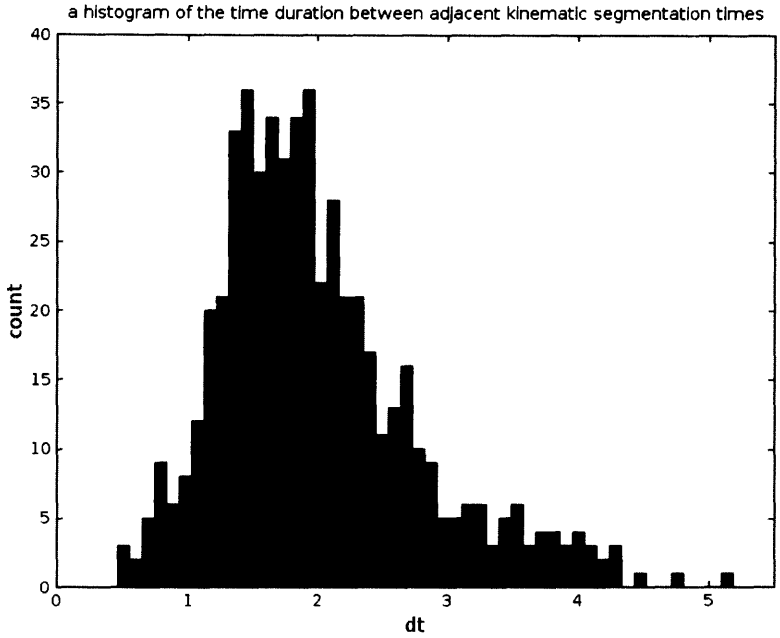


Figure 5-1: This figure shows a histogram of the time differences between adjacent kinematic segmentations found in dataset 1, using a filter with a scale of a half second. With this dataset and this filter, the salient transition points tend to occur at an interval of 1.75 seconds.

to avoid overfitting the data, since the error will continue to drop to zero in the limit as k goes to infinity [13]. A variety of measurements for model complexity have been developed to attempt to choose k and balance the benefits of lower error with the costs of higher model complexity. We are able to avoid these issues representing the error in terms of 3D distance in units of arm length. This allows us to select k in a reasoned way. Specifically we choose the first k that results in an average error less than a fifth of the arm's length, which is approximately the length of the hand and should match well with the error level of our hand position estimates, since we only measure wrist rotation and not bending at the wrist. This choice leads to a value of $k = 4$ for the positions measured relative to the torso, and $k = 3$ for the positions measured with respect to the camera.

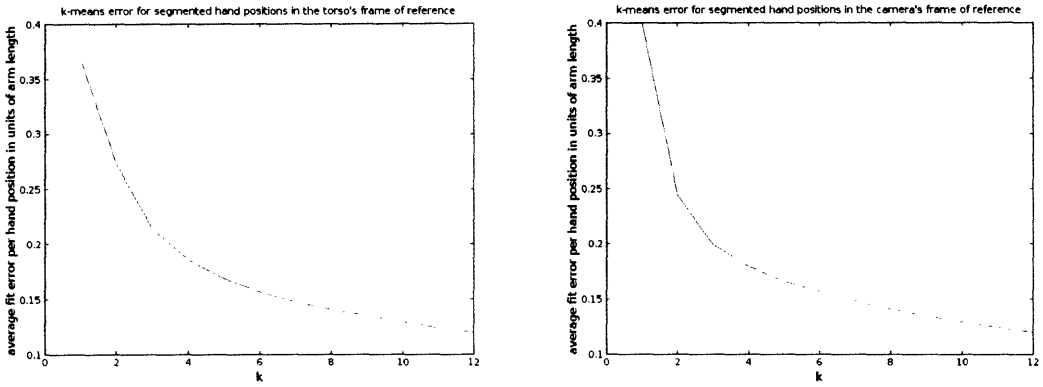


Figure 5-2: These two graphs show the dependency of the model error on k when using k-means on kinematically segmented hand positions from dataset 1. The left graph shows the fitting process on positions measured with respect to the torso and the right graph shows results on hand positions measured with respect to the camera. For each value of k, we fit 10 randomly initialized k-means models and keep the model with the lowest error. The key point to note with these graphs is that we can represent the error in terms of 3D distance in units of arm length. This allows us to select k in a reasonable way. Specifically we choose the first k that results in an average error less than a fifth of the arms length, which is approximately the length of the hand and should match well with the error level of our hand position estimates, since we only measure wrist rotation and not bending at the wrist.

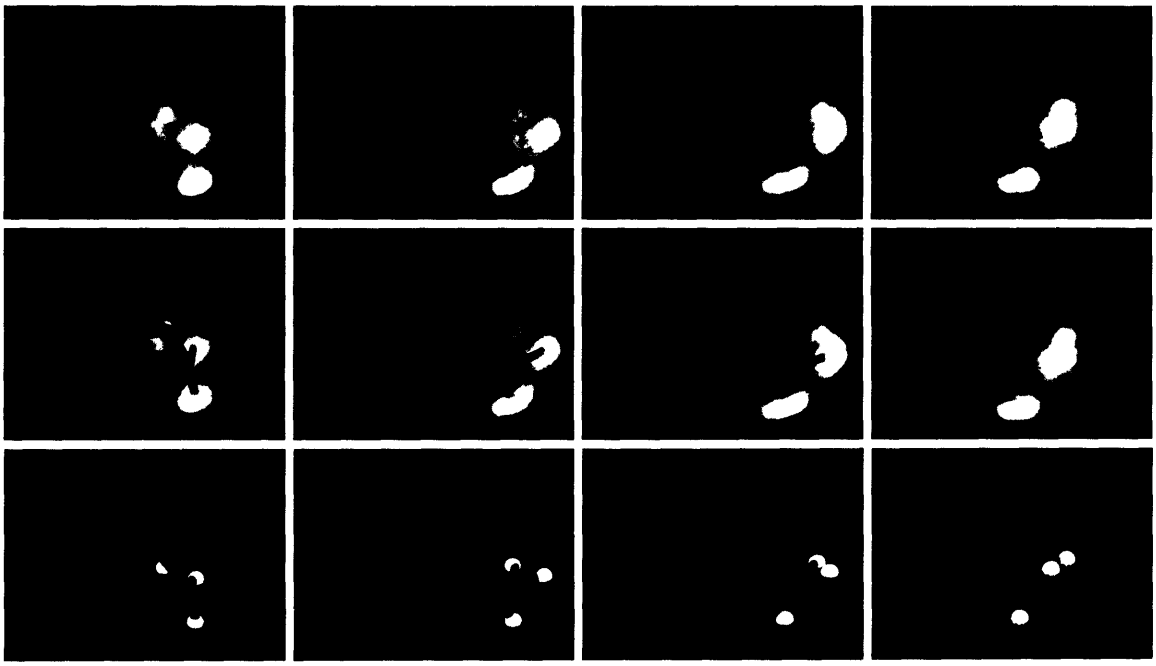


Figure 5-3: This figure shows the three means resulting from the k-means clustering we performed on the kinematically segmented hand positions as measured with respect to the camera's frame of reference. The distribution of 3D hand positions is shown alone in the top row. In the second row the four configurations of the learned kinematic model that best match the four resulting clusters are shown with the distribution. On the last row, these kinematic configurations are shown with white spheres representing the mean locations. In order to convey the 3D structure of the data, the columns are rotated versions of one another. The first column corresponds with us looking through the back of the camera.

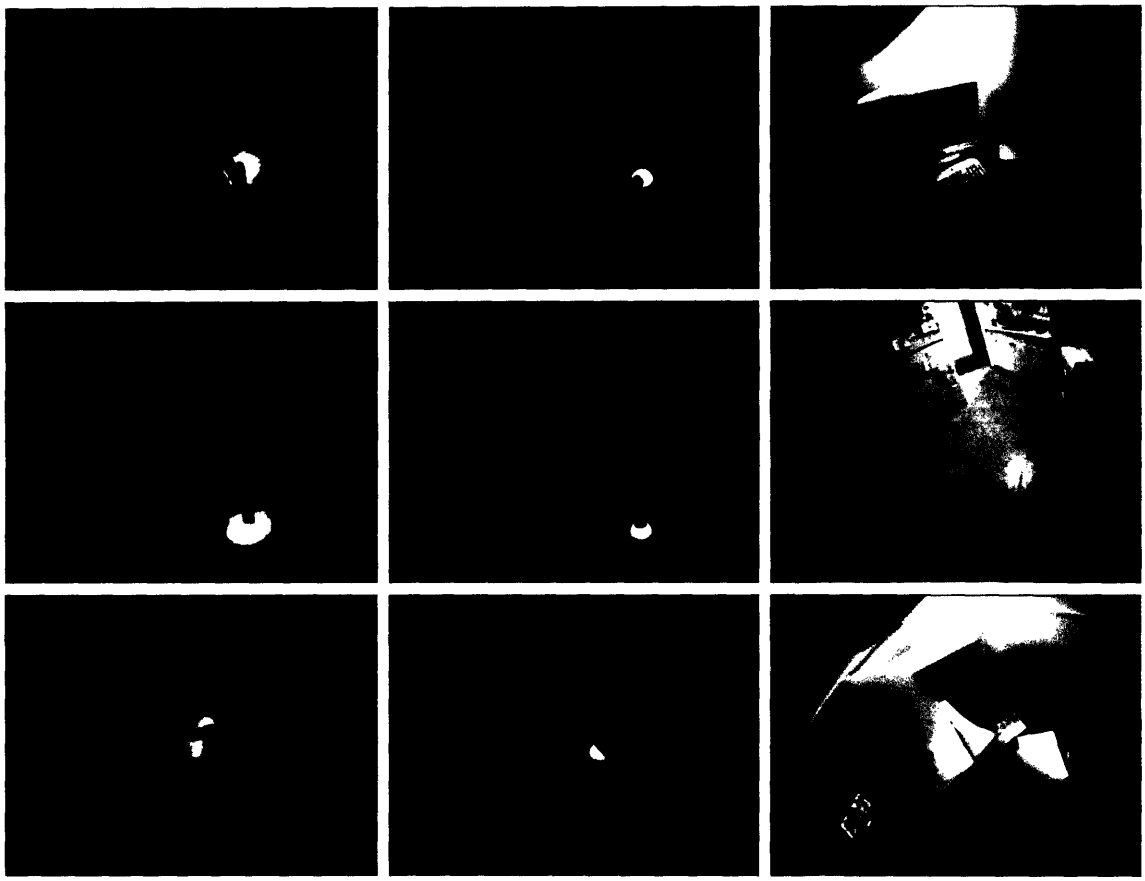


Figure 5-4: This figure shows the three means resulting from k-means clustering on the segmented hand positions with respect to the camera's frame of reference. The first column shows the configuration of the arm associated with the best matching position for each cluster along with the part of the distribution closest to that particular mean. The second row shows the same arm configuration with a white sphere located at the mean. The final row shows the image associated with the best matching hand position for each cluster. The top cluster corresponds with the head looking at the hand while reaching in the world. The middle cluster corresponds with the hand being at rest by the wearer's side. The bottom row corresponds with the hand being close to the camera and relatively centered. Fitting a line between the top cluster and the bottom cluster might indicate the gaze direction of the wearer relative to the camera.

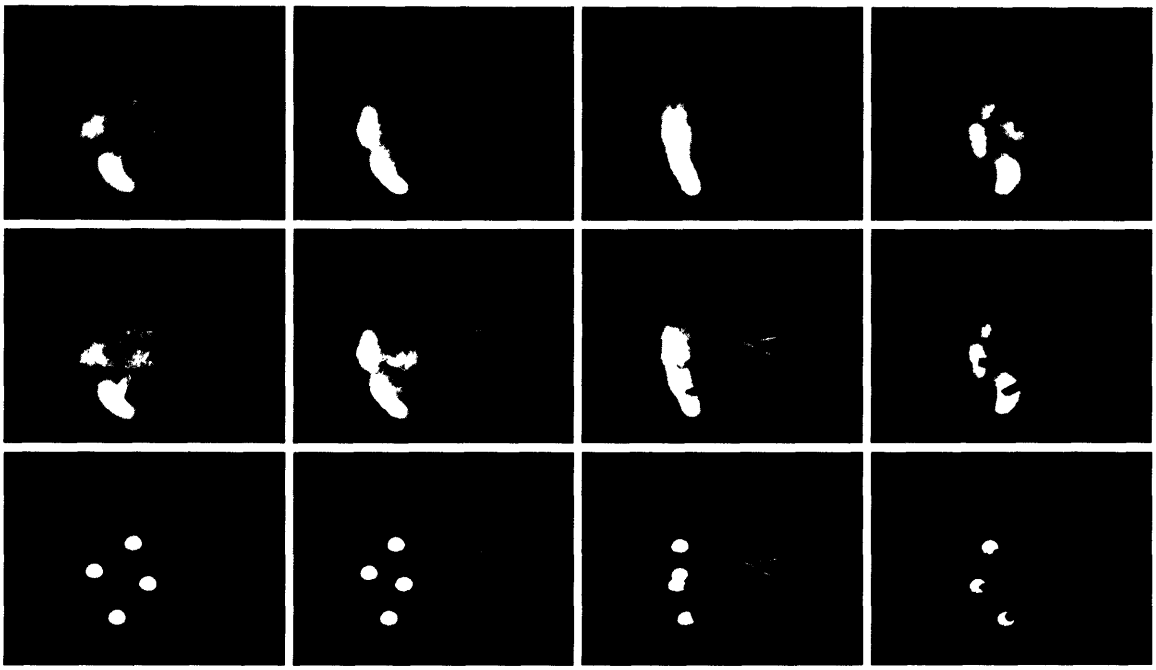


Figure 5-5: This figure shows the results of k-means clustering performed on the kinematically segmented hand positions as measured with respect to the torso's frame of reference. The distribution of 3D hand positions is shown alone in the top row. In the second row the four configurations of the learned kinematic model that best match the four resulting clusters are shown with the distribution. On the last row, these kinematic configurations are shown with white spheres representing the mean locations. In order to convey the 3D structure of the data, the columns are rotated versions of one another. The learned kinematic model does not result in a strong canonical orientation for the torso, since it does not estimate the torso's major axis. These images are shown with respect to the torso's unnormalized coordinate system, so the model is leaning back with respect to world coordinates. The configurations can be more easily in the next figure, in which the means are shown in isolation from one another and adjacent to a corresponding image.

