TRACKING THE HUMAN TORSO IN MONOCULAR VIDEO

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Micah Ethan Metsger, B.S.

* * * * *

The Ohio State University

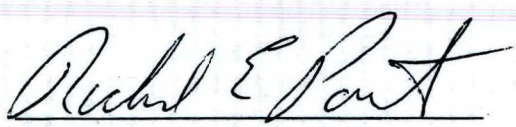
2005

Master's Examination Committee:

Dr. Richard E. Parent, Advisor

Dr. Raghu Machiraju

Approved by



Advisor

Graduate Program in Computer and

Information Science

ABSTRACT

We present a means of tracking the human torso through a monocular video sequence by leveraging the COCONE curve-reconstruction and segmentation algorithm. This solution uses geometry and topology to perform tracking instead of an *a priori* learned model, which permits a reasonable level of generality. Implementation of the algorithm is straightforward and tracking is automatic, requiring little to no user input other than a sequence of images.

This solution is unique in that tracking is conditioned on the input sequence instead of attempting to fit the sequence to a given model. Since the model is amorphous, a more general result may be derived. This more closely mimics the human visual system, which appears to track without regard to the semantics of the tracking problem.

For my grandparents, whose graduations have filled me with hope.

ACKNOWLEDGMENTS

In the course of writing this thesis, I incurred debt from many people. My advisor, Dr Richard Parent, provided patient input and timely observations. Dr Raghu Machiraju reviewed this thesis and provided valuable insight in several courses. Dr Tamal Dey graciously provided Dr Samrat Goswami's implementation of the Flow Discretization segmentation algorithm. Yisheng Chen provided several data sets and other resources that made it possible to test my work.

Dr Paolo Bucci was a great boss for two years and really made working as a TA enjoyable.

The support of my family and parents, Scott and Lucy, in particular has been tremendous. My sister Carin and girlfriend Ericka Samuels both proofread the manuscript and provided feedback on certain sections.

When the editorial "we" is used in this thesis, I trust the reader will understand I include those mentioned here, without whose contributions these pages would be surely blank.

VITA

August 7, 1981 ..... Born – Columbus, OH

June 14, 1999 ..... Diploma, Downingtown Senior High School, Downingtown, PA

March 20, 2003 ..... B.S., Computer and Information Science, The Ohio State University

September 2003-present ..... Graduate Teaching Associate, The Ohio State University

PUBLICATIONS

1. E. Metsger and K. Miles, *Everyday Linux*. Prentice Hall, Upper Saddle River, New Jersey, USA. 2001.

FIELDS OF STUDY

Major Field: Computer Science

Concentration: Computer Graphics

Contents

ABSTRACT ii
ACKNOWLEDGMENTS iv
VITA v
Table of Contents vi
List of Figures ix

1 INTRODUCTION **1**
1.1 Motion Tracking 1
1.2 Applications of Motion Trackers 2
1.3 Limits of Motion Trackers 3
1.4 Contributions of This Work 4
1.5 Torso Tracking 4

2 RELATED WORK **7**
2.1 Particle Filtering 7
2.2 CONDENSATION 9
2.3 Image-space Solutions 10
2.3.1 Bharantanatyam Dance 10
2.3.2 Camera Fitting 11
2.3.3 Linear Subspaces 11
2.3.4 Physical Constraints 12

3 TORSO TRACKING **14**
3.1 Algorithm Features 14

3.2 Motivation 15
3.3 Human Figure Model 15
3.4 Assumptions Made 16
3.5 Algorithm 16
3.5.1 Pre-Processing 17
3.5.2 Figure Segmentation 18
3.5.3 Torso Identification 22

4 IMPLEMENTATION **26**
4.1 System Specifications 26
4.2 Supporting Software 26
4.2.1 Programming Language 26
4.2.2 Flow Discretization Implementation 26
4.3 Sequences Tested 27
4.4 Algorithm Specifics 29
4.4.1 Edge Detection 29
4.4.2 Segmentation 33
4.4.3 Torso Identification 33
4.5 Default Settings 35
4.6 Problems Encountered 35
4.6.1 In Edge Detection 35
4.6.2 In Segmentation 36
4.6.3 In Identification 36

5 DISCUSSION OF RESULTS **37**
5.1 Concerning Objectives 37
5.1.1 Automatic Tracking 37
5.1.2 Limited Optical Cues 38
5.1.3 Robustness to Noise 39
5.1.4 Speed 39
5.2 Memory Complexity 40
5.3 Example Output 41

6 CONCLUSION **49**

6.1 Full Body Tracking 49

6.1.1 Location Confidence 50

6.1.2 Segment Integration 50

6.1.3 Initialization 51

6.1.4 Multiple Segment Tracking 51

6.2 Real-Time Location 52

6.3 Tracking Multiple Figures 52

6.3.1 Figure Classification 53

6.3.2 Figure Occlusion 53

6.4 Multiple Movement Types 53

6.5 Summary 53

Bibliography **55**

List of Figures

1.1 Various problematic inputs. 5

3.1 Algorithm Flowchart. 17

3.2 Flow Discretization segmentation examples. 21

4.1 Edge detector outputs. 28

4.3 Occlusion response. 31

4.2 An example strutural element for erosion. 31

4.4 Morphological operations. 32

4.5 Hausdorff Distance Algorithm 33

4.6 Hausdorff Distance using Distance Transform 34

5.1 Example stages (1). 42

5.2 Example stages (2). 43

5.3 Example stages (3). 44

5.4 Change of depth effect. 45

5.5 Breakdancing. 46

5.6 Hopping. 47

5.7 Hopping (2). 48

Chapter 1

INTRODUCTION

1.1 Motion Tracking

Motion tracking is the process by which a computer analyzes input video and produces a description of the movement of interesting features in that video. What constitutes an interesting feature is defined before analysis, which requires models of the feature and how it moves.

We use the term *motion description* to encapsulate various outputs from a motion tracker. Visual output (*e.g.*, a video with pertinent features highlighted by bounding boxes) as well as other forms of data (*e.g.*, an ordered list of joint positions for each frame) are included. In complex scenarios, the description may include surface deformations (due, for example, to muscle movement or contraction).

Low-level motion descriptions label pixels (*e.g.*, as "tracked" or "not tracked") whereas *high-level* descriptions classify pixels according to feature groups (*e.g.*, as "arm" or "head"). A high-level description, therefore, associates semantic meaning with each tracked pixel. These descriptions generally correspond to their models: low-level models are usually general with less semantic meaning; high level models are usually specific with a great deal of meaning.

This is important because the desired output description constrains the models used in analysis. To obtain more meaningful output, we often must assume characteristics of the input that compromise the generality of the tracker. For example, if we desire output in the form of a human skeleton, the *a priori* assumption is that we are tracking a human. One of the objectives of this work is to demonstrate how low-level models may produce high-level descriptions. This reduces the need for *a priori* assumptions and helps to generalize the tracker.

Motion trackers typically address two related problems: *object recognition* and *object tracking*. In object recognition, the feature model is matched to the current frame of video. Object tracking leverages the motion model to produce candidate positions for the object in the next frame. Consequently, motion tracking is a search problem: given an input sequence, the tracker searches for interesting features in each frame.

Tracking posable bodies like the human figure is difficult due to the extreme number of candidate positions. However, applications for human motion tracking in animation, surveillance, and human-computer interaction are extensive [4, 7, 22, 23] and motivate the desire to produce motion trackers that can reliably track the human figure.

1.2 Applications of Motion Trackers

Motion capture systems, widely used by animators to generate believable motion for complex sequences, rely on sensors and multiple cameras to identify and track limbs and joints. Motion capture produces a stick figure model (*i.e.*, a skeleton) which an animator may later clothe with a skin.

Automatic motion capture describes the use of a motion tracker to analyze an input sequence and to generate this model without the need for sensors or multiple cameras. (Brey has written an excellent survey.) This represents more freedom for actors and less cost for animators, since expensive motion capture studios may be replaced with commodity hardware.