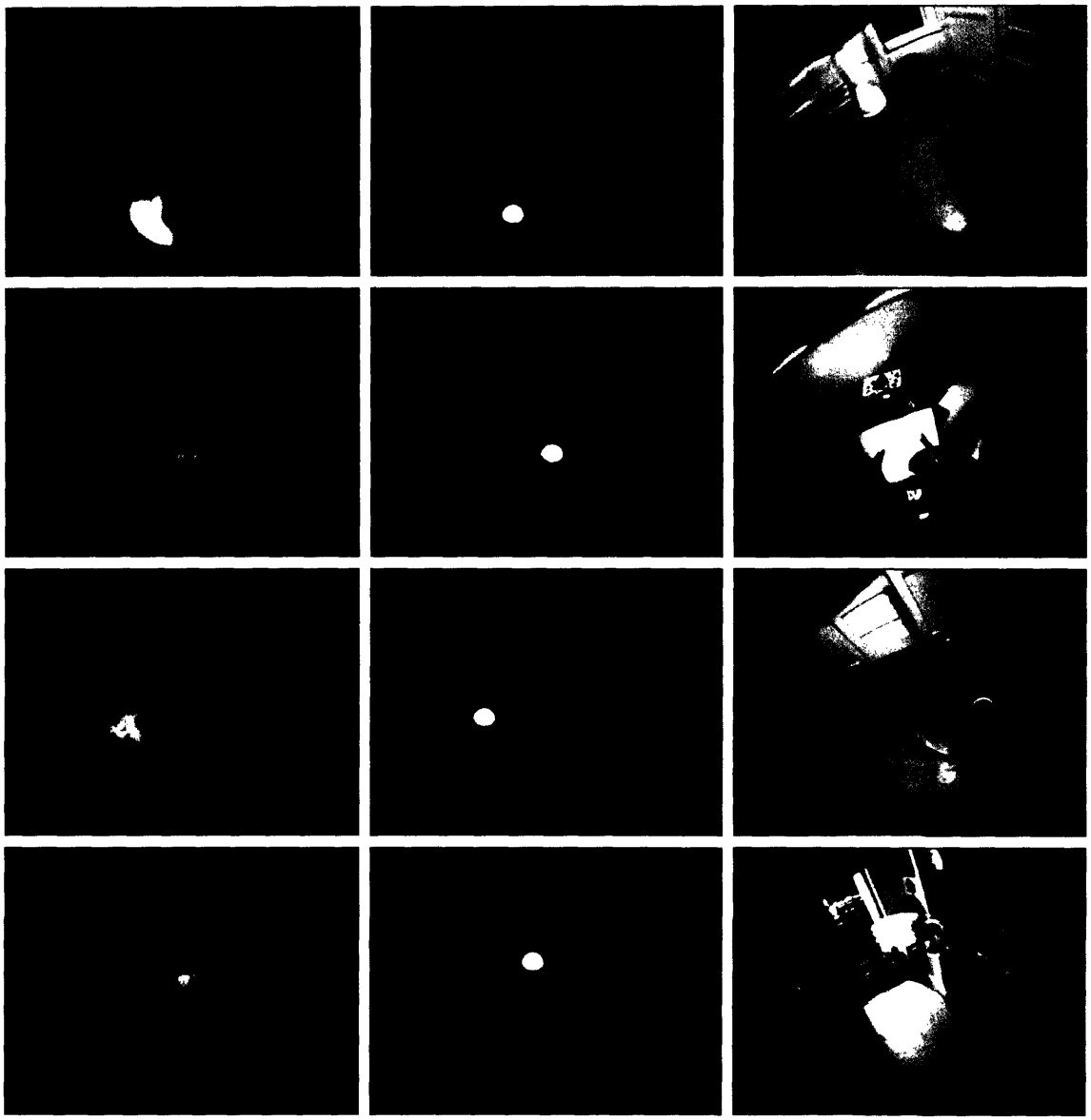


Figure 5-6: This figure shows the four means resulting from k-means clustering on the segmented hand positions with respect to the torso's frame of reference. The columns are analogous to those of figure 5-4. The top cluster corresponds with the hand being a the torso's side. The second cluster corresponds with the hand being in front of the torso in a location at which people often manipulate or hold objects. The third row corresponds with the hand reaching away from the torso into the world. The bottom row shows the hand being held close to the front of the head. The cluster for the hand being in the rest state is particularly strong, which corresponds with our intuition for the rest state.

5.2 Visual Discovery of the Wearer's Hand and Manipulated Objects

During a manipulation event, objects that move with the hand and against the background are likely to be related to the manipulation event. We bias the segmentation system to find hands and manipulated objects using our visual attention methods from the previous chapter. The attention system kinematically selects times at which the hand is highly visible in the image and the hand is moving rapidly in the image. During most manipulation events the wearer will at some point observe the hand. Likewise, if the hand is not visible to the camera our system will not have an opportunity to visually learn about the manipulation event anyway. This simple filtering step rapidly removes approximately 90% of the images from data set 1. With the remaining images, we only need to process the area around the kinematically predicted location of the hand, which on average effectively throws out 50% of each image. At moments when the hand is moving rapidly in the image, the visual interest point operator, applied to an edge map weighted by foreground motion, is more likely to select interest points at positions and scales that correspond with the hand and any objects being manipulated. We initialize 10 visual segmentations at the scales and positions of the maximally responding motion weighted interest points that are within a threshold distance from the kinematically estimated position of the hand within the selected images. We then attempt to remove segments that are approximately duplicates of another of the 10 member segments. When applied to data set 1, this process collects approximately 12000 visual segments.

We cluster appearance feature vectors associated with these 12000 resulting visual segments using k-means with $k = 10$. For the appearance feature vector we use three 16 dimensional 1D histograms for the hue, saturation, and value inside the segment, and one 128 dimensional vector representing a 16x16 square, brightness normalized image patch of the segment scaled to fit within the small square. Prior to concatenating these four components into the resulting 176 dimensional feature vector, we compute the PCA representation of the four raw individual feature vectors, reduce their dimensionality, normalize the total variance for each of them, and weight them. For the particular results we show here, we do not whiten or reduce the dimensionality of the three color histograms, which start out as 16 dimensional 1D histograms, so PCA for these three components does not actually serve

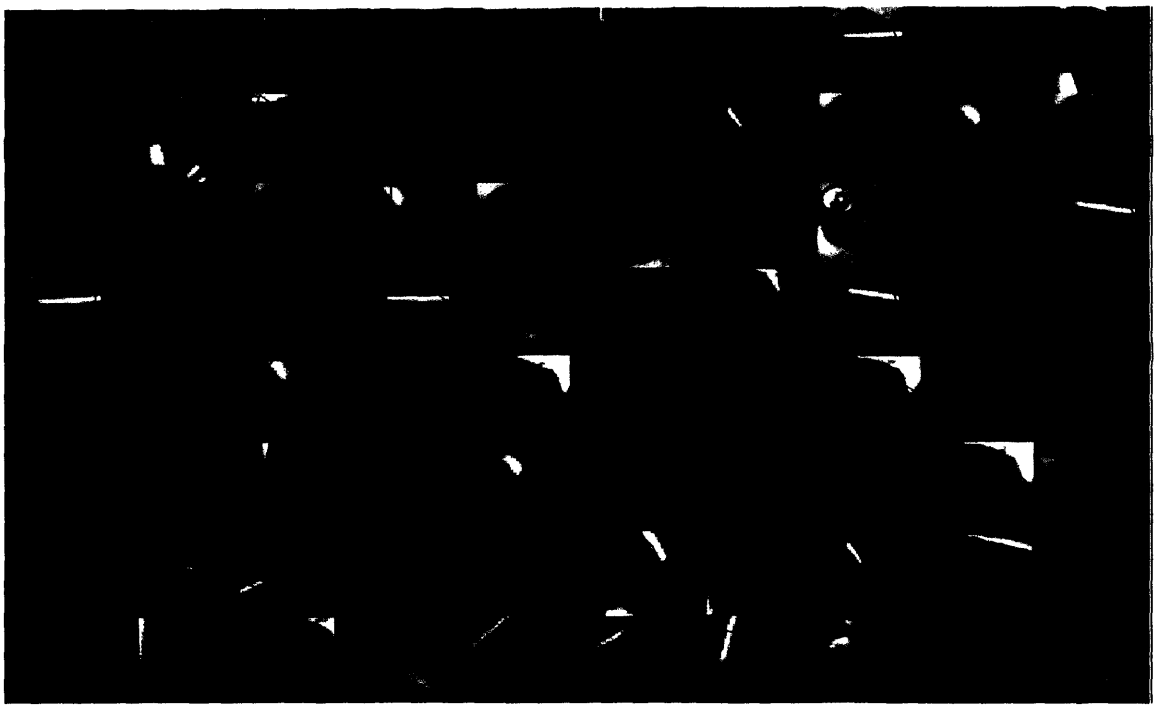


Figure 5-7: This figure shows the 79 member segments closest to one of the k-means clusters. The blue polygon shows the polygon generated by the associated log-polar segmentation. A rectangular image patch that bounds the polygon is shown for each segment. This cluster primarily consists of segments of the wearer's hand. Note that there are some similar segments, since the interest point operator initializes 10 segments per kinematically interesting image, and we only remove obvious duplicates.

any role in the k-means clustering. The original dimensionality of the 16x16 image patches is 256, which we reduce to 128 dimensions by dropping the minimum variance dimensions as determined by PCA. Prior to concatenating these four component feature vectors we weight them, multiplying the hue, saturation, and image patch components by 0.3 and the value histogram by 0.1, so that brightness differences are given less weight.

We show the best matching members of these clusters in the following 10 figures: 5-7, 5-8, 5-9, 5-10, 5-11, 5-12, 5-13, 5-14, 5-15, 5-16. Hands strongly represent three of the clusters, while several of the other clusters are strongly represented by hand held objects manipulated during data set 1.

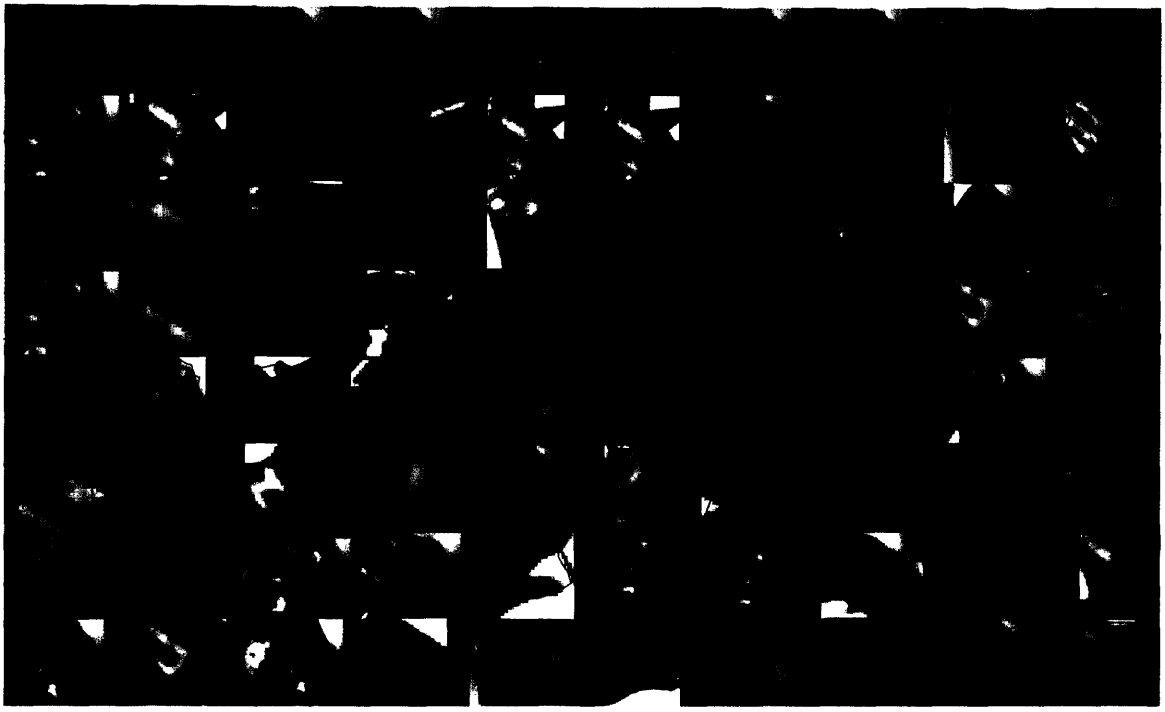


Figure 5-8: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. This cluster mostly contains hand segments.