Surveillance may also benefit from motion tracking by permitting cameras equipped with tracking logic to respond to movement in their input. For example, normally stationary cameras may pan upon detecting movement. This principle extends to firing weapons, locking doors, and other responses to intrusion. Coombs and Brown have a good discussion of moving cameras for tracking applications [7].

Human-computer interaction can use motion tracking to produce more believable response to human motion. For example, a robot head may pan with human motion. Object recognition is also a fundamental component of human-computer interaction.

1.3 Limits of Motion Trackers

Unfortunately, robust tracking remains somewhat elusive. Solutions to the problem typically attempt to fit recorded data to figure and motion models. For example, a figure model may be a connected series of line segments that approximate the human skeleton. A motion model may include joint constraints to prevent the tracker from identifying an outlier as a correct movement of a joint.

We may think of the tracker as an intelligent search agent looking through the input sequence to identify features that look like the figure model and that also conform to its understanding of how that figure should move. This is somewhat easier said than done. Object recognition depends on a great number of factors that are extremely difficult to characterize and estimate: illumination, occlusion, background clutter, and noise are but a few examples. In the case of human figure tracking, the situation is even more complicated due to the complexity of the model. A realistic one will have at least thirty degrees of freedom, so a tracker may have to search through a thirty-dimensional space to find a correct answer. A brute-force search of this dimensionality is intractable.

Current tracking systems vary in their solutions to this problem. Many rely on a learned probabilistic model [8, 17, 24, 27], which both obscures variations in the input sequence and trims the search space by predicting the location of the tracked figure in the next frame. Others use image-based techniques, in which specific characteristics of the input sequence are used to track the figure [6, 15, 20, 21, 23]. Solutions often contain a combination of the two.

General tracking—reliably disambiguating any novel figure from its background and following it without error—has not been accomplished. Isard and Blake's CONDENSATION algorithm uses particle filtering and a learned motion model to great effect, but each motion must be first learned. Ho, Lee, Yang, and Kriegman present an elegant, fairly general solution in [15], but its motion description is low-level and therefore not useful for many applications.

These two examples present an essential question: how can we produce a high-level motion description from a low-level model? Low-level models permit more generality in the tracking process, and high-level descriptions specify the output more precisely.

This work attempts to answer that question by conditioning the model to the input sequence and deriving semantic information from it. Rather than relying on skeletal or blob models, we segment the human figure into six pieces and use the area of the segments to perform tracking.

1.4 Contributions of This Work

The principle contribution of this work is to show how such a model may be used in conjunction with robust segmentation software (in particular, the Flow Discretization Algorithm [9]) to perform reasonably reliable, general tracking. By limiting the specificity of the model and deriving information from the geometry of the figure, we demonstrate that it is possible to track the human torso without semantically identifying it as such. This allows substantially more flexibility than many current solutions whose models are so specific as to limit them to a particular domain [21] or that require extensive learning processes for each motion learned [17]. It also produces more useful output than solutions that use low-level models [15].

We limit ourselves to tracking the human torso to provide proof-of-concept, though it should not be difficult to extend the algorithm to track other segments (see Section 6.1). Our algorithm is similar in spirit to other image-based solutions, differing principally in the model used to perform tracking.

1.5 Torso Tracking

It may be argued that torso tracking is a domain of little importance; however, it merits consideration for several reasons.

First, the torso is very rarely completely occluded, and never by itself. It is almost always at least partially visible, which simplifies identification and location.

Second, reliably segmenting the torso from the human figure limits the search space for other features such as the head, arms and hands, and legs. This reduces the complexity of the tracking problem and improves performance. The presented system is also amenable to extension to locating and tracking other body parts.

Third, the torso has restricted mobility compared to the other parts of the human body. Therefore, motion models for the torso do not need to specifically accommodate wide ranges of motion, but may instead restrict themselves to relatively simple changes in orientation caused by affine transformations.

Tracking the torso is still problematic, however, especially in monocular video. Yet monocular video is the most common form of data and the easiest to produce. Moreover, monocular video is the

Figure 1.1: Various problematic inputs.

Figures (b) through (d) have been inverted for clarity.

format of all archival video footage, so algorithms that can analyze single-camera video sequences are necessary.

Projection forces the tracker to infer three-dimensional data, which requires simplifying assumptions that are not always accurate [23]. As a result, certain occlusions cannot be resolved, and it is difficult to determine the direction of rotational movements.

Object recognition becomes more problematic during occlusion since elements of the human figure may be included as a part of the torso as a consequence of poor segmentation (see Figure 1.1 for examples). The arms may be included when crossed over the body (1.1.b). The inverted V-angle made by the legs is hidden when viewing the body in profile, often causing the legs to be incorrectly assigned to the torso (1.1.a, 1.1.d). In situations when the shoulders are hunched, the curves of the neck are obscured, which may cause the head to be included (1.1.a). Full occlusion of the torso, while rare, also causes problems (1.1.c).

Trackers must also account for noise, which may arise due to data acquisition and other operations such as background subtraction. Noise corrupts the figure of interest and may destroy pertinent torso features (1.1.d). Robustness to noise may be achieved by a combination of methods. Morphological operators are used to fill in holes in this solution; machine learning has been used in other solutions to permit tracking in cluttered environments ([17, 27]).

In order to generalize the solution as much as possible, the tracker incorporates an amorphous, area-based model for the human figure. Six segments are allocated: one for the each arm and leg, one for the head, and one for the torso. This permits quite a bit of flexibility when examining each image in the input sequence since no orientation fitting is assumed or required. Moreover, it reduces the need for a motion model. It may even be applied to objects that are not human at all.

This results in some significant advantages compared to other motion trackers:

1. No off-line learning process is required to track. Initial investments in most tracking solutions are time-consuming.

2. Any segment may be tracked. While we demonstrate here how to track the torso, the algorithm may be extended easily to track any segment.

3. Using an area-based model avoids pose estimation, angle fitting, and other computationally expensive techniques.

There are trade-offs to these benefits, however. The tracker may require several observations before settling on the torso. This may be caused by noise, occlusion, or poor initialization. The granular nature of the model also makes it difficult to guarantee that the selected segment actually *is* the torso. Its dependence on segmentation may also be considered a weak point; poor segmentation will undermine the algorithm considerably.

The following section describes prior work in the field of motion tracking for comparison to the presented one. Chapter 3 describes the details of the algorithm used to identify and locate the torso within a sequence of images. Chapter 4 describes the implementation of the algorithm, with results and conclusions following in Chapters 5 and 6 respectively.

Chapter 2

RELATED WORK

Motion tracking is a well-studied topic and has been addressed in several ways. The standard reference for probabilistic solutions is Blake and Isard's CONDENSATION algorithm [17]. CONDENSATION uses a Bayesian framework (a form of particle filtering) to track objects through frames. A summary of the algorithm is below, along with mention of two important extensions [8, 27].

Also considered are works by Mamania, Shaji, and Chandran [21], Remondino and Roditakis [23], Ho, et al. [15], and Chen, Lee, Parent, and Machiraju [6], each of whom provide solutions more similar to the one presented here. Their methods typically rely on a variety of image characteristics and model approximation to identify and track body parts. Typical image characteristics include color cues and depth information. Curve fitting is used to permit trackers to identify curves on a silhouette with some degree of probability (one example, for instance, might be identifying the legs by finding the inverted V-angle [6]).

2.1 Particle Filtering

The subject of particle filtering is extensive, so only the very basics are described here. Presented with a system, we wish to estimate its prior, current, or even future state by sampling. In high-dimensional systems (such as the pose of a human body with thirty degrees of freedom), sampling must be sparse or the process becomes intractable. Samples are usually referred to as "particles." Deciding where to sample the system most effectively is a difficult problem.

Provided with a model for the system and for how it changes over time (*e.g.*, a model for the human body and movement or joint constraints), hypotheses may be formulated to help predict where samples should be taken to glean the most accurate view of the state of the system. These hypotheses are usually discretized as probability distributions over the image space.