5.3 Tracking Segments Through Time

Tracking segments over time provides another method by which to associate visual segments with one another, which takes advantage of the dense time sampling of video. For offline tracking we have used a shortest paths algorithm applied to visual segments in consecutive frames. Although this process is relatively inefficient, shortest paths provides a simple, flexible, and effective framework for offline tracking of visual segments. At each frame i the system creates a set of visual segments $S_i = \{s_1, s_2 \ldots s_n\}$ by segmenting locations selected by the attention system. We define a cost function C that assigns a cost for matching one segment to another. The cost function C can take into account a number of features of the segment, such as position, shape, and appearance. The system then performs all points shortest paths on a directed acyclic graph formed by connecting all the segments in S_i with all the segments in S_{i+1} using directed edges with costs defined by C. We can vary the graph by sequentially connecting m segmentation sets, $S_i$to S_{i+m}. Likewise, if we wish to allow segments to disappear for some number of frames we can insert additional directed

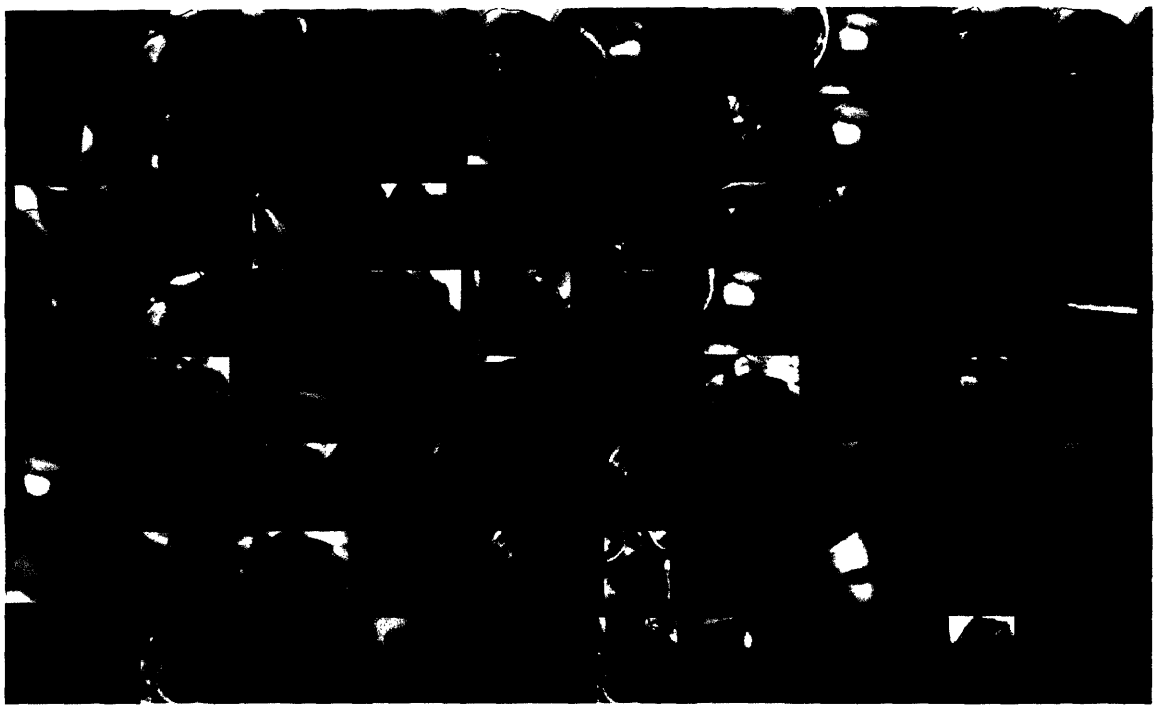


Figure 5-9: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. This cluster mostly contains hand segments.

edges that skip j frames and define a cost function $C_{skip}(s_\alpha, s_\beta, j)$ that takes the number of skipped frames as an additional argument that will give an appropriate penalty for the disappearance of a segment.

Besides associating segments across time, this method tends to filter out segments that are sporadic and of low quality. Segments that do not correspond well with object boundaries will tend to be filtered out because they will tend to vary as the underlying, unrelated regions move independently. Likewise, redundant segmentations will compete to match with segments in the next frames, which tends to reinforce segments that are stable over time. Segments in the foreground will tend to be occluded less than segments in the background, which also biases the resulting paths to relate to foreground objects, which is desirable for our purposes, see figure 5-17. The total costs for the surviving segment paths of length m can be used to compare these segment paths with one another and provides a measure of salience for the segment paths where more stable paths are considered more salient, see figure 5-18. In addition, we can custom design the cost function to select paths of interest, such as paths whose color model matches skin color, see figure 5-19. The major drawback

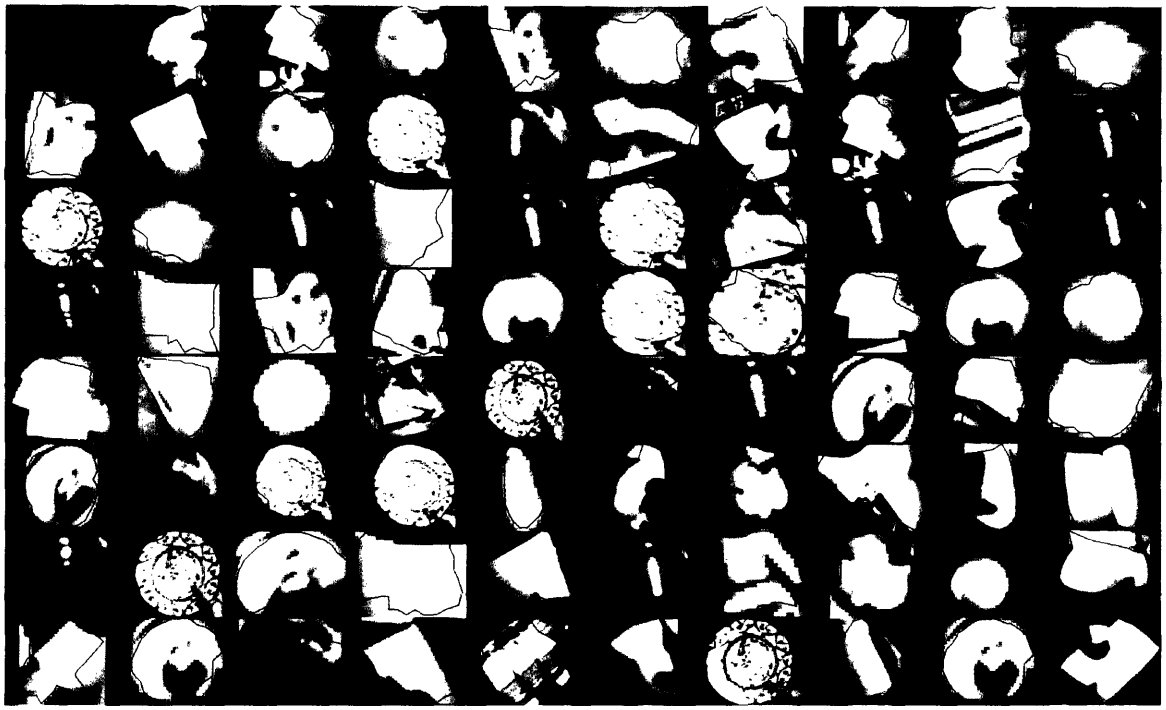


Figure 5-10: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. The top members mostly consist of manipulated objects including a bowl, a plate, and a coke can.

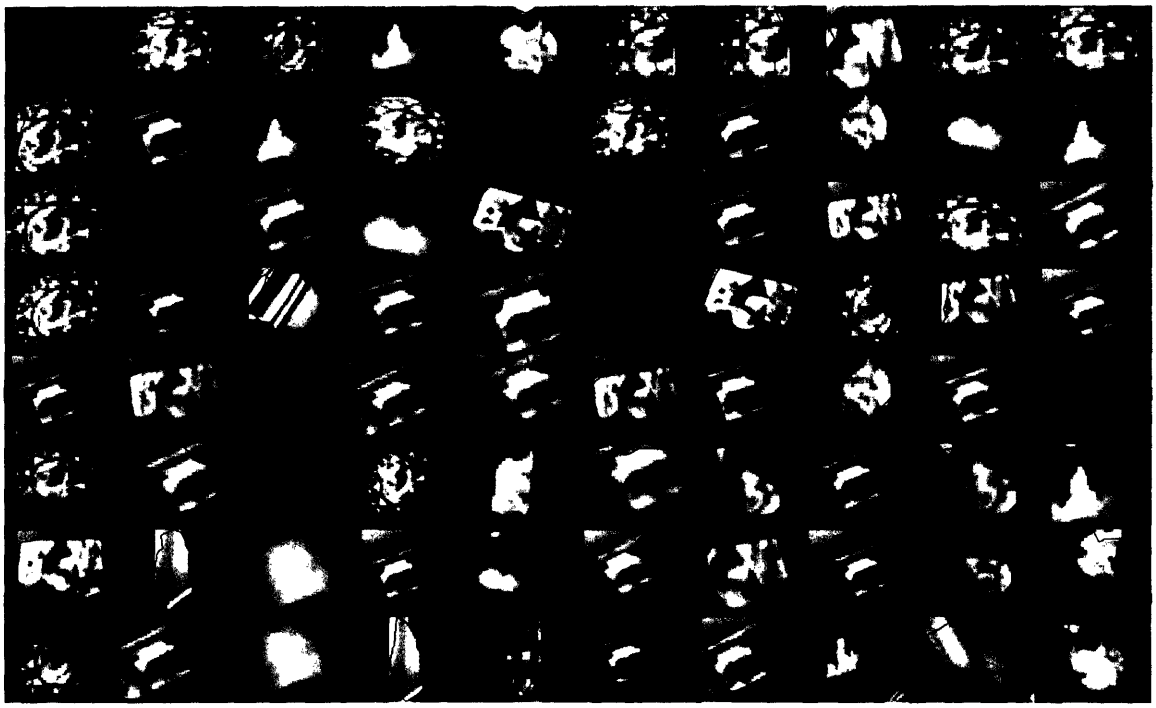


Figure 5-11: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. The top members mostly consist of manipulated objects including a plate, and a framed picture.

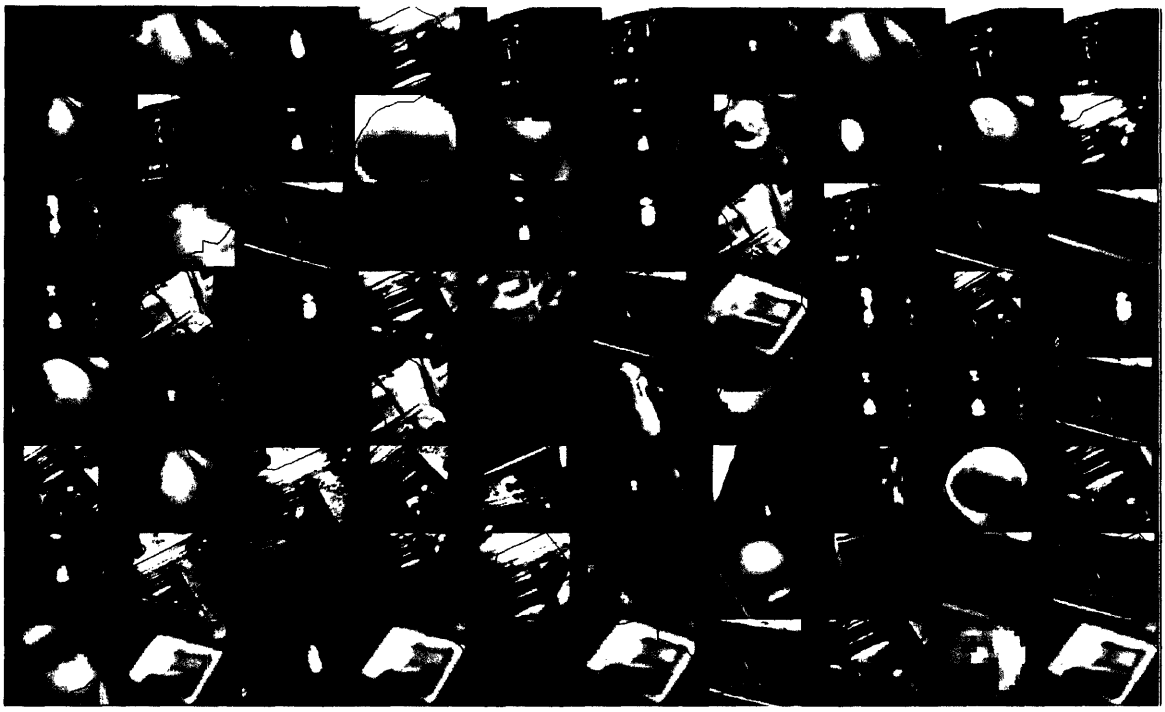


Figure 5-12: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. The top members mostly consist of manipulated objects, including a bowl, a silverware drawer, and a cup.

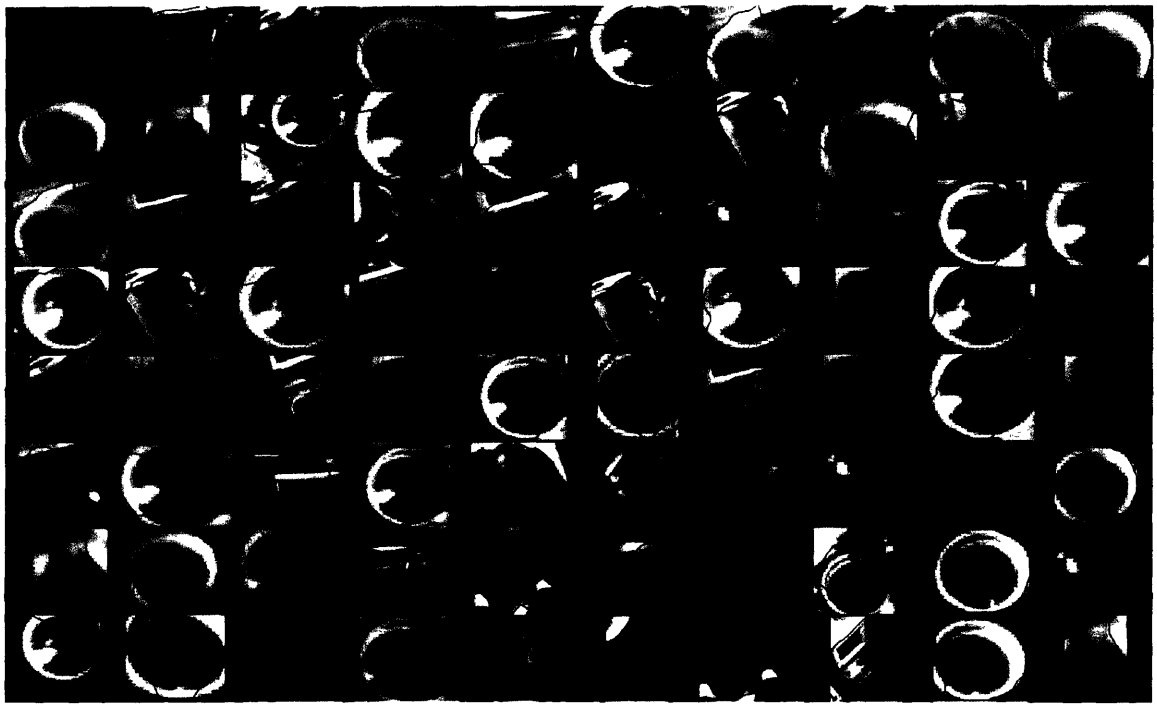


Figure 5-13: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. This cluster contains some manipulated objects, some hands, and parts of the background. Views of a cup make up the majority of the object related segments shown.