Particle filtering is used in a variety of applications: signal processing, motion tracking, and even economics. Modern particle filtering began in 1960 with the Kalman filter (see [30] for a good introduction). There are two main drawbacks to using a Kalman filter to estimate system state: (*1*) it cannot handle multiple hypotheses and (*2*) it expects Gaussian processes, *i.e.*, normally distributed data.

Motion trackers based on particle filtering, such as CONDENSATION, use a Bayesian filter rather than the Kalman filter. They accommodate multiple hypotheses and multi-variate data in non-linear, non-Gaussian processes. They are naturally more attractive for motion tracking, in which multiple hypotheses are common. In addition, Bayesian filters are typically easier to implement than Kalman filters.

Bayesian filtering [3, 12] takes its name from Bayes' theorem, upon which it relies:

$$P(A_i \mid A) = \frac{P(A_i)P(A \mid A_i)}{\sum_{j=1}^{N} P(A_j)P(A \mid A_j)} \tag{2.1}$$

in which $P(A_i \mid A)$ should be read as "the conditional probability of event A_i given the occurrence of events A." Each one of the A_i is a particle or sample from the system. Bayes' theorem describes the entire state of the system and uses it to find the probability of the occurrence of a particular event. Generally, the Markov assumption[1] is used when Bayesian filters are implemented; otherwise, summarizing the state of the system becomes computationally intractable.

In particle filters, the current state of the system is guessed from samples and updated according to the following equation:

$$Bel(\mathbf{x}_{t+1}) \approx S_{t+1} = \{\langle \mathbf{x}_t^i, w_t^i \rangle \mid i = 1, \ldots, n\} \tag{2.2}$$

where each $\mathbf{x}_t^i$ represents the state of the system (a vector) and w_t^i the weight of a particular sample. The time step is represented by t and the sample index by i. The predicted state of the system

[1] *I.e.*, that the current state depends only on the prior state, or that the prior state encodes information about all states preceding it.

is given by $Bel(\mathbf{x}_{t+1})$ (under the Markov assumption, this becomes S_{t+1}) [12]. The weights are adjusted iteratively by inputs to the system—in our case, successive images from a video sequence. This probabilistic model accommodates new samples and updates the weights accordingly.

Bayesian filters, however, have several problems. Sminchisescu and Triggs note that statistical approximations in many applications are difficult, if not impossible to find. Moreover, Bayesian filters may be trapped into choosing incorrect samples at local minima rather than attempting to optimize around a global minimum. Minimal-cost samples are often found around high-cost samples that are unlikely to be resampled, so the samples need to be locally optimized [25, 26].

2.2 CONDENSATION

Blake and Isard's seminal CONDENSATION (CONditional DENSity propagATION) algorithm demonstrates the use of Bayesian filtering in motion tracking problems and has become a standard reference for the technique. Specifically, it applies to tracking curves through cluttered backgrounds based on prior learned models of motion and hand-drawn contours of the figures to be tracked.

Clutter in a scene is typically manifested as a different mode in the distribution; should the tracked object be occluded for a sufficient time, a Kalman-based tracker may fixate on the occluding object. By permitting multiple hypotheses, the CONDENSATION filter may recover from situtations that would render a normal Kalman filter useless.

The motion model is learned by applying background subtraction to video sequences and applying Kalman filtering to those sequences. Iterative filtering produces increasingly good results when used with the CONDENSATION filter on novel data sets. The learning process requires substantial user-intervention, as does the generation of the contour model.

The authors report good results, noting that they are able to run the tracker on complicated sequences at roughly six frames per second in 1998. Though the initial investment in training and image processing is extensive, the application to later data sets is tremendous: the tracker reliably recognizes objects in clutter and tracks complex motion. A typical example is tracking a wind-tossed leaf on a bush.

The CONDENSATION algorithm has been extended and refined in two particular cases to allow tracking of multiple objects, such as a flock of birds [27] and to reparameterize the search space by simulated annealing [8]. Other statistically-grounded solutions also exist [24, 25, 26].

The principle disadvantage of the CONDENSATION algorithm is that it requires training for novel data; the authors have produced a system to track objects in clutter, but training is required for each new object or motion, severely compromising the generality of the solution. This makes it difficult to envision using it in a production environment.

2.3 Image-space Solutions

Another class of solutions may be referred to as *image-space solutions*. These rely less on probabilities and more on image cues such as contours, constrast, depth, and color. The solution formulated in this work is an image-space solution.

Image-space solutions are not as widespread as particle filtering methods for several reasons. The principal difficulty is that tracking figures in a cluttered background has proven difficult, and preprocessing (especially background subtraction) is problematic. Image characteristics are also sensitive to fluctuations in intensity and color that may skew results. Nonetheless, there are some interesting applications of image cues.

Mamania, et al. use an image-space method to find 3-D model data in the Bharantanatyam dance [21]. Remondino and Roditakis suggest a simple camera model so that an *a priori* known scaling constraint may be used to extract 3-D model data [23]. These papers describe pose estimation, a form of model fitting that is most often applied to automatic motion capture. Chen, et al. combine image features and physical constraints to track limbs and the head [6]. Ho, et al. use learned linear subspaces to track the head [15]. Both papers describe means of performing tracking.

2.3.1 Bharantanatyam Dance

Mamania, et al. extract 3-D data from a monocular video sequence by using visual cues and assumptions about the nature of the camera. Their work focuses exclusively on the Bharantanatyam dance. The complexity of the problem is reduced somewhat by a wide golden belt common to all costumes for the dance. They identify skin regions by color to identify the arms, belly, and head of the dancer.

Such a solution is undesirable since it relies on color information and operates in a very limited domain (though it should be noted that the dance itself is extremely complicated!). This restricted generality permits accurate pose estimation but would be difficult to apply to other input.

2.3.2 Camera Fitting

Remondino and Roditakis simplify the matter somewhat by using a pinhole model for the camera. They assume that the camera is far enough away from the scene that perspective effects may be ignored. Anthropometric data is used to recover perspective poses after they are extracted from the video. (The authors also describe how to apply the algorithm to a perspective camera if its focal length is known.)

This work improves upon Mamania, et al. by eliminating domain considerations and leveraging a camera model for pose estimation. Pose estimation is elegant in this system due to their assumptions and a user-provided scaling factor used to recover the relative lengths of line segments. While it is quite useful in extracting 3D information from 2D vertices, the authors do not describe how they identified the figure in each frame.

2.3.3 Linear Subspaces

Ho, et al. describe an algorithm that relies on learning linear subspaces to track objects through video. It relies solely on intensity information and learns the position of the object based only on prior observations in the sequence. No off-line learning is required.

Data points surrounding the area to be tracked are provided by the user. The surrounding area is sampled at different locations and orientations, from which the authors generate a vector space $\mathbb{R}^K$. The appearance model of the figure is represented as a linear subspace of $\mathbb{R}^K$, where K is the dimension of the sample space. Ho, et al. sampled from a 19x19 grid. Tracking is accomplished by updating the subspace over time using a Gramm-Schmidt process [28]. The authors' main contribution is the use of the uniform L^2 reconstruction norm:

$$Error^{\infty}(L, \{x_1, \ldots, x_N\}) = \max_i d^2(L, x_i) \tag{2.3}$$

where L is the subspace, the samples are $\{x_1, \ldots, x_N\}$ and d^2 is the standard distance between the subspace L and the observation x_i. This permits a better assessment, they argue, of the quality of the estimation. Moreover, the generality of the uniform reconstruction norm improves the algorithm's efficiency.

However, it does not appear from their results that precise contours are extracted from the video. Such contours are useful if additional model fitting is desired. Moreover, the lack of a model means

that it is difficult to glean semantic information from the output. This is undesirable in many situations.

2.3.4 Physical Constraints

Chen, et al. describe another method that couples image characteristics and physical constraints [6]. The focus of the work is how to reconstruct motion capture data from a monocular video sequence, though many of the techniques apply to the tracking problem.