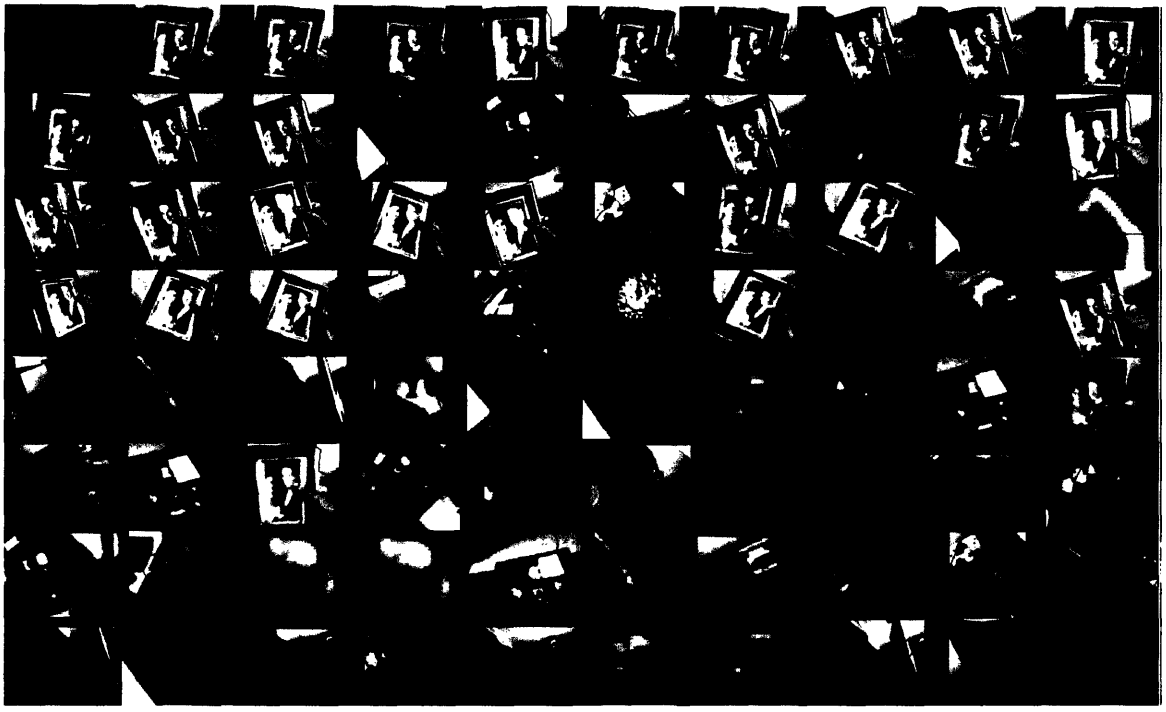


Figure 5-14: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. The top members mostly consist of background and manipulated objects, including a framed picture and a coke can.

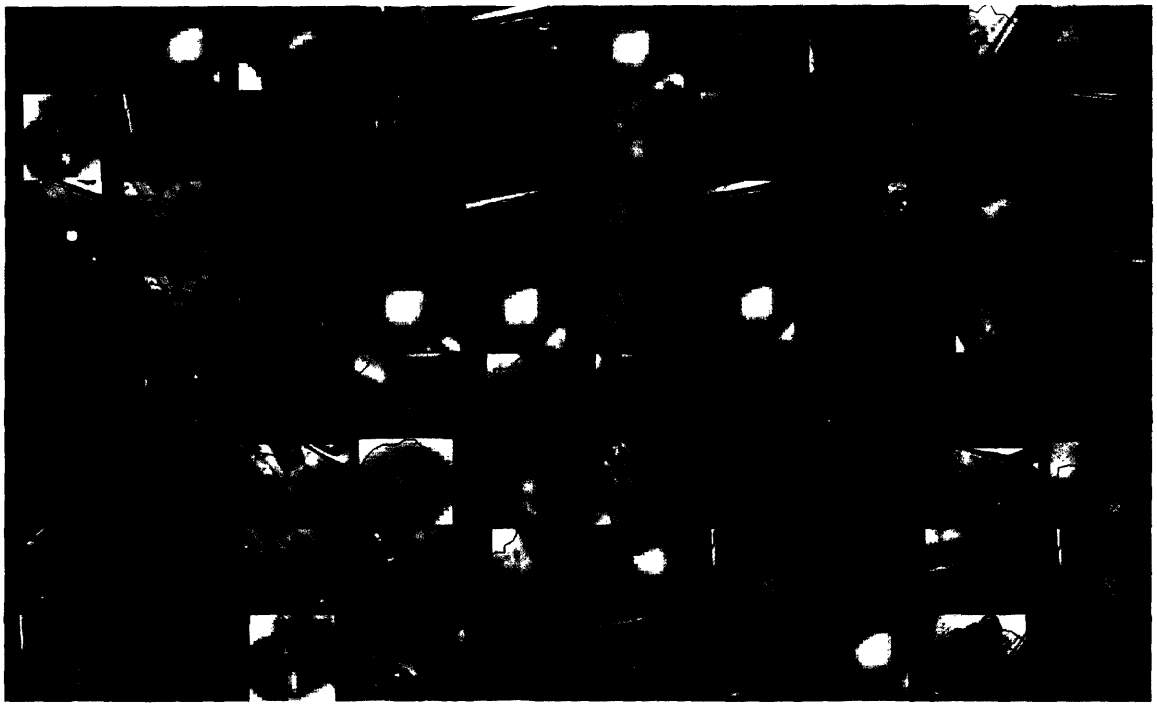


Figure 5-15: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. This cluster contains some manipulated objects, some hands, and parts of the background. The top segments include the back of a chair that was moved, a cup, a can, and a bowl.

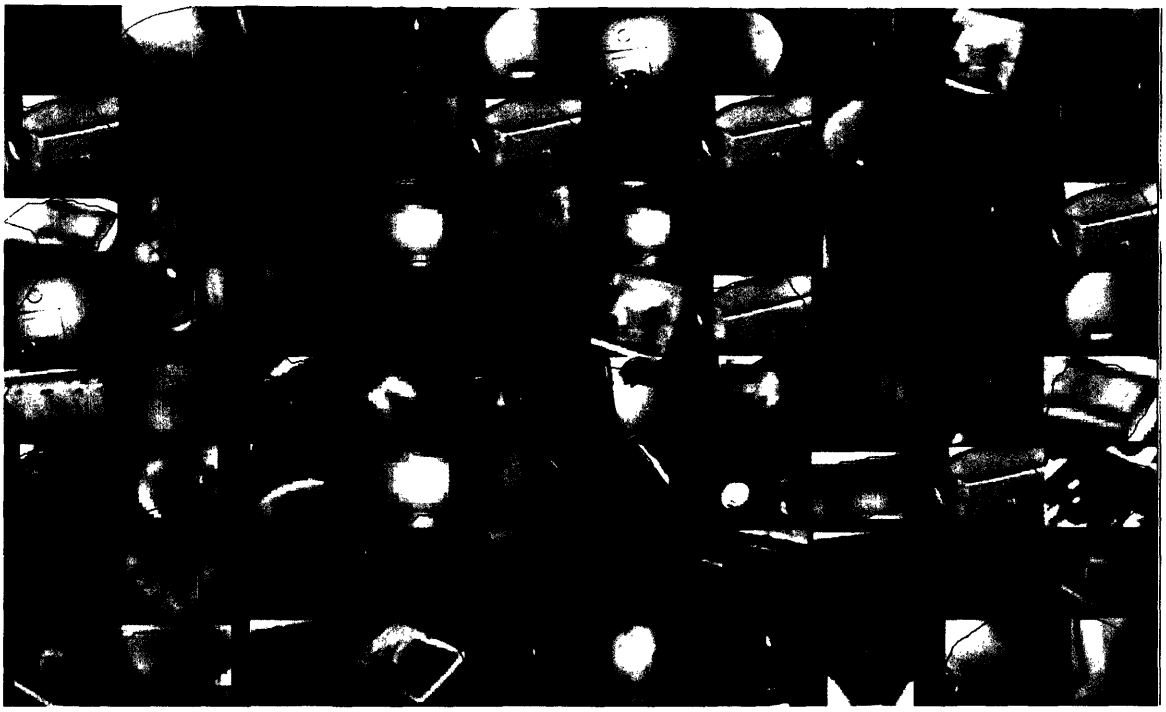


Figure 5-16: This figure shows the top 79 segments of another cluster in the same way as figure 5-7. The top segments mostly consist of background.

of this method is that it requires almost all of the desirable segments to have been found in advance, which can be a very time consuming process. The interest point operator and other elements of the attention system can mitigate this cost, but the total number of necessary segments per frame can still be prohibitive and lead to overnight computations on a cluster of machines in order to process less than an hour of video. Preliminary experiments indicate that the visual interest point operator and associated shape descriptors might be able to perform rapid tracking and filtering of salient elements of the video prior to computing visual segments, resulting in significant computational savings.

5.4 Towards Autonomous Machine Learning about Everyday Human Manipulation

The results of this chapter suggest promising directions for research into machines that autonomously learn about everyday human manipulation.

The clear structure of the specially selected hand positions indicates that many manip-

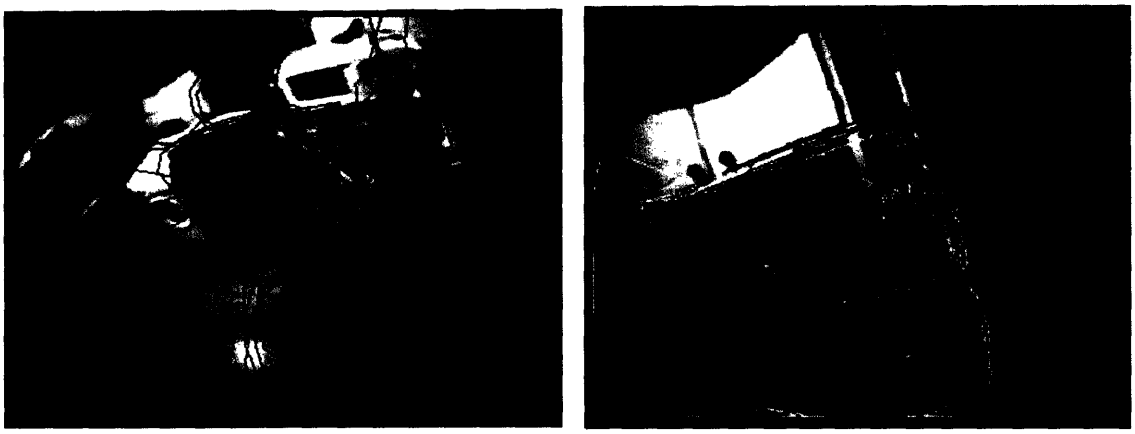


Figure 5-17: These two images show examples of the remaining segment paths after a five frame shortest paths computation. Polygons of the same color display the segments over time that form a particular shortest path.

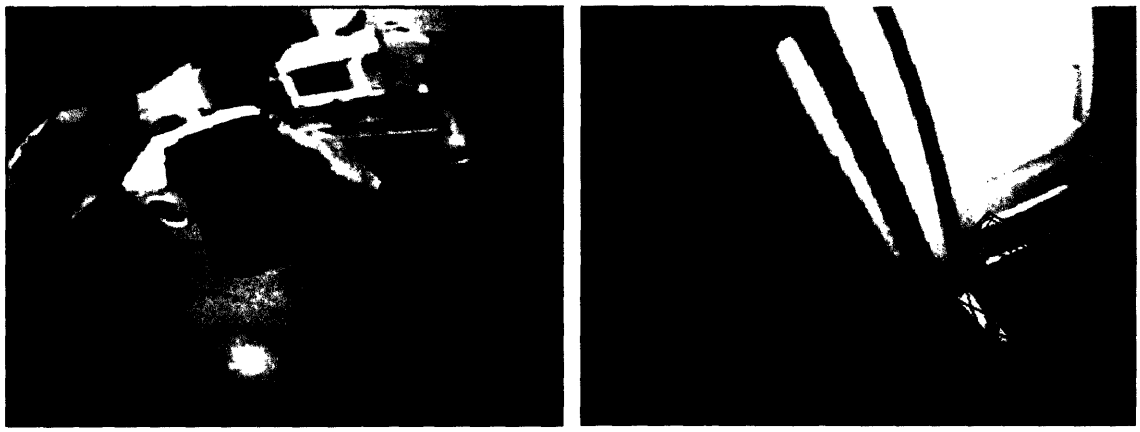


Figure 5-18: These two images show examples of the most salient segmentation path, which in these two cases is the arm. This is a common occurrence, probably because the arm is well modeled as a rigid 2D segment undergoing rotation, translation, and scale, along with its propensity to be in the foreground and occluding other segments during activity.

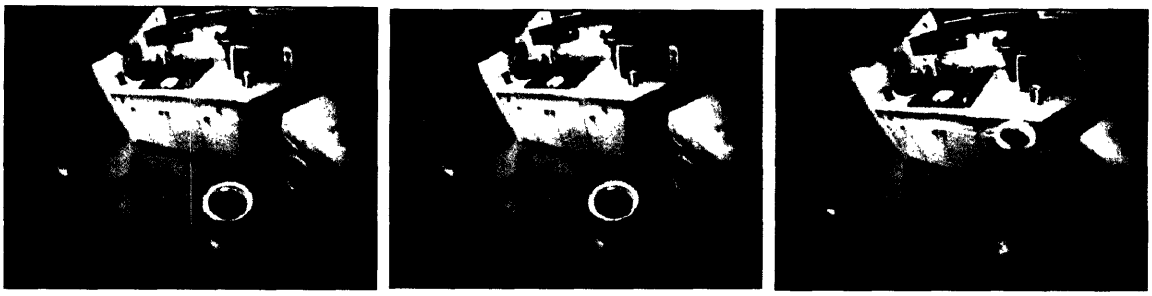


Figure 5-19: This sequence of images shows an example of using a specialized cost function that is biased to find the hand by decreasing the cost for segment paths that have some hand color and are spatially close to the kinematic prediction of the hand's location.

ulation activities could be well-modeled at a higher level of abstraction by modeling the transitions of the hand among the hand-states represented by these clusters. The statistics for these transitions could naturally be modeled with HMMs [13]. Like the hand-states themselves, the statistics for the transitions among the hand states could be easily collected autonomously. After modeling these transitions, transitions of a particular type could be selected and associated with features, such as representations of the hand's configuration, and the rotation of th wrist. We could search for distinct types of transitions, such as the difference between bringing a cup up to the head to drink, which would involve a particular type of wrist and hand location, and bringing a cup up to the head for visual inspection. Given the results of this thesis, research along these lines should be able to create machines that can autonomously associate distinctive kinematic actions.