Using a single video stream makes reconstructing the model in the input sequence an ill-posed problem, which they attempt to address by extracting silhouettes from the input sequence and matching them against silhouettes generated by fitting model parameters to the input.

The authors introduce a novel cost function, to select a motion parameter set that most closely approximates the difference between the input silhouette and model silhouette:

$$f(S_{input}, S_{model}) = \sum_{i}^{H} \sum_{j}^{W} c(i,j) \tag{2.4}$$

where S_{input} is the silhouette extracted from the input and S_{model} is the silhouette generated from the fitted model. H and W describe the dimensions of the images.

The cost function

$$c(i,j) = \begin{cases} 0, & S_{input}(i,j) = S_{model}(i,j) \\ d(i,j), & \text{otherwise} \end{cases} \tag{2.5}$$

bears some discussion, since it is used aids tracking limbs and other features that may be obscured by occlusion. The authors define a *core space* that contains pixels close to the skeleton of the figure. The skeleton characterizes limbs well, and a distance function may therefore be used to penalize the cost function when the model silhouette does not overlap regions near the core area. This distance function is defined by

$$d(i,j) = D(S_{input})(i,j) + wD(\tilde{S}_{input})(i,j) \tag{2.6}$$

where $D(S)$ is the Euclidean distance transform of the binary image S, $\tilde{S}$ is the inverse image of S, and w is a weighting factor that weights the importance of the coverage of the core area relative

to the area of mismatch outside of the silhouette area. Functions f and c are together called the *core-weighted XOR*.

In cases where minimizing the objective function is insufficient, the authors use human anatomy and joint constraints to help characterize motion. The authors report tracking results of about three seconds per frame in simple cases, five in more complex.

In each case, except for Ho's linear subspace algorithm, the models and characteristics used represent significant *a priori* constraints that make general tracking relatively difficult. The results from Ho, et al. are so general that they are probably difficult to use in situations where tight fitting is required.

Chapter 3 describes our algorithm, which attempts to provide some generality in tracking by leveraging geometric and topological information contained in the contour of the figure. Enough data are retained that it should be possible to perform higher-level functions as well, such as pose estimation and extraction (*i.e.*, automatic motion capture).

Chapter 3

TORSO TRACKING

We begin to specify the algorithm by describing its desired features. We then discuss its motivation, the human model used, assumptions made, and individual components of the algorithm.

3.1 Algorithm Features

A good tracking algorithm should have the following characteristics:

1. Achieve nearly automatic tracking.
2. Eliminate dependency on color and stereo information.
3. Robustly handle noisy or severely corrupted video.
4. Process video quickly.

When successfully met, these goals produce a tracking system capable of operating on commodity hardware while still producing reliable data in a reasonable amount of time. Most of the objectives have been met satisfactorily, although room for improvement certainly exists. See Chapter 5 for details.

3.2 Motivation

The question posed in Chapter 1 drives the motivation for our tracking algorithm. How can we generate high-level motion descriptions using low-level models? Low-level models permit good generality when tracking, but make deriving useful information from the motion description fairly difficult.

We address this problem by using a low-level model that is conditioned by input image features to derive semantic information that can be used to produce motion descriptions suitable for higher-level functionality such as automatic motion capture.

3.3 Human Figure Model

The human body has been commonly modelled as a stick figure or kinematic chain [8, 27], contours or bounding boxes [15, 17], or blobs (*e.g.*, cylinders, cones, metaballs, ellipsoids, etc. [13, 20, 24, 25, 26]). Usually coupled to the geometric model are motion or joint constraints (*i.e.*, a movement model) [6, 17, 27]. These help to minimize the search space and to correct poor tracking estimates.

Unfortunately, current models are relatively inflexible. We argue that flexibility is more easily achieved when the model used enforces no semantic constraints on the input sequence. Using a kinematic chain or skeletal model naturally assumes that the input contains such information, for example. Models whose meaning and definition vary with the input are critical to making trackers with a higher degree of flexibility.

To that end, we model the human figure with six segments and allow the characteristics of those segments (such as location and area) to change with the input. The six segments represent the limbs (two arms and two legs), the torso, and the head. The segment with the largest area is assumed to be the torso. Anatomically, the torso is considered to extend from the neck to the hips.

Given an image I, we define $C \subset I$ to be the set of pixels that compose the extracted contour of the figure. When segmented, $C = \{S_1, S_2, \ldots, S_6\}$, where each S_i is a segment. A segment, therefore, consists of vertices on the contour of the figure. Characteristics such as area and location may be determined by using axis-aligned bounding boxes or by triangulating the surface and using Heron's formula [29].

No *a priori* curve-fitting, stick-models, or motion constraints are considered or required.[1] Rather

[1] It is important to realize that such underlying representations are still possible in the segmentation stage. The Flow Discretization Algorithm [9], for instance, relies on the medial axis of the contour, so it has a skeletonization of

than approaching a motion model from a deductive perspective (*i.e.*, here is a database of all of the motions learned by the tracker, please fit the input to one of them), we present an inductive method (*i.e.*, based on what is visible, please decide what part of the figure is the torso).

An additional advantage is that the semantic meaning of each segment may change with the application. No longer does the model enforce any semantic constraints on the underlying image. If one wished to track the palm, for instance, it would be possible to use the same method. By attempting to search by a different area or location, one may also easily track the head (see Section 6.1).

3.4 Assumptions Made

Input to the algorithm is a sequences of images that make up a video. We assume that the input sequence has undergone a background subtraction step prior to running the algorithm.[2] Robust background subtraction details may be found in [18]; Kim has written a program that seems suited to surveillance situations [19].

The initial pose of the figure is used to initialize the tracker; we assume that the initial configuration is correctly segmented. This is done to avoid requiring user interaction at this level, not because it is always an accurate assumption. While this assumption may cause poor tracking at the outset, it should be corrected after several observations.

We finally assume that only one figure appears in the image. This avoids complexities difficult to handle in the segmentation step that are not pertinent to the tracking problem. Means for including multiple figures are discussed in Chapter 6.

3.5 Algorithm

The algorithm is divided into three stages: pre-processing, segmentation, and post-processing. During the pre-processing stage, the input data are filtered to remove noise and extract contours. The Flow Discretization Algorithm [9] is then applied to analyze the contours and segment the figure.

the figure. However, this skeleton is "translated" into segments rather than being used for model-fitting. In this sense, we may argue that the model is derived *from the input.*

[2]The term "segmentation" may refer to segmenting the foreground from the background and also to breaking the figure into pieces. In this work, the latter definition is used exclusively; when background subtraction is meant, that term is used.

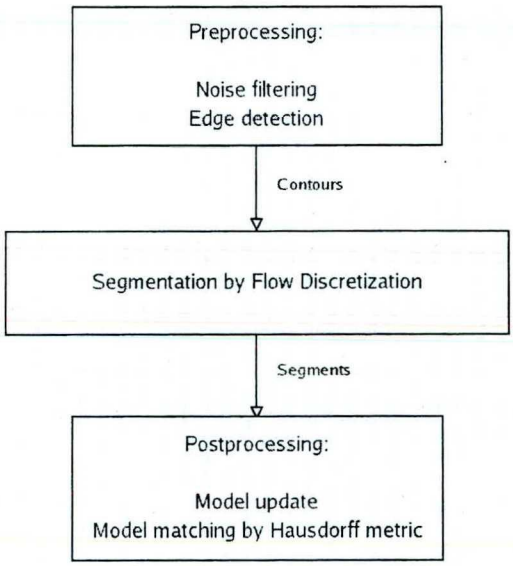


Figure 3.1: Algorithm Flowchart.

Initially, the torso is assumed to be the largest segment. During post-processing, the torso model is updated statistically and matched using a Hausdorff metric [16] to the current frame.