The clear structure of the clusters of vision segments also suggests directions for future research. The first priority should be to autonomously select the clusters that represent the hand and then use them to autonomously train a visual hand detector. We have already done this with some success, but have chosen not to report the preliminary results in this thesis. Selecting the clusters that contain hands can be accomplished by ranking the clusters based on how close the member segments are to the kinematically predicted hand locations, and how well the sizes of the segments vary with the kinematically estimated depth of the hand. A variety of possible hand detectors are available. Our tests used the shape descriptors associated with our visual interest point operator, along with associated integral appearance features. Once the hand can be detected visually, the system could

combine this visual detector with the hand position predictions from the learned kinematic model to produce better estimates of the hands location throughout the data set. Amplified versions of these improved position estimates, or the unaltered improved position estimates, could then be used to train a better kinematic model. If more high quality hand data would be beneficial, the system could track the hand from moments of high confidence to generate additional high-quality hand position estimates.

Given a visual hand detector and better hand position estimates, the system could focus on two types of learning. The first would be to model the appearance of distinct configurations of the hand. The hand clusters contain images of the hand in a large number of distinct configurations, varying by grip, orientation, and such. Modeling these distinct configurations of the hand would help to distinguish among different manipulation activities and be informative about the object being manipulated. For example, a pincer grip is highly predictive of the location of part of the object being manipulated. The other focus would be on modeling the objects being manipulated. The clusters of visual segments from this chapter already indicate the high quality of object segments that can be produced by Duo's visual system. These could be further filtered by explicitly tracking the segments over time and monitoring their position with respect to the hand, as detected by the newly trained kinematic model and visual hand detector. After this further filtering, the objects should be clustered based on their member parts and appearance.

After performing these methods of autonomous learning to better model the hand's appearance, the appearance of manipulated objects, and associated kinematic activities, the system could begin to look for relationships between manipulative actions and the manipulated objects. Although a few research steps away, this opportunity for rich autonomous learning served as the original motivation for this thesis. Darnell Moore has already impressively shown, albeit under highly-constrained circumstances, that actions and objects can be mutually informative, [42]. By learning about the relationships between actions and objects, the system would have an opportunity to learn about the function of objects, which to some extent is a more fundamental view of an object's significance than its appearance. Since kinematic activity strongly relates to the abstract function of objects, it could be used as a proxy for this abstract function and directly associated with objects. For example, cups vary greatly in the details of their appearance, but the kinematic activity associated with drinking from a cup is highly stereotyped since it directly relates to the cup's fundamental

function of transporting liquids to the mouth, and moving the liquids into the mouth for drinking.

We are optimistic about the research path we outline here. For this thesis, we have built a strong foundation on which to build a machine that autonomously learns about the world by watching the activities of the wearer. The system is not mearlymerely a demo, but a fully functional hardware and software platform. We hope that future research will build upon this foundation. We intend to encourage this by releasing significant parts of our software as open source. With this is mind, we have been careful to minimize dependencies on non-open source software. The device drivers for the orientation sensors and the use of Matlab in camera calibration are the only non-open source dependencies for the Duo platform of which we are aware. The software side of the platform is already being used on work by Aaron Edsinger on the humanoid robot Domo [14].

Chapter 6

Machine Augmented Annotation of Captured Experience

The results from the previous section give some indication of how much machines might be able to autonomously learn by passively monitoring human manipulation. Within the next two chapters, we present two systems we have designed and implemented that attempt to enable Duo to cooperate with a human in order to better learn about everyday manipulation. In the next chapter, we will describe a system for real-time collaboration between Duo and the wearer. Within this chapter we describe machine augmented offline annotation and browsing tools that can help humans and machines collaboratively browse and annotate a database of captured experience.

Although real-time cooperative applications with the wearable, such as the one we describe within the next chapter, are appealing and highlight potentially useful ways for people to help machines learn, offline machine assisted annotation and browsing of captured kinematic and video data is more practical in the near term. By coupling fast machine perception and learning algorithms with interactive browsing and annotation tools, a flexible software architecture, carefully designed user interfaces, and a cluster of workstations, we can greatly enhance the ability of a person to help machines learn. Furthermore, this offline approach allows the wide variety of time scales of operation for various algorithms for perception and learning to be more seamlessly integrated together. Fast algorithms such as our kinematic and visual segmentation systems can help the person help the machine at interactive rates, while slower estimations such as the nonlinear estimation of the kinematic

model and discovery of the hand can take place in the background across the cluster, or even overnight.

6.1 System Architecture

Figure 6-1 shows the architecture for the system. Three relational databases form the core of the system. Each database is designed to hold particular types of data. The database of captured experiences is updated after an episode of data capture, but otherwise is read only and replicated across the cluster of machines. The database of results from learning holds learned structures that tend to apply to most of a capture episode or a series of capture episodes in the database, such as a learned kinematic model. The database of annotations holds annotations organized by time that directly refer to the contents of the capture database, such as visual segments and kinematic segments. Both the database of learned results and the annotation database typically reside on the root machine so that the user can access them through local tools at real-time interactive rates. The cluster machines also read and write to these two databases by way of two custom XML-RPC servers.

We use the open source embedded database SQLite3.0 for each of the three databases. Although not designed for heavy, large-scale, mission-critical transaction-based operation, this database is excellent for our purposes. First, the database is extremely fast, which allows us to query the databases at real-time rates from Python. So, for example, the Python based video browser is able to play video and overlayed annotations at frame rate by quickly submitting time-based SQL queries to the capture database and the annotation database. Second, the database is easy to replicate and manage since it exists as a single file in the application's directory. This allows us to easily backup particular annotation and learning databases, and switch out the current databases for previous ones. Finally, SQL provides a simple and uniform query interface for the software applications. The user on the root machine and the automated annotators across the cluster all update the databases in the same way, except that their entries are labeled as being from a machine or a human. SQL provides a good ability to search over the annotations, captured data, and learned data using a variety of criteria, and all databases are indexed by time.

A typical session of use starts by connecting Duo to the wired network after a capture session, copying the raw capture directories on Duo to the root machine, and then running

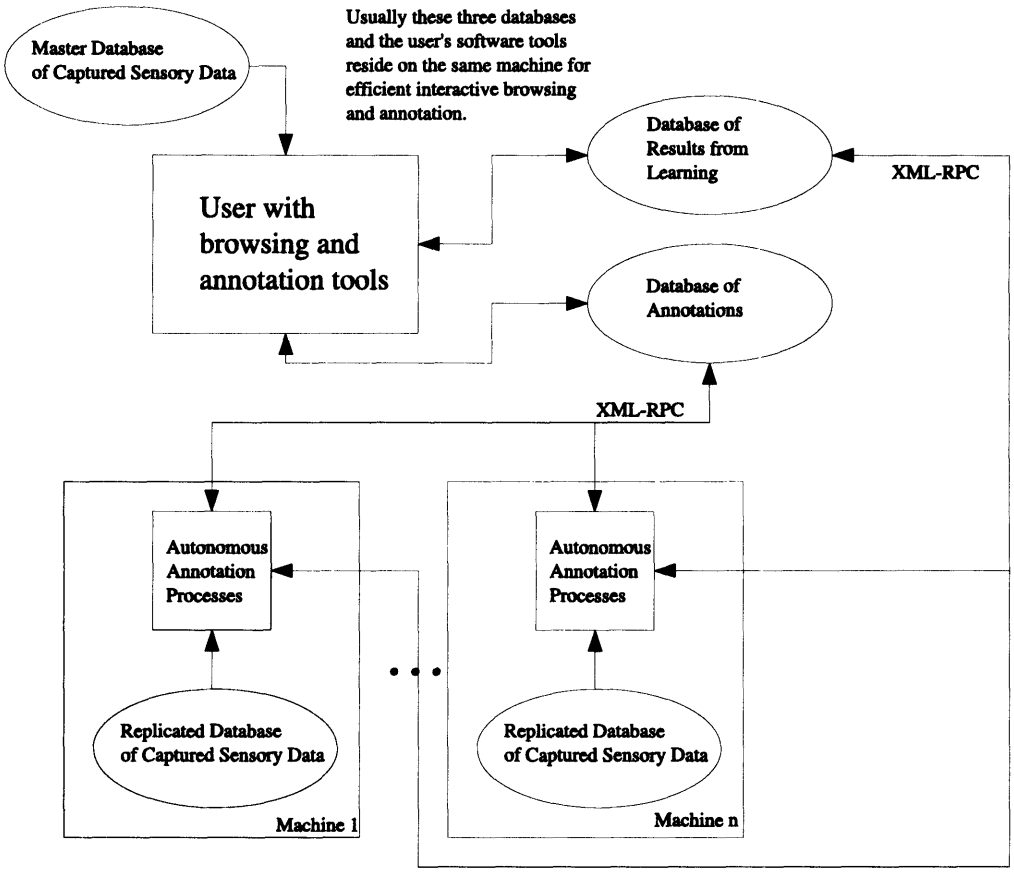


Figure 6-1: This diagram shows the architecture for Duo's semi-automated browsing and annotation system. The system uses three databases, one for captured data, one for learned results, such as the estimated kinematic model, and one for user and machine made annotations. The captured database is replicated across a cluster of machines for parallel processing of its contents. The other two databases are queried through either a local library or through two XML-RPC servers over the network. The Annotator software directly queries all three databases in real time at frame rates and higher to help the user help the machine annotate the captured data.

a script that automatically synchronizes the capture database with the currently available capture directories, and replicates the database across the cluster. Next a cluster wide script updates the learned database with the estimated kinematic models and the estimations on which they depend for all of the episodes in the capture database for which the estimations have not been made. Next a script uses the newly estimated kinematic models to automatically segment the kinematic data and then adds these segmentation times to the annotation database. Other autonomous processes, such as the collection of likely hand segments and manipulated object segments shown in chapter 5, also make use of the databases for input and as a repository for results. Consequently, the user can see these autonomously generated segments when browsing through the database.

6.2 A Machine Augmented Interface

The user can browse through the captured database using the machine generated kinematic segmentations to summarize the episode, see figure 6-3 for a view of the interface. This interface simplifies both browsing and annotation of the wearer's activities.

The user can also browse the episode as video with visualized kinematic data, annotating visual segments in the process, see figure 6-2. The video viewer, shown in figure 6-2, uses a custom display based on OpenGL [67] to seamlessly display images, 3D models, and annotations. Within the video view, the same visual segmentation algorithm used by Duo to segment video autonomously, facilitates annotation by allowing the user to select significant parts of the image for annotation with a single click of the mouse. By using the same segmentation method Duo can better make use of the segments, since if Duo were to select the same location, Duo would get the same segment as the person using the software. At any point the user can click on a part of the video, which will freeze the video and perform a log-polar segmentation at that location that the user can either discard or label. The interface is intuitive and fast to use, since clicking the segment automatically puts the window's focus on the annotation label box, hitting enter while in the label box inserts the current visual segment annotation into the database, and clicking the image prior to hitting enter discards the current visual segment and performs a new segment. The performance of the log-polar segmentation is excellent for this application, since it can usually be performed in less than 0.2 seconds, which results in no significant lag from the user's perspective. Also,

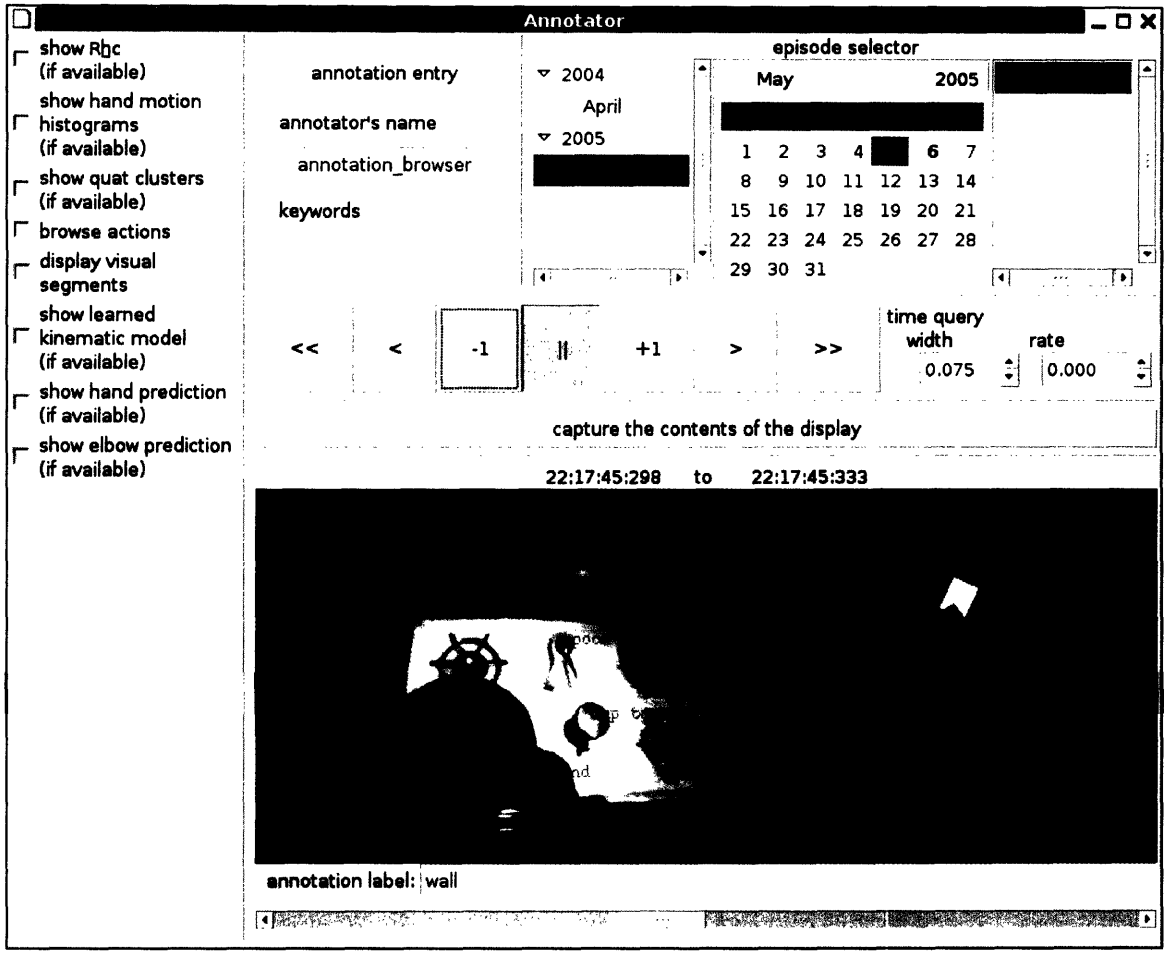


Figure 6-2: This is a screen shot of the Annotator annotation software in its video browsing mode. From here the user has access to all of the captured episodes within the database, which the user can select with the calendar based episode selector. Video updates are equivalent to SQL queries to the capture and annotation databases, and can be played at frame rate and above when the software is used on the root machine that holds the annotator database. As the video is playing, previously created visual segments from machine and human annotations are overlayed at frame rate with their labels. At any point the user can click on a part of the video, which will freeze the video and perform a log-polar segmentation that the user can either discard or label.