3.5.1 Pre-Processing

The results from the pre-processing stage are the contours of the figure. Since noise corrupts the contour significantly and may greatly influence the results of the segmenter, it is important that noise be filtered from the input image. Noise may arise from several sources, so a combination of filters may be required to eliminate noise. Gaussian smoothing is used to filter random noise; morphological filling closes holes generated by background subtraction. Thresholding is then applied to produce a binary image from which the contours can be extracted.

To facilitate robust segmentation, the extracted contours should exhibit the following properties:

Thin contours. The complexity of reconstruction and segmentation depends on the number of input vertices and becomes computationally expensive when contours are several pixels thick.[3] The Flow Discretization Algorithm will also report packed vertices as segments, so a continuous, thick contour may have more area than the inside of the figure and be incorrectly represented as the torso.

[3] In practice, widths of more than seven or eight pixels greatly increased processing time.

Geometric plausibility. The edge detector should avoid detecting false boundaries due to noise. See, for example, Figure 4.1, which shows an actor masquerading as a genie. The input image has been corrupted by noise from background subtraction, and so the boundaries discovered by various detectors are also poor. Noise adversely affects segmentation, so contours that are not truly parts of the figure will often create spurious segments.

The suggested means for meeting these criteria are examined in Section 4.4.1.

3.5.2 Figure Segmentation

To segment the input image, we use Samrat Goswami's implementation of the Flow Dicretization Algorithm [9]. It takes as input vertices on the contour and returns a list of segments sorted by area. The mathematical properties of the algorithm are interesting and bear some investigation, and its use in tracking is novel.

The Flow Discretization Algorithm relies on computing several features from the input point sample. In particular, critical points are identified and the Voronoi diagram and Delaunay tesselation are computed. These are ultimately combined with the flow induced by points on the boundary to form closed stable manifolds. These manifolds are the segments produced by the algorithm.

Some discussion of each of these features of the algorithm follow. This discussion, derived from definitions provided by the authors, is included here for completeness. The formulations are included from [9].

3.5.2.1 Critical Points and Flow

Dey, et al. begin to develop their segmentation algorithm by describing a theory of flow induced by shape. Flow is derived from the input boundary which is analyzed to find critical points. The definition of flow proceeds through first defining a height function for the shape, its anchor set, and finally regular and critical points.

Height Function The height function for a shape Σ is a distance function h:

$$h : \mathbb{R}^d \rightarrow \mathbb{R} = \min_{p \in \Sigma} \|p - x\|^2 \text{ for all } x \in \mathbb{R}^d \tag{3.1}$$

where p is a point on the surface of Σ. The authors describe Σ as a compact manifold of dimension $d-1$ that is embedded in $\mathbb{R}^d$. It is this function the authors hope to characterize to describe the flow induced by shape.

Anchor set Associated with the height function is an anchor set, which describes the set of points closest to $x \in \Sigma$. In the discrete domain, this corresponds to Voronoi planes. The anchor set for each point in the d-dimensional real space is given by

$$A(x) = \operatorname*{argmin}_{p \in \Sigma} \|p - x\| \tag{3.2}$$

These two definitions permit the authors to define the critical points of the height function *h(x)*:

> For every point $x \in \mathbb{R}^d$ let *H(x)* be the convex hull of *A(x)*, *i.e.*, the convex hull of the points on Σ that are closest to x. We call x a critical point of h if $x \in H(x)$. Otherwise we call x a regular point.[9]

(Regular points are smooth; critical points are non-smooth.) Critical points are the local extrema and saddle points of the height function. These points may be used to find a vector field that describes the gradient of the height function. The direction of steepest ascent at every point x in the height function may be derived by computing the distance between x and the point in *H(x)* closest to it.

The normalized vector field defines the flow on the space described by the point sample.

3.5.2.2 Stable Manifolds

Having derived the critical points and flow of a point sample, we may now consider the stable manifold of a critical point:

$$S(x) = \{y \in \mathbb{R}^d : \lim_{t \to \infty} \phi_y(t) = x\} \tag{3.3}$$

where $\phi_y(t)$ defines the flow function. The stable manifold of a critical point is the set of all points that flow into that point. The stable manifolds of all critical points are the segments of the space $\mathbb{R}^d$:

$$\mathbb{R}^d = \bigcup_x S(x) \tag{3.4}$$

This definition of the segments of a space is defined in a continuous domain, and so the flow must be discretized. This is accomplished by taking a finite point sample from the boundary. Critical points are determined as before, but flow is found using Voronoi and Delaunay objects.

3.5.2.3 Application of Flow Discretization

The Flow Discretization Algorithm provides a fairly intuitive segmentation of the input figure. The segments are relatively natural, generally corresponding to geometric discontinuities such as hard angles or abrupt changes in curvature. Since the angle and curvature of the human body changes throughout an input sequence, this allows our algorithm to condition its results on the input sequence itself, rather than requiring *a priori* constraints on the movement of the figure. This generalizes the tracker significantly.

This can, however, cause problems. Figure 3.2 displays several results from Goswami's implementation of the algorithm. In many cases, discontinuities that would normally help segment the torso are obscured and poor segmentation results. Post-processing helps correct these errors.

An additional disadvantage is that the implementation does not recognize multiple figures. Spurious noise data included in the sequence may seriously harm segmentation, since the algorithm will interpolate a curve between the genuine contour and the false noise. An example of this may be found in Figure 5.2. An extremely large segment has been derived due to a very little bit of noise; this, in fact, was assumed to be the torso!

3.5.2.4 Advantages

The Flow Discretization Algorithm is extremely efficient. Algorithms for computing the Voronoi diagram and Delaunay tessellation have been designed to run in $O(n \log n)$ [14]. Segmentation proceeded in real-time for the samples provided during testing.

The authors do note that the algorithm suffers in the presence of noise [10]. However, in practice, it suffers only when noise *adds* artifacts to the input image, as in Figure 5.2 (note the horizontal bar in the upper-left corner of the input). This type of noise is relatively easy to filter out via morphological operations, smoothing, and thresholding. In many cases, it may be done automatically. With respect to noise that corrupts or *removes* parts of the input image, the segmenter responds admirably. In fact, in these cases, when relatively few contour points are provided (Figure 5.3), the results may be even better than when a dense sample is taken.

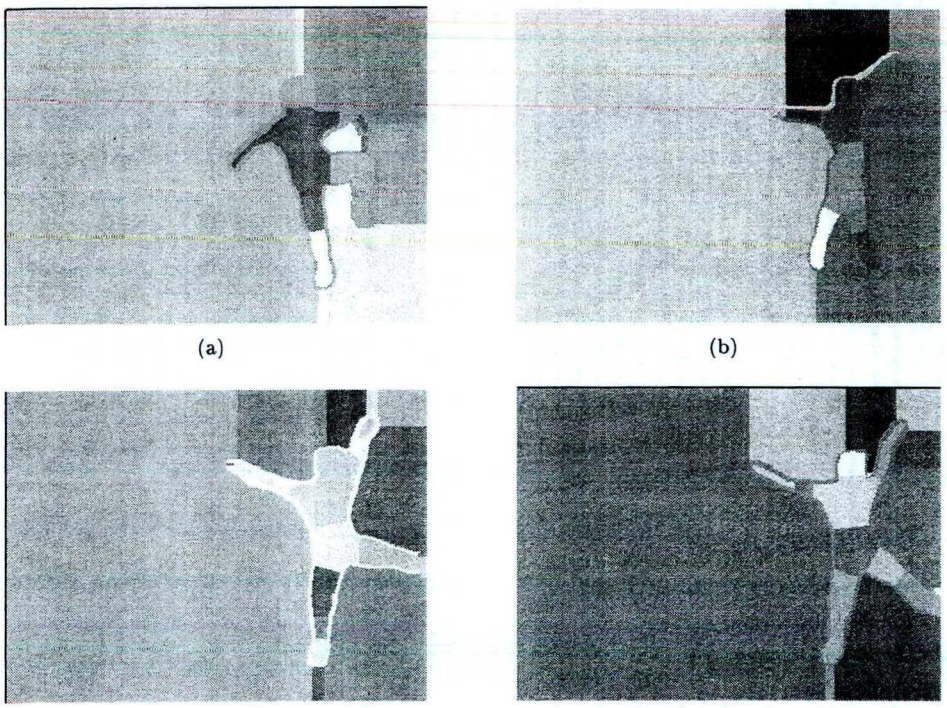


Figure 3.2: Flow Discretization segmentation examples.

In this figure, data with no noise were used as input. (a) demonstrates rather poor segmentation—note that the right arm of the figure is included as a part of the torso. (b) shows a slightly better example, though the torso includes a bit too much of the legs and is located rather low. (c) is better, correctly removing arms, legs, and hips, but including the head. This happens because the neck is not visible. (d) suggests the best torso of the four, removing arms, legs, hips, and head.

3.5.3 Torso Identification

Torso identification proceeds in two sub-stages. In the first, the model of the torso is updated according to prior observations in the sequence. In the second, the updated model is matched to the current input image.

3.5.3.1 Model Update

There are a variety of circumstances in which it is difficult to segment the torso reliably from the input figure, and so the model for the torso must be guessed. While the examples shown are generated by Flow Discretization, this is likely to be common in any segmentation step. See Figures 5.1 to 5.7.

We briefly consider the three affine transformations and how they impact the segmentation process.

Translation In general, translation in the x-direction poses little problem for the algorithm. The side-to-side motion of the torso is usually only accompanied by swinging of the arms, and occlusion is rare. Translation in y and z (see "Scale", below) does cause more problems, however.

Movement in y is often accompanied by extra movement in the arms and legs. This movement may obscure features that aid in segmenting the torso. In order to alleviate this problem, denser sampling is necessary.

Scale Scaling of the torso usually arises due to changes in the depth of the figure, since depth under perspective projection may be approximated by scale differences under orthographic projection [23]. Therefore, the torso detected farther away from the camera should have a smaller area than the one detected closer to the camera.

In practice the depth of the figure influences the segmentation process. If smoothing or closing obscures features, detail is lost and segmentation may recommend a torso that is too large (Figure 5.4).

Rotation As shown in Figure 3.2, rotation has an unfortunate effect on segmentation. When the silhouette is viewed in profile (*i.e.*, during rotation about the y-axis), the torso will be tall and thin, often extending from the head or shoulders to the knees.

Recovery In order to recover from mistakes by the segmenter, the relative confidence of each frame compared to others in a subset of the input sequence is assessed.

Define $F = \{f_1, f_2, \ldots, f_n\}$ to be the set of all frames in the sequence and $W \subseteq F$ to be the window of frames under observation. Let f_i be the current frame such that $W = \{f_{i-m}, f_{i-m+1}, \ldots, f_i\}$. We refer to W as a *backward window* or *back window of* F.

Further define

$$SEM = \frac{\sigma_a}{\sqrt{m}} \tag{3.5}$$

where σ_a is the standard deviation of the area of the torsos in W. SEM, then, is the standard error of the mean of the area of the torsos in a given window. If the value of SEM is high, then the confidence in each frame is low; if the SEM is low, the confidence is high.

Given the area of the frame's torso a_i, we define the *area confidence* of the i^{th} frame as follows:

$$ac_i = \begin{cases} 1 - SEM/\mu_a & , a_i \leq \mu_a \pm t * \sigma_a \\ 0 & , a_i \geq \mu_a \pm t * \sigma_a \end{cases} \tag{3.6}$$

where μ_a is the mean area of the torso over W and t is a user-specified tolerance value. In essence, we say that if the current observation falls within a user-defined tolerance (expressed in standard deviations), the frame is acceptable. If the observation does not fall within this tolerance, the frame is unacceptable and will not be weighted in the final outcome. Having thus ascertained the confidence of each frame in W, we proceed to update the model.

Let $T = \{t_1, t_2, \ldots, t_m\}$ be the set of torsos associated with each frame $f \in W$. A torso is described by a set of vertices on the contour of the figure (as extracted from step one of the algorithm). These vertices are enclosed by an axis-aligned bounding box $b = \{min, max\}$, where min and max are the vertices that define the box. It should be noted that min and max have coordinates x and y respectively. The new vertices are determined as follows:

$$mnx = s * \sum_{j=1}^{m} (ac_j * mnx_j) \tag{3.7}$$

where mnx is the x-coordinate of the minimum vertex of the new bounding box and s is a scaling factor that reduces $\sum_{j=1}^{m} ac_j$ to unity. The other coordinates are likewise determined.

Given the new bounding box, an inclusion test is performed for each of the vertices on the contour. If the bounding box contains the given vertex, it is added to the new torso segment. In this fashion, the current representation of the torso is modeled.

3.5.3.2 Torso Matching

The difficulty with using a back window of frames—a necessary requirement for any application that would analyze data in real time—is that the position of the torso drifts significantly away from center. That is, the torso will be weighted more towards prior observation and will continually drift toward previous positions. In order to combat this, the updated torso is matched to the input sequence using a modified version of the Hausdorff matching technique described in [16], which is detailed briefly in the following paragraphs.

The torso is also shifted by a number of pixels to obtain a tight bound on the contour of the figure.

Hausdorff-based Matching The *bidirectional Hausdorff distance* between two sets A and B is defined mathematically as follows:

$$H(A, B) = \max(h(A, B), h(B, A)) \tag{3.8}$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{3.9}$$

is the *forward Hausdorff distance*. The Hausdorff distance $h(A, B)$ describes the maximum of the minimal distances between points in sets A and B. In general, the Hausdorff distance is asymmetric, and so the maximum of the *forward* (from A to B) and *backward* (or *reverse*, from B to A) distances is reported as the Hausdorff distance.

When the Hausdorff distance is small, we say that sets A and B are similar. It is worth noting, however, that outliers can have a dramatic effect on the reported distance. This motivates the

definition of the *partial Hausdorff distance*:

$$H_K(A, B) = K^{th}_{a \in A} \min_{b \in B}(A, B) \tag{3.10}$$

If we express K as a percentage of the points in a set, the partial Hausdorff distance is the Hausdorff distance between K percent of the points in set A to set B. This reduces the influence of outliers. More importantly, the partial Hausdorff distance may be used to match the torso model to the figure in the presence of occlusion, since only a percentage of the vertices need be visible.

An additional advantage of using the partial Hausdorff distance is that it automatically selects the best matching points. Other correlative techniques require a search, which may be time-consuming.

Huttenlocher and Rucklidge [16] describe a Hausdorff-based method for searching for a model in an input image, which was modified for use in this algorithm.

The input to their algorithm is a set of input vertices (the input image) and a set of model vertices. Using the forward partial Hausdorff distance, the authors subdivide the input image into "interesting" and "uninteresting" regions. The interesting regions are searched in the second stage, in which the model vertices are subjected to translation and scale transformations. Good matches are determined by calculating the reverse Hausdorff distance.

Due to simplifying assumptions made in this application, certain processes in the original algorithm are unnecessary. In particular, because vertices are taken from the contour of the figure, no scaling transformation is required. Background subtraction makes the area of interest considerably smaller. Since only one figure is in each frame, we may accept the first exceptional match. These features improve the time complexity of the algorithm significantly.

An unfortunate consequence of Hausdorff matching, however, is that the area of interest may vary widely between different data sets, and a tight bound on performance is difficult to establish. There is no guarantee that two input sequences of the same resolution will take the same amount of time to process, since the area of interest is dictated by the movement of the figure in the video. An upper-bound on performance may be found by using the size of the input frame. In practice, the actual performance is orders of magnitude faster.

Chapter 4

IMPLEMENTATION

4.1 System Specifications

The algorithm was implemented in a Fedora Core 4 Linux environment on middle-road hardware: an Athlon XP 1800+ processor (clocked at 1.53GHz) using the nForce2 chipset with 512MB DDR RAM. This is commodity hardware that is easy to find and replace.