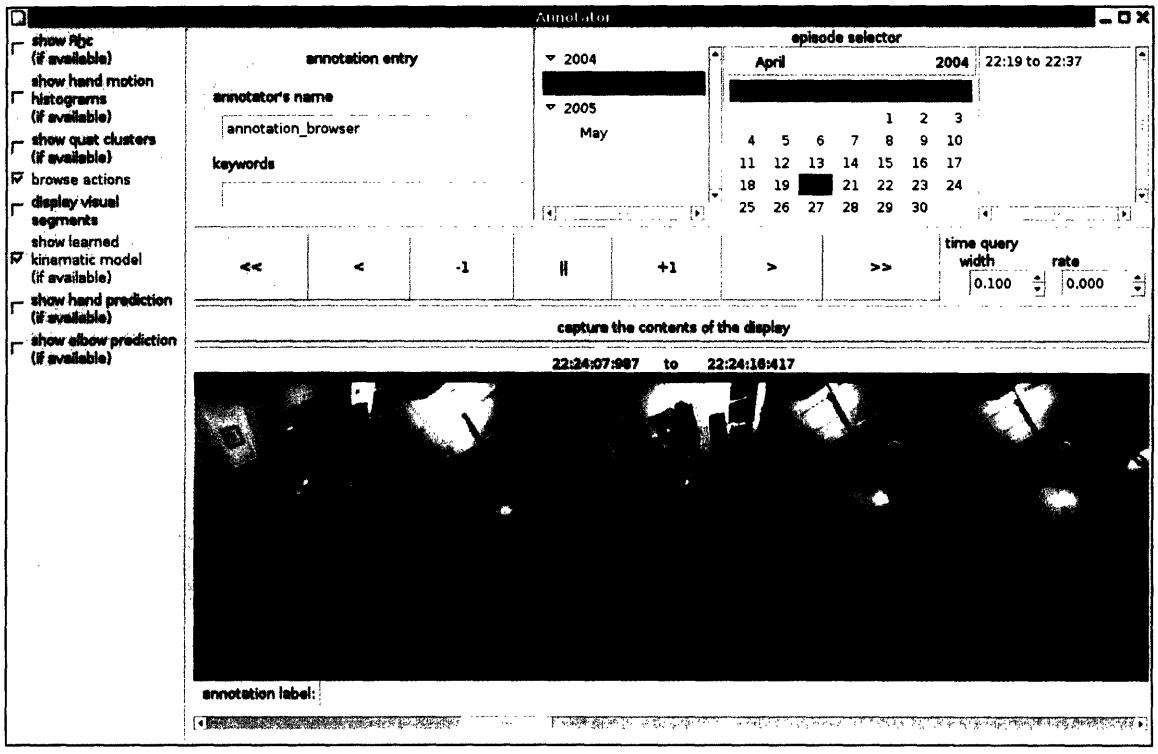


Figure 6-3: This figure shows a screen shot from the Annotator software in action browsing mode, which allows the user to efficiently move through the video based on the automatically segmented kinematic activity.

the log-polar segmentation is intuitive to work with since it relates to a form of natural center for objects which people are inclined to click. Each of the segmentations shown in the screenshot of figure 6-2 were made with a single click to a point within the object.

Chapter 7

A Real-Time Wearable System with Kinematic Sensing

Much of the research for this thesis was originally directed towards the eventual creation of a wearable creature that would both passively and actively learn from the wearer. We now believe that passive data capture and machine augmented annotation will be a more fruitful direction for research in the near term. However, wearable learning systems that actively ask the wearer for help do have some distinct advantages that are well worth considering. Within this section, we describe a demo application of such a real-time wearable system that elucidates some of these advantages.

7.1 The Application

The first and only real-time application we implemented with the Duo platform was also our very first project with Duo. For this work, we used the original backpack version of the Duo hardware with the original software system, which was written entirely in C++. We designed this application to acquire segmented images of hand-held objects manipulated by the wearer during everyday activities. In the terms of Daniel Dennett, this wearable creature was designed to be a form-a-vore that wished to learn about the objects people worked with during the day. This demonstration application was significant in the way kinematic sensing was used to enable tight integration between the behaviors of the wearable system and the wearer.

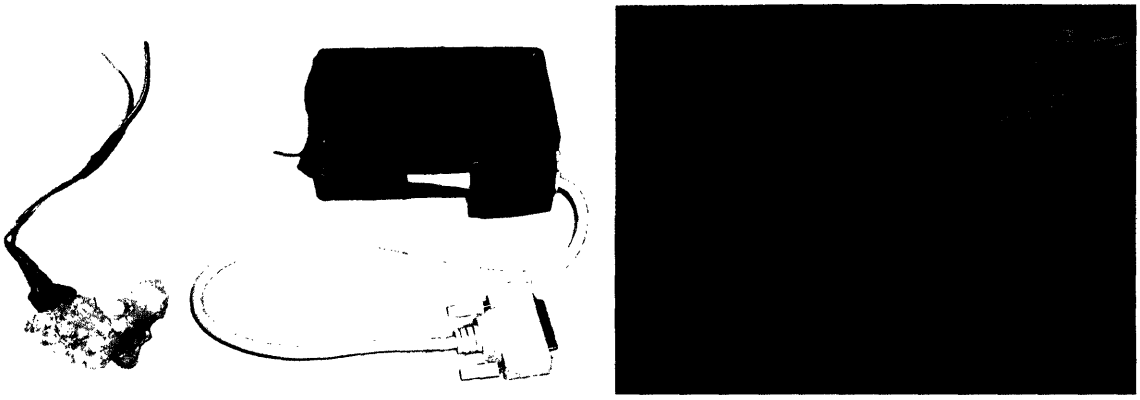


Figure 7-1: For a time, the white LED array shown on the left was wrapped around the bottom of camera. The LED array was used to illuminate objects held up to the camera, so that they could be easily segmented. The black box shown in the middle held two 9V batteries and custom circuitry on a bread board in order to power the LED array and allow the laptop to turn the LED array on and off through the parallel port. In the image on the right, the LED array is barely visible below the camera. Unfortunately, this is the only picture of the attached LED array we were able to locate. We removed the LED array in order to simplify the system and focus on segmentation algorithms that did not require active illumination.

7.1.1 The Behavior System

Duo monitored kinematic activity and attempted to detect movements that were likely to indicate that the wearer reached out into the world, grabbed an object, and brought it back near his body for comfortable manipulation. Upon detection of such a movement, Duo would verbally request that the wearer, "look at the object". After making the request, Duo would monitor kinematic activity in order to detect whether or not the wearer was holding the object up for close visual inspection. If Duo detected that the wearer was cooperating and that the wearer's hand was in the appropriate position, Duo would flash an LED array for every other captured image from the hat-mounted camera, so that the object could be easily segmented, see figures 7-1 and 7-2. While flashing the LED array, Duo also monitored the wearer's head movement. If the head movement of the wearer was too large, Duo would make the request to the wearer, "keep your head still". If the wearer cooperated, high-quality, segmented images of the object would be obtained.

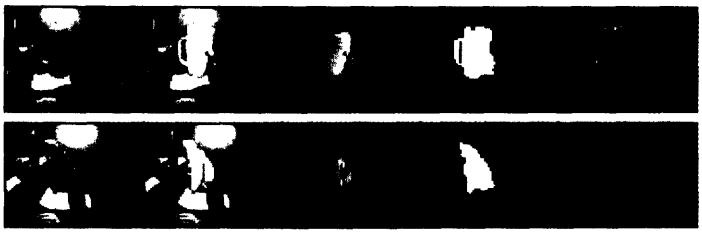


Figure 7-2: This figure shows two segmentations of common manipulable objects by Duo. When Duo detects that the wearer has reached for an object, Duo requests that the person look at the object via speech through the headphones. When the person holds up the object to look at it, Duo flashes the LEDs in order to produce the segmentations shown in this figure. The first column shows Duo's view before the LED flash and the second column shows the view during the LED flash. The third column shows the difference between the flashed and non-flashed images. The fourth column shows the mask produced by thresholding this difference. The final column shows the masks applied to the images to segment the hands holding the objects in the images.

7.1.2 Active Visual Segmentation

The array of white LEDs provided active illumination that clearly differentiated between foreground and background since illumination rapidly declines as a function of depth. By simply subtracting the illuminated and non-illuminated images from one another and applying a constant threshold, Duo was able to segment the object of interest and the hand. see figure 7-2.

By keeping his head still, a cooperative human would minimize the image motion, which improved the success of this simple segmentation algorithm and reduced the need for motion compensation prior to subtracting the images. The location at which the LED array was most effective sat about 25cm from the face, centered on the eyes. Humans also get a strong sense of depth around this location through stereopsis. The wearable could have feasibly used a stereo camera configuration to get a similar segmentation, but the computational cost and additional hardware complexity would not have been justified for this application. Also, less obtrusive infrared LEDs could have been substituted for the white LEDs, but debugging would have been more difficult and less feedback would have been provided to the human about optimal object placement and system activity. Others have used LEDs for

active segmentation, such as for the segmentation of walking hazards in a wearable system designed to help people with very poor vision [34].

7.1.3 Kinematic Detection

This demo employed three distinct kinematic detectors. The reach detector and object inspection detector both used an approximate kinematic model, while the head motion detector monitored the magnitude of change in the head's direction over time.

The reach detector used a hand-coded matched filter to detect when a person was likely to be grabbing a new object. The filter operated on measurements derived from the kinematic model and its estimated configuration based on the measured orientations from the human body. The filter was run on the results of projecting the estimated velocity of the hand, with respect to the world's coordinate system, onto a unit vector extending from the center of the torso to the previous position of the hand. The resulting measurement indicated the velocity at which the hand was moving toward or away from the center of the human's torso. The matched filter detected when the wrist moved away from the torso for an extended period at a relatively high velocity, slowed down to a stop, and then moved toward the torso at a relatively high velocity for an extended period. Specifically, a block filter derivative, equivalent to a single Harr wavelet, was convolved with the 1D signal representing the hand's distance from the center of the body over time. Time stamps were also used to preprocess the signal and produce a linearly interpolated signal with sampling points at a uniform frequency. The Harr wavelet is a form of multi-scale derivative, much like the methods for kinematic segmentation we described in chapter 4.

The object inspection detector would signal when the kinematic model indicated that the dominant hand was within a volume in front of the eyes of the wearer.

7.2 Properties of Real-time Wearable Systems

The demo application we have described, serves as a useful example that illustrates several properties of real-time wearable systems that use kinematic sensing.

7.2.1 High-level Control

The demonstration wearable only had coarse high-level influence over the wearer's body on which it depended. Yet, with a cooperative person, this control was sufficient to move interesting objects to a location ideal for segmentation. Requesting other types of high-level object-directed actions would also be feasible. In general, any strictly verbal command could conceivably be made by a wearable system, so the space of possible actions that could be performed by the wearer at the wearable's request is quite large. Kinematic sensing helps the wearable to both determine what request would be appropriate and measure how well the wearer responds to a request that involves body motion. This effectively lets the wearable share the body of the wearer, which makes the complete system not unlike a humanoid robot with coarse, high-level control.

7.2.2 Subsumption Architecture

The overall architecture used for this real-time application can be well-modeled as a subsumption architecture with the human serving as the lower layer of behaviors on which the wearable's behaviors were built. Figures 7-3 and 7-4 illustrate this perspective, which emphasizes the ways in which a kinematically perceptive wearable can benefit from the natural behaviors of the wearer.

7.2.3 Passively Monitor, Then Interrupt

This wearable's behavior followed a generally useful pattern, which is to monitor the kinematic activities of the wearer until something interesting occurs, and then interrupt the activity in order to actively learn about it. This pattern of behavior has several benefits. First, it allows the wearable to make a tradeoff between acquiring relatively unbiased samples from the everyday behavior of the wearer, and intentionally biasing the samples away from natural behavior in order to aid the wearable in its goals. Second, as emphasized throughout this thesis, kinematic data can be processed much more efficiently than visual data, so the system can monitor kinematic data in a low-powered sleep mode and wake-up upon an interesting activity to perform more intensive and power hungry processing, such as visual computation. This is an interesting variation on the methods of chapter 4 which were discussed in terms of the benefits of kinematic attention for offline processing. Rather

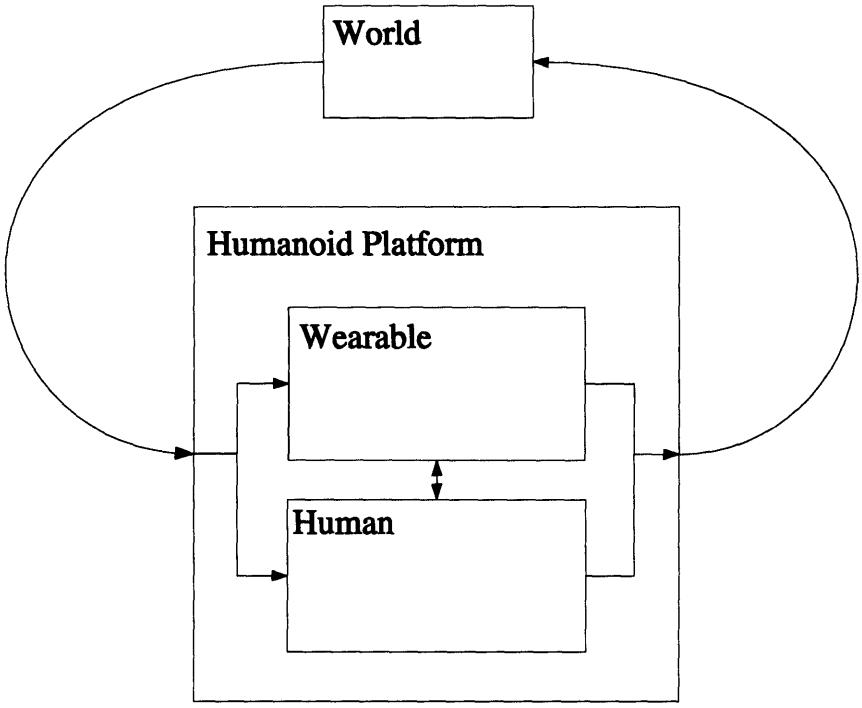


Figure 7-3: This diagram shows a high-level view of the real-time human/wearable platform, which can be well-modeled as using a subsumption architecture with the wearable's behaviors opportunistically built on top of the human's behaviors.