The only special component that the system boasts is a Serial ATA hard disk, which permits somewhat faster transfers than typical disks. The video sequences had a resolution of 320 x 240.

4.2 Supporting Software

4.2.1 Programming Language

The tracker was written in Java version 1.5. All image processing components were written from scratch, including smoothers, edge detectors, and morphological operators.

4.2.2 Flow Discretization Implementation

We rely on Dr. Samrat Goswami's implementation of the Flow Discretization Algorithm, which is attached to the tracker by means of a Java `Process` object.

This incurs some overhead due to thread management and disk access. This is masked by the time taken to perform edge detection and does not affect performance. In practice, Goswami's implementation runs in real time on the contours supplied.

4.3 Sequences Tested

Several different motions and actions were tested in order to stress the segmentation and location processes:

Dance — The dance sequence is a computer generated sequence of breakdancing. It was the most complicated sequence, since it contained significant (author-induced) noise, rotation, translation in the xy-plane, and full occlusion of the torso. This sequence was generally poorly tracked.

Genie — The genie sequence featured a genie-like actor waving his arms wildly. This was used to test the effects of occlusion on the tracker. The input was corrupted by noise.

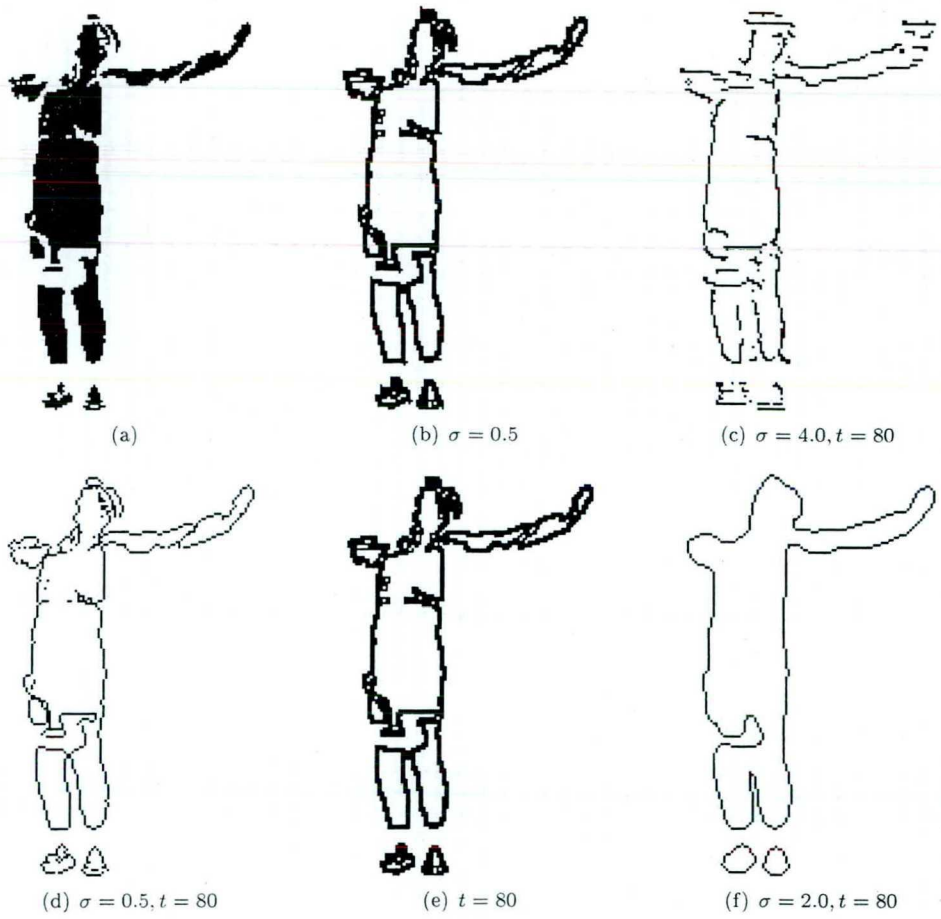
Hop — The hop sequence is computer generated. It features a figure hopping on one foot and rotating in a small circle. It stresses up-and-down translation and rotation. This ten-second sequence also exercised a variety of movements that "distract" the tracker.

Squat — The squatting sequence is computer generated and tests movement in the y-direction.

Tharp — The Tharp ballet sequence was used initially as a means of testing the tracker. The input is very clean, the contours easy to extract, and the motion relatively uncomplicated. It did test robustness against rotation.

Walk — The walking sequence is computer generated and tests translation in the x- and z-directions.

Each sequence contains markedly different motion and stresses the generality of the algorithm.



(a) (b) $\sigma = 0.5$ (c) $\sigma = 4.0, t = 80$

(d) $\sigma = 0.5, t = 80$ (e) $t = 80$ (f) $\sigma = 2.0, t = 80$

Figure 4.1: Edge detector outputs.

All images have been inverted for clarity and cropped for inclusion. (a) is the input image. (b) shows the output of the Canny detector with a relatively small σ and (c) the output with a high σ. (d) is the output from the difference of Gaussians. (e) shows the results of the Sobel edge detector; this is not substantially different from the Canny detector.

While the noise in the boundary may be reduced by smoothing, more problematic is the loss of the contour near the legs. Using morphological operators creates a more attractive boundary (f). The legs are still problematic, but noise has been reduced a great deal. More aggressive thresholding allows the curve reconstruction algorithm to produce better results, as in figure 5.3.

4.4 Algorithm Specifics

4.4.1 Edge Detection

Edge detection proved somewhat more difficult than expected. Contour detection is a vital part of the algorithm, since it is the input to the segmentation step. The contours should be well-formed in order to eliminate spurious segments from arising.

Figure 4.1 shows the output from several detection schemes. Spurious segments are created in this image because the legs are corrupted. Faithful edge detection picks up this corruption and diverts the segmentation process, which motivates a pre-filtering process consisting of morphological filling and Gaussian smoothing. Filling helps to eliminate spurious segments from arising because of background subtraction, which tends to leave holes in the input.

In constructing the algorithm, two means of edge detection were considered: Canny and the difference of Gaussians.

4.4.1.1 Canny Edge Detection

The Canny detector [5] was considered principally because of its ability to produce pixel-width contours, which helped reduce computation time in the segmentation process. Since it is a first-derivative edge detector, it is less sensitive to noise than second derivative operators.

However, Canny's operator is fairly slow. And, while its analysis of input data is impressive, such analysis is wasted in this application. Neither edge strength nor direction are used, though they are maintained by Canny's detector to perform hysteresis.

4.4.1.2 Difference of Gaussians

The difference-of-Gaussians (DoG) operator is a second-derivative method for edge detection, *i.e.*, it detects edges by finding the zero-crossings in the input signal. Second derivative operators are particularly sensitive to noise in the signal, but the DoG filters noise using Gaussian smoothing.

More importantly, the Gaussian convolution kernel is *separable*, meaning that a two-dimensional convolution may be performed using a one-dimensional convolution in both the x and y directions. The Gaussian in one dimension is formed by

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{\sigma^2}} \tag{4.1}$$

where σ is the standard deviation. The mean of the Gaussian is assumed to be centered at $x = 0$, and the kernel width is calculated by

$$w = 1 + 2 * \lceil 2.5 * \sigma \rceil \tag{4.2}$$

The kernel's separability reduces the computational complexity of convolution from $O(n^2)$ to $O(n)$, though it requires two passes. On large images, the impact is impressive. Even on small images, the results from the tracker improved markedly when using the DoG operator compared to Canny.