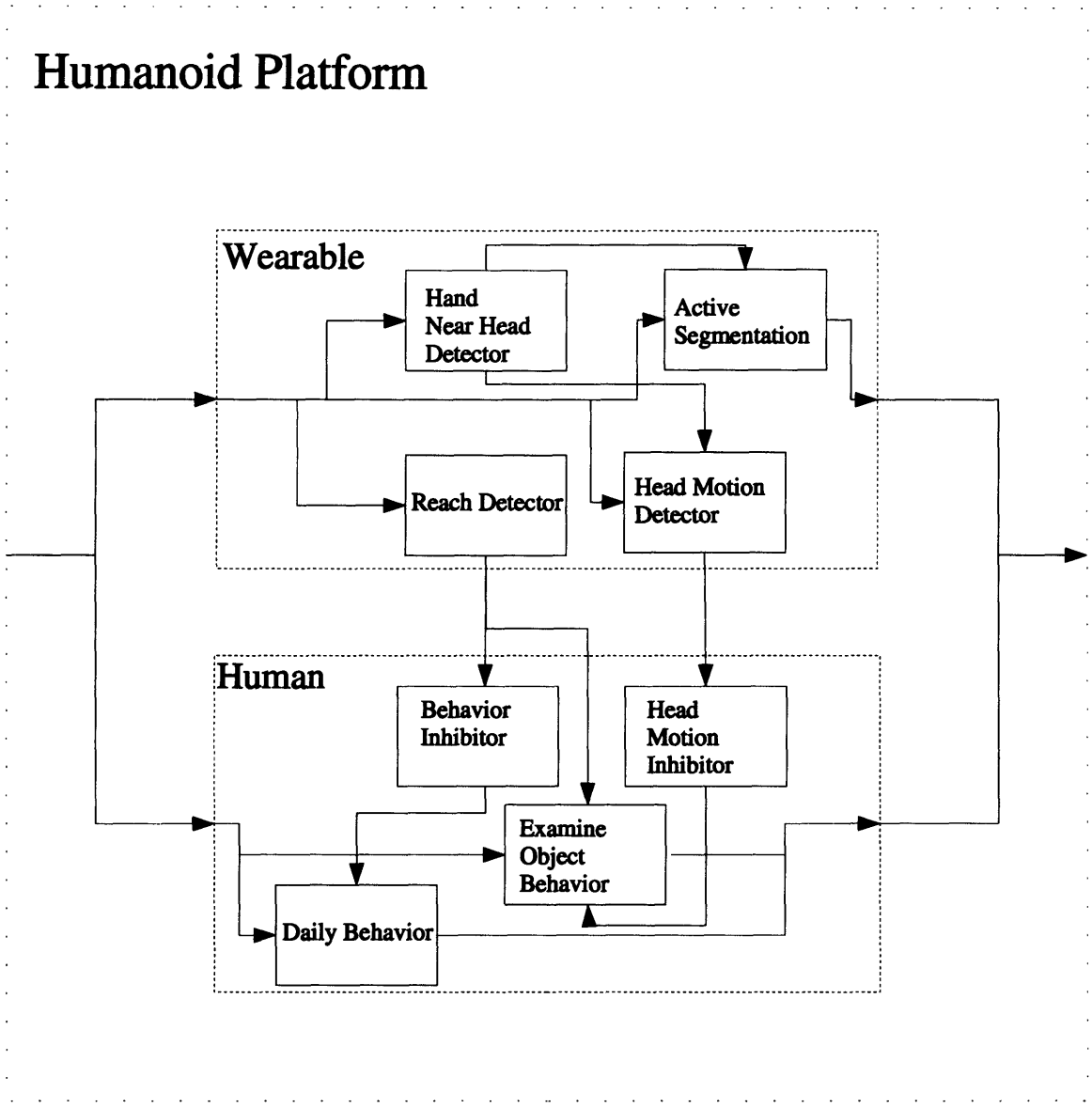


Figure 7-4: A detailed block diagram of the human/wearable platform. **Within the Wearable**: Reach Detector activates when the human reaches for an object. Hand Near Head Detector activates when the human's hand is close to his head. Head Motion Detector activates when there is head motion and the Hand Near Head detector is active. Active Segmentation synchronously flashes the LEDs and segments the illuminated foreground object from the background when the Hand Near Head Detector is active. **Within the Human (Conceptually)**: Behavior Inhibitor and Examine Object Behavior activate when the wearable requests to see the object better. Behavior Inhibitor inhibits Daily Behavior which was active. Examine Object Behavior brings the object close to the head for visual inspection, which triggers the wearable's Hand Near Head Detector. Head Motion Inhibitor activates when the wearable requests that the wearer keep his head still, and inhibits head motion associated with Examine Object Behavior.

than simply saving precious computation time while processing enormous databases, in the context of real-time systems these same methods can help save precious battery power and help to properly allocate very limited computational resources.

7.2.4 Shared Sensory Systems

An important aspect of wearable systems with respect to learning is the potential for wearables to capture all of the sensory experience and behavior of the wearer. A related issue that is particularly relevant to real-time wearable systems, is the extent to which the wearable shares the sensory systems of the wearer. In the real-time application we describe in this chapter, the wearable could be usefully described as having shared the kinematic and visual sensory systems of the wearer. The kinematic sensors took measurements that approximated the proprioceptive sensing of the wearer, providing an estimate of the kinematic configuration of the head, torso, and dominant arm, while the hat-mounted camera approximated the view from the wearer's eyes. Neither of Duo's modalities is close to the resolution and breadth of the human's senses, but they do provide coarse approximations.

In general, a shared sense is one for which the wearer's sense and wearable's sense provide approximately the same information. This sensory configuration provides both advantages and disadvantages. One advantage is that the wearable can more easily model the behavior of the wearer, since the wearable can directly monitor the information that influences the wearer's behavior. Similarly, if the wearer is going through a learning process, the wearable has some opportunity to tag along and learn in a similar way. Behaviors that help the wearer in a learning task may potentially be beneficial to the wearable as well. For example, when the wearer inspects an object, he brings it closer to his eyes, which both separates it from the environment in depth and allows for higher resolution imaging of the object. The real-time application described within this chapter was able to take advantage of both of these properties by using the LED array to segment the foreground by depth and by grabbing higher resolution images of the object. Additionally, a wearable could request that the wearer rotate the object, in order to view it from another perspective. Shared sensory systems can also make the problem of shared attention simpler for a wearable. Rather than attempting to infer the object of the wearer's attention based on observations from a third person perspective and then servoing sensory systems to observe the object, a wearable with shared sensors can simply let the wearer/caregiver focus attention on the object, which will

automatically focus the wearable's attention on the object,

The primary disadvantage of shared senses is that control over the senses is also shared. Sometimes the goals of the wearable and the wearer will not be aligned. The wearer might not look at what interests the wearable. The wearer might not interact with the world in the way the wearable would like. The wearable can encourage the user to inspect an interesting object and keep his head still, but the wearable is ultimately dependent on a helpful and responsive wearer. Wearables could mitigate this problem by combining shared senses with omni-senses that sense everything and actuated-senses that allow the wearable to direct its own senses. For example, the wearable could use multiple omni-directional cameras to see most of the world, and a pan-tilt camera mounted to the torso could look where the wearable wants to look.

Kinematic sensing helps mitigate these control problems by allowing the wearable to directly monitor the control choices made by the wearer. This allows the wearable to find moments at which the human's behaviors are aligned with the wearable's objectives, and to more easily determine the extent to which the human is responding to the requests of the wearable. Robust designs for this type of system require that the wearable be able to detect when the human is cooperating, since even a very helpful person will have moments during which he can not assist.

7.2.5 Looking for Special Moments

The previous two points fit into a larger pattern that involves Duo monitoring and observing over long periods of time to find situations for which the problem is easier. The real-time system can use the kinematic model to detect situations in which the senses are helpfully aligned or the context of the wearer's behavior fits with the interests of the wearable. Our offline system from earlier chapters also used this method to find moments in the database at which the wearer's hand and manipulated objects could be more easily segmented. Even if favorable situations are sparse, enough of them may be detected over the coarse of multi-day recording sessions to facilitate the process of learning.

	real-time wearable	offline wearable
interactivity	can exploit the wearer	reduced burden and corruption
annotation	context aware annotation	browse and annotate over all time
complexity	more processing, etc.	optimized for data capture
algorithmic	strict real-time	flexible
active sensing	possible	not possible

Figure 7-5: This table highlights some of the more significant differences between real-time and offline wearables for learning.

7.3 Real-time Versus Offline Learning

The real-time wearable system we describe in this chapter used kinematic sensing and active segmentation with an LED array to acquire segmented images of the wearer's hand and manipulated objects. The clear advantage is that with the user's help and some special sensory equipment this relatively simple wearable system was able to acquire the same type of information collected by the complex, autonomous, offline system we have described in the previous chapters. However, real-time wearable systems for learning also have disadvantages. The major disadvantages are that real-time wearable systems often require significant portable computation, additional perceptual hardware, and real-time perceptual algorithms, which greatly limits the algorithmic options for perception. They also require more rigorous and complicated coding. Furthermore, in order to take advantage of this real-time processing, the wearable must interrupt or otherwise influence the wearer's natural activities, thus increasing the burden on the wearer and altering the wearer's natural behaviors.

Table 7.3 highlights some of the properties that distinguish real-time and offline wearables for learning. Real-time wearables can interact with the wearer which may be advantageous, while strictly offline wearables are less likely to burden the wearer or corrupt his natural behavior, but do not have the option to have the wearer make things easier. With a real-time wearable, the user can annotate aspects of a situation while privy to the full context of the situation, while with offline processing, the person's view of the situation is limited to the captured data and the person's memory, who may not have even been the

wearer during the capture session. On the other hand, offline annotation can make use of helpful machine augmented tools for annotation and browsing, such as those described in the previous chapter. These tools can allow the person annotating the data to rapidly find interesting situations across the entire capture session with a comfortable user interface, while in a real-time wearable setting, browsing across previously captured sessions would be more challenging in terms of the interface and the computational requirements. Real-time wearables require more computation and more power for real-time perception than a wearable designed to strictly capture data for offline processing, which, as mentioned in chapter 2, can significantly impact the comfort and style of the wearable. Finally, real-time wearables can be used to actively sense the situation in a reactive way by influencing the wearer and using additional sensing hardware, while offline processing requires that all the sensing has already been performed. Of course, there is room for designs that sit in between these two extremes. For example, a wearable that is primarily designed for capture might make use of very simple processing to actively sense the environment at opportune times.

7.4 Summary

As discussed in chapter 1, many applications can benefit from kinematic sensing. Within this chapter, we looked at a specific, implemented, demonstration application that illuci-datedelucidated several aspects of real-time wearable design, particularly with respect to creating wearables that actively learn from the wearer. We then compared real-time wearables for learning to offline wearables for learning.

Chapter 8

Conclusions

In this thesis we presented Duo, the first wearable system to autonomously learn a kinematic model of the wearer via body-mounted absolute orientation sensors and a head-mounted camera. We demonstrated the significant benefits of endowing a wearable system with the ability to sense the kinematic configuration of the wearer's body. We showed that a kinematic model can be autonomously estimated offline from less than an hour of recorded video and orientation data from a wearer performing unconstrained, unscripted, household activities within a real, unaltered, home environment. We demonstrated that our system for autonomously estimating this kinematic model places very few constraints on the wearer's body, the placement of the sensors, and the appearance of the hand, which, for example, allows it to automatically discover a left-handed kinematic model for a left-handed wearer, and to automatically compensate for distinct camera mounts, and sensor configurations. Furthermore, we showed that this learned kinematic model efficiently and robustly predicts the location of the dominant hand within video from the head-mounted camera even in situations where vision-based hand detectors would be likely to fail. Additionally, we showed ways in which the learned kinematic model can facilitate highly efficient processing of large databases of first person experience. Finally, we showed that the kinematic model can efficiently direct visual processing, so as to acquire a large number of high quality segments of the wearer's hand and the objects the wearer manipulated.

Within the course of justifying these claims, we presented methods for estimating global image motion, segmenting foreground motion, segmenting manipulation events, finding and representing significant hand postures, segmenting visual regions, and detecting visual

points of interest. We also described our architecture and user-level application for machine augmented annotation and browsing of first person video and absolute orientations, as well as a real-time application that uses kinematic sensing.

Appendix A

Angular Perception

The results from the methods of chapter 3 allow us to very rapidly estimate the orientation of the camera within the world via the rotation matrix R_{hc}. Within this appendix, we briefly discuss two ways in which the camera's orientation in the world can be used to enhance perception. For both of these methods, we first compute our estimate for the orientation of the camera in world coordinates R_c using $R_c = R_h R_{hc}$ with our estimate for R_{hc} from chapter 3 and the current head orientation sensor output R_h. This operation has trivial computational complexity, so it is well suited to real-time applications.

A.1 Tagging the World by Viewing Angle

Given the world orientation of the camera, R_c, we might like to know to what extent this orientation of the camera in world coordinates influences the things that Duo sees in the world. Statistically, the wearer's head and hence Duo's camera occupy a small percentage of the volume of rooms and the world in general. The directions parallel to the ground plane tend to be free parameters, while the elevation of the head tends to be highly constrained. While active, people are usually sitting or standing and their head tends to be above the ground plane at corresponding heights. In addition, human environments are strongly structured with respect to the ground plane as defined by gravity. Given these strong limits on viewing position and the strong oriented structure of human environments, we would expect Duo to be able to find some strong relationships between the azimuthal angle of view and the things it sees. For example, the floor should be more prominent when looking down and ceilings should be more prominent when looking up.

In order to investigate these properties, we can tag each visual segment, or pixel, by it's elevation angle with respect to the camera's position. With our camera model an image pixel p with the coordinates (u, v) corresponds with the point $(x_c, y_c, z_c) = \frac{z_c}{f}(u, v, f)$ and hence the vector $v = \alpha(u, v, f)$ where α is an arbitrary scalar and $(0, 0, 0)$ is the position of the camera and its optical center. Assuming that the y axis in global coordinates is parallel to gravity and points upward, the elevation angle η associated with a pixel p can be determined by computing the normalized vector $\widehat{v}$

$$\widehat{v} = \frac{v}{\|v\|}$$

rotating it into world coordinates

$$\widehat{v}_w = R_c \widehat{v}$$

and computing the arcsinof its y component

$$\eta = \arcsin\left(-\widehat{v}_{w_y}\right)$$

where $-\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2}$ with $\eta = -\frac{\pi}{2}$ pointing straight down with gravity, $\eta = \frac{\pi}{2}$ pointing straight up against gravity, and $\eta = 0$ pointing perpendicular to gravity and hence horizontally. Tagging visual stimuli with η, such as visual segments, can provide a powerful feature with which to interpret the world and better detect significant components of the environment, such as the floor and ceiling.

A.2 Making Use of Projected Angle as a Feature

As illustrated in chapter 3, our rapid estimates of the camera's orientation in the world can be further used to enhance perception by relating edge directions to meaningful vanishing points. Any known orientation in world coordinates, such as the direction of gravity or the direction of the wearer's forearm, corresponds with a vanishing point in the images captured by the camera. Given the direction vector, we can find this vanishing point and then weight edges by how well they point to the vanishing point. It's important to note that within real images captured from a head-mounted camera by a person performing unconstrained daily activities, gravity aligned edges will often be distinct from strictly vertical edges in the images, so that knowledge of the camera's orientation within the world is important.

These edges, weighted by the extent to which they correspond with a given vanishing point, could serve as a feature in and of themselves, since, for example, many objects have sides that are parallel to gravity, and edges parallel to the direction of the forearm are more likely to correspond with the forearm's borders. By projecting the direction of the major axis of the forearm onto the image, we could estimate what the dominant forearm direction should be within the image and how it should be rotating from frame to frame. These estimates could then be used to weight edges for attention. Another possible use for this information, would be to quickly search for surfaces on which objects are placed. The system could look for locations in the image at which many short, gravity aligned, parallel edges, terminate, which would tend to correspond with the bases of objects on a surface. Likewise, some classes of objects have strong properties relative to gravity, such as bookshelves, which often have many gravity aligned edges over a small area. Additionally, we could define local coordinate systems at each point in the image based on the direction of gravity at the image position, which could help with object detection and recognition by constraining the likely orientations of the stimuli in the image, since many objects, such as a tree, tend to be found in the world with canonical orientations with respect to gravity. This contextual information about the world coordinate system with respect to gravity at an image location could be powerful, and is easy to estimate with our system.

We would like to use R_c to estimate the projected image angles of known angles in the environment including the direction of gravity, represented by the unit vector d_g, and the dominant direction of the forearm, represented by the unit vector d_f, which can be estimated from the forearm orientation sensor output R_f. The projection of points from the world into the camera are well modeled by the equation $X_c = [R_c^T | -t_{c2}]X_w$. We position the origin of the world's coordinate system at the camera's optical center, so that the world frame and camera frame only differ by the camera's rotation with $X_c = R_c^T X_w$, where affine coordinates are unnecessary. Assuming the direction d is not parallel to the image plane, we find the point (u, v) at which a vector from the origin with direction d and length α would intersect the image plane, which corresponds with the vanishing point we desire. Solving for (u, v) in

$$\begin{bmatrix} u \\ v \\ f \end{bmatrix} = R_c^T \alpha d = \alpha \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} d$$

gives

$$\alpha = \frac{f}{r_3 d}$$

and

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{r_3 d} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} d$$

The direction, ϕ, of the projection of another vector parallel to d that passes through point (u_2, v_2) on the image plane is equal to the direction of the ray from point (u_2, v_2) to (u, v), which is defined everywhere on the image plane except (u, v).

$$\phi\left(u_2, v_2\right) = \arctan2\left(v - v_2, u - u_2\right) = \arctan2\left(\frac{f}{r_3 d} r_2 d - v_2, \frac{f}{r_3 d} r_1 d - u_2\right)$$

This is clear from the geometry, since (u, v) is the vanishing point on the image for all the rays parallel to d, which we can easily confirm. We start with the following equation that shows the projective relationship between a vector $\alpha d + m$ and the resulting image point (u_3, v_3) as a function of α, where m is a constant three dimensional vector describing a point in space through which the vector passes when $\alpha = 0$.

$$\frac{z}{f} \begin{bmatrix} u_3 \\ v_3 \\ f \end{bmatrix} = R_c^T(\alpha d + m)$$

This can be written as

$$\begin{bmatrix} u_3 \\ v_3 \\ f \end{bmatrix} = \frac{f R_c^T(\alpha d + m)}{\alpha r_3 d}$$

which we can use to confirm that (u, v) is the limiting projection point as α, and consequently the length of the vector, goes to ∞.

$$\lim_{\alpha \to \infty} \frac{f R_c^T(\alpha d + m)}{\alpha r_3 d} = \frac{f R_c^T d}{r_3 d} = \begin{bmatrix} u \\ v \\ f \end{bmatrix}$$

If $r_3 d = 0$, then the direction d is parallel to the image plane and the projected directions are all parallel lines. In this case, we simply need to find the direction between the origin and the rotated vector d, without worrying about projection or intersections. So,

$$\begin{bmatrix} u_4 \\ v_4 \\ 0 \end{bmatrix} = R_c^T d$$

where the image vector (u_4, v_4) specifies the projected direction at all positions when $r_3 d = 0$.

$$\phi_{parallel}(u_2, v_2) = \arctan2(v_4, u_4) = \arctan2(r_2 d, r_1 d)$$

Bibliography

[1] Padmanashan Anandan. *Measuring Visual Motion From Image Sequences.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1987.

[2] Gordon C. Baylis and Jon Driver. Shape-coding in it cells generalizes over contrast and mirror reversal, but not figure-ground reversal. *Nature Neuroscience*, 4(9), September 2001.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, 2000.

[4] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, April 2002.

[5] Alexander C. Berg, Tamara L. Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondence. In *CVPR*, 2005.

[6] Bobby Bodenheimer, Chuck Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. In *Eurographics CAS'97*, 1997.

[7] Armin Bruderlin and Lance Williams. Motion signal processing. *Computer Graphics*, 29(Annual Conference Series):97–104, 1995.

[8] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference on Visual Information Systems.* Springer, 1999.

[9] Brian Clarkson, Alex Pentland, and Kenji Mase. Recognizing user context via wearable sensors. In *Proc. of the Fourth Intl. Symposium on Wearable Computers*, pages 69–76, 2002.

[10] Brian Patrick Clarkson. *Life Patterns: structure from wearable sensors.* PhD thesis, MIT Media Laboratory, September 2002.

[11] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, 1992.

[12] Intel Corporation. Open source computer vision library: Reference manual, 2001.

[13] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley and Sons, New York, 2001.

[14] Aaron Edsinger-Gonzales and Jeff Weber. Domo: A force sensing humanoid robot for manipulation research. In *Proceedings of the IEEE/RSJ International Conference on Humanoid Robotics*, 2004.

[15] Alexei A. Efros, Alexander C. Berg, Greg Mori, and Jitendra Malik. Recognizing action at a distance. In *IEEE International Conference on Computer Vision*, Nice, France, October 2003.

[16] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR: Workshop on Generative-Model Based Vision*, 2004.

[17] D. A. Forsyth and Jean Ponce. *Computer Vision: a modern approach.* Prentice Hall, 2002.

[18] E. Freeman and D. Gelernter. Lifestreams: A storage model for personal data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(1):80, 1996.

[19] Doug Gage. Ipto proposal for research on lifelog, 2003.

[20] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. Mylifebits: fulfilling the memex vision, 2002.

[21] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: Assembling run-time animations. *Proceedings of the Symposium on Interactive 3D Graphics*, pages 181 – 188, 2003.

[22] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, 3rd edition.* Johns Hopkins University Press, 1996.

[23] M.S. Graziano. Where is my arm? the relative role of vision and proprioception in the neuronal representation of limb position. *Proceedings of the Nationanl Academies of Science*, 96:10418–10421, 1999.

[24] Daniela Hall, Bastian Leibe, and Bernt Schiele. Saliency of interest points under scale changes. In *Proceedings of the British Machine Vision Conference*, Cardiff, UK, September 2002.

[25] Alexander Hauptmann and Wei-Hao Lin. Informedia at PDMC, July 2004.

[26] S. Hsu, P. Anandan, and S. Peleg. Accurate computation of optical flow by using layered motion representations. In *Proceedings of ICPR*, pages 743–746, Jerusalem, Israel, 1994.

[27] Odest Chadwicke Jenkins and Maja J. Mataric. Performance-derived behavior vocabularies: Data-driven acquisition of skills from motion. *International Journal of Humanoid Robotics*, 1(2):237–288, 2004.

[28] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[29] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell, editors. *Principles of Neural Science.* McGraw-Hill, 2000.

[30] Tatsuyuki Kawamura, Yasuyuki Kono, and Msatsugu Kidode. Wearable interfaces for a video diary: Towards memory retrieval, exchange, and transportation. In *Proceedings of the International Symposium on Wearable Computers 2002*, 2002.

[31] N. Kern, H. Junker, P. Lukowicz, B. Schiele, and G. Tröster. Wearable sensing to annotate meeting recordings. *Personal and Ubiquitous Computing*, 2003.

[32] Nicky Kern and Bernt Schiele. Context-aware notification for wearable computing. In *Proceedings of the 7th International Symposium on Wearable Computing*, pages 223–230, New York, USA, October 2003.

[33] James Landay, Mark Newman, and Jason Hong. The shadow: An experience capture system, 2001.

[34] C. M. Lee, K. E. Schroder, and E. J. Seibel. Efficient image segmentation of walking hazards using IR illumination in wearable low vision aids. In *Proc. of the Sixth Intl. Symposium on Wearable Computers*, pages 127–128, 2002.

[35] Tony Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, 1998.

[36] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[37] David G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision, Corfu, Greece*, pages 1150–1157, September 1999.

[38] David G. Lowe. Towards a computational model for object recognition in it cortex. In *First IEEE International Workshop on Biologically Motivated Computer Vision, Seoul, Korea*, pages 20–31, May 2000.

[39] J. Malik, D. Martin, C. Fowlkes, and D. Tal. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Technical report, Computer Science Division, University of California at Berkeley, January 2001.

[40] Steve Mann, Daniel Chen, and Sam Sadeghi. Hi-cam: Intelligent biofeedback processing. In *ISWC '01: Proceedings of the 5th IEEE International Symposium on Wearable Computers*, page 178, 2001.

[41] T.B. Moeslund and L. Norgaard. A brief overview of hand gestures used in wearable human computer interfaces. Technical report, Laboratory of Computer Vision and Media Technology, Aalborg University, Denmark, 2003.

[42] Darnell Janssen Moore. *Vision-Based Recognition of Actions Using Context.* PhD thesis, Georgia Institute of Technology, April 2000.

[43] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *ECCV02: deformation model and video sequences*, 2002.

[44] C.S.R. Taylor M.S. Graziano, D.F. Cooke and T. Moore. Distribution of hand location in monkeys during spontaneous behavior. *Experimental Brain Research*, 155:30–36, 2004.

[45] D.F. Cooke M.S. Graziano, C.S.R. Taylor and T. Moore. The cortical control of movement revisited. *Neuron*, 36:349–362, 2002.

[46] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001.

[47] J. F. O'Brien, B. E. Bodenheimer, G. J. Brostow, and J. K. Hodgins. Automatic joint parameter estimation from magnetic motion capture data. In *Proceedings of Graphics Interface*, pages 53–60, Montreal, Quebec, Canada, May 2000.

[48] Jennifer J. Ockerman and Amy R. Pritchett. Preliminary investigation of wearable computers for task guidance in aircraft inspection. In *Proceedings of the 2nd International Symposium on Wearable Computers 1998*, October 1998.

[49] J.M. Odobez and P. Bouthemy. Robust multiresolution estimation of parametric motion models. *International Journal of Vision Communication and Image Representation*, 6(4):348–365, 1995.

[50] Richard Alan Peters II, Christina L. Campbell, William J. Bluethmann, and Eric Huber. Robonaut task learning through teleoperation. *Proceedings - IEEE International Conference on Robotics and Automation*, 2:2806 – 2811, 2003.

[51] D. L. Pham, C. Xu, and J. L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–338, 2000.

[52] Nava Rubin. Figure and ground in the brain. *Nature Neuroscience*, 4(9), September 2001.

[53] Feiner S., MacIntyre B., and Seligmann D. Knowledge-based augmented reality. *Communications of the ACM*, 36(7):52–62, July 1993.

[54] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of CHI'99*. ACM Press, May 1999.

[55] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[56] A. Singh and P. Allen. Image-flow computation: an estimation-theoretic framework and a unified perspective. *CVGIP: Image Understanding*, 56:152–177, 1992.

[57] Thad Starner and Alex Pentland. Real-time american sign language recognition using desk and wearable computer based video. *PAMI*, 20(12):1371–1375, 1998.

[58] C. Stiller and J. Konrad. Estimating motion in image sequences, a tutorial on modeling and computation of 2d motion. *IEEE Signal Process. Mag., vol. 16, pp. 70–91*, 1999.

[59] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, 1986.

[60] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[61] Paul Viola and Michael Jones. Robust real-time object detection. In *2nd intl workshop on statistical and computational theories of vision*, June 2001.

[62] Howard Wactlar, Mike Christel, Alex Hauptmann, and Yihong Gong. Informedia experience-on-demand: Capturing, integrating and communicating experiences across people, time and space. *ACM Computing Surveys*, 31(9), June 1999.

[63] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers. Technical report, M.I.T. Media Laboratory, 1994.

[64] Markus Weber, Max Welling, and Pietro Perona. Unsupervised learning of models for recognition. In *Proc.6th Europ. Conf. Comp. Vis., ECCV2000, Dublin*, 2000.

[65] Y. Weiss and E. Adelson. A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models, 1996.

[66] T. Wiegand, E. Steinbach, and B. Girod. Affine multipicture motion-compensated prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(2):197–209, Feb 2005.

[67] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide.* Addison-Wesley, 1997.