The DoG is applied by first smoothing the input image and then subtracting the result from the original. If I is the input image and $G_\sigma(I)$ is the image smoothed by a Gaussian with standard deviation σ, we may express the pixel-wise subtraction as

$$I - G_\sigma(I) \tag{4.3}$$

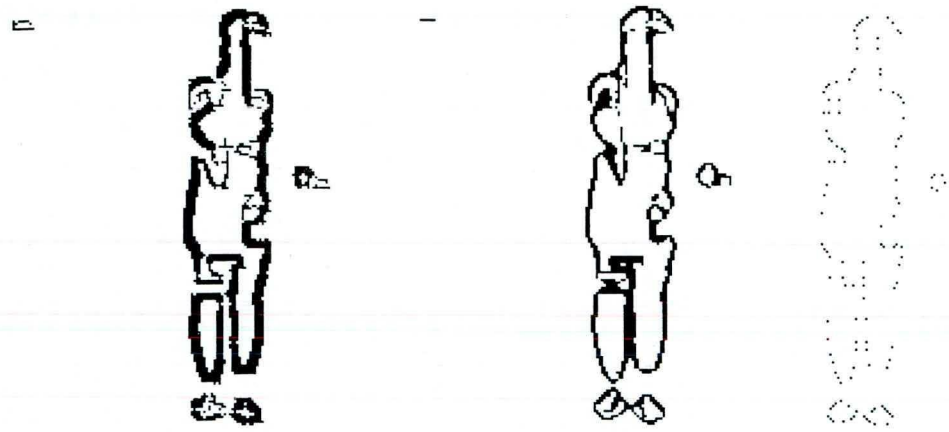
In many cases, the DoG is computed by performing two smoothing passes and subtracting the second from the first. This requires two standard deviations and may be formulated as

$$G_{\sigma_1}(I) - G_{\sigma_2}(I) \tag{4.4}$$

With respect to the first criterion specified in Section 3.5.1, pixel-width response, the DoG is very good. In fact, in situations when the provided noise is small enough, the DoG is used to great effect in the tracker. Unfortunately, it fails the plausibility criterion. It is even more faithful than the Canny detector in finding edges that are not a part of the "real" contour and so will not perform well when the silhouette is corrupted. Therefore, additional filtering is required.

4.4.1.3 Morphological Operators

When the input image is corrupted significantly, as shown in Figure 4.1, smoothing alone will not compensate for artifacts in the image, and so alternate approaches were examined. In particular, holes in the silhouette are very problematic since they are more likely to be segmented separately from the torso.



(a) $\sigma = 2.0$ (b) $\sigma = 2.0$ (c) $\sigma = 0.5$

Figure 4.3: Occlusion response.

Images were thresholded at $t = 128$. Note how the Canny detector in (a) and the difference of Gaussians in (b) both highlight the contour created by the arm. In (c), the location of the arm is reasonably well preserved—note the curve next the torso–but the middle noise is removed.

In the genie sequence, for instance, the hand occludes the torso for some frames, and the occlusion will be picked up by both the Canny and DoG detectors. This skews the segmentation in these frames and may affect the final identification of the torso (Figure 4.3).

In each frame, then, the holes need to be filled before smoothing and contour detection. A class of discrete and binary convolutions known as *morphological operators* fulfills this need. Morphological operators are so called because they function to recognize specific shapes in an image.

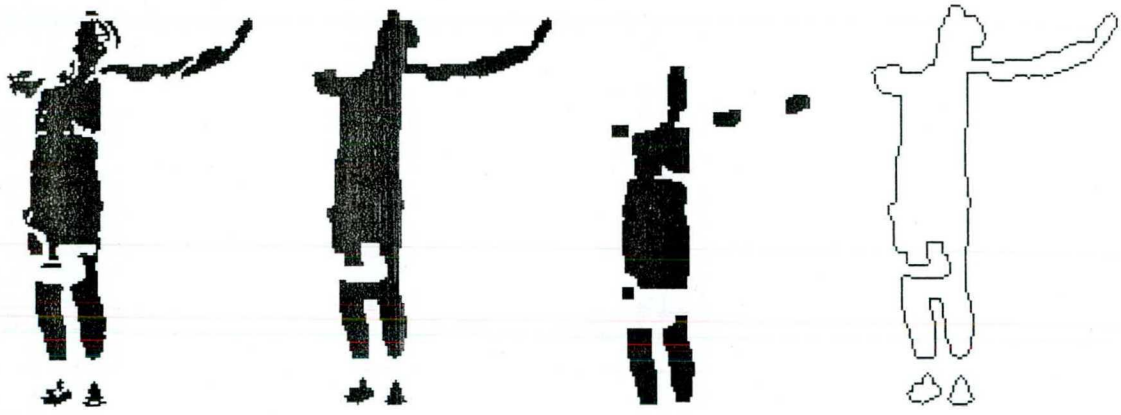
$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Figure 4.2: An example strutural element for erosion.

Holes may be filled by performing the *close* operation. The close operation consists of two steps, *erosion* and *dilation*. Morphological operators may also be used to perform edge detection.

Close, Erode, Dilate In erosion, a structural element (*i.e.*, a kernel used for purposes other than convolution) such as the one in Figure 4.2 is passed over the image and the pixel values in the neighborhood are summed together. If $sum \leq n * w^2$, where sum is the computed sum, n is the foreground intensity value, and w is the width of the kernel, then the center pixel is set to the background color. Dilation is the dual to erosion. In this case, if the sum is greater than zero, the center pixel is set to the foreground color.

A close operation first performs erosion and then dilation. The initial erosion eliminates all



(a) (b) size = 5 (c) size = 5 (d) $\sigma = 0.5$

Figure 4.4: Morphological operations.

(a) is the input image. (b) shows the result of morphological closure; note that the noisy holes in the torso are filled in, though there is still a very ugly gap at the hips. Potential problems also arise at the legs, which have been joined because of the size of the structural element. (c) shows the result of opening. This is clearly undesirable as portions of the arms have been removed and detail has been lost at the head. Worse, the feet are entirely missing. (d) shows the results of a difference of Gaussians on (b).

artifacts in the image that are smaller than the dimensions of the structural element, and the dilation fills in the remaining holes. This has two particular advantages in that it eliminates salt-and-pepper noise in the initial erosion step, and helps to fill in holes in a corrupted silhouette.

The dual of closure is opening. Opening is not preferred as a morphological operation because it will exaggerate holes created by noise.

Examples of each of these operators may be found in Figure 4.4.

4.4.1.4 Final Process

The final process included a combination of detectors. The difference of Gaussians is particularly attractive because of its pixel-width response and speed. In sequences where corruption to the figure is minimal, the difference of Gaussians is applied without any other processing. This is determined by the input value for σ. When $\sigma < 1.0$, no morphological operators are applied.

When the silhouette is corrupted in the image ($\sigma \geq 1.0$), the image is first subjected to closure and smoothing before the difference of Gaussians is applied. The delta between the Gaussian distributions was fixed at 0.5. This produced thin contours for minimal cost and preserved the contour effectively.

4.4.2 Segmentation

The implementation of the Flow Discretization Algorithm takes several parameters, only one of which was specified: the number of segments in the figure.

When it was unable to find six segments in the input data provided by the edge detector, it produced an error message, which caused processing to stop. This was typically a result of overly aggressive thresholding, which produced input that was too sparse. A thresholding level of intensity values at 80 produced good results on the sample input data, even when the contours were relatively sparse. See, in particular, figure 5.3.

4.4.3 Torso Identification

Identifying the torso consists of two steps. The first updates an internal model of the torso and the second matches it to the input image. The observation is then incorporated into the state of the tracker.

4.4.3.1 Model Update

The update step is generally straightforward, though there are some situations in which better results were obtained by providing a somewhat more complicated decision procedure. The confidence of the area was measured as described in Chapter 3. If it was found wanting, the segment was combined with the second-largest and the confidence re-measured.

```
function Hausdorff (list
A, list B)
  float dist ← −∞
  int i ← 0
  foreach a in A
    foreach b in B
      float tmp ← ‖a − b‖²
      if tmp > dist then
        dist ← tmp
      endif
    endfor
  endfor
end
```

Figure 4.5: Hausdorff Distance Algorithm

This was motivated by empircal evidence that the torso is often split into two pieces by the Flow Discretization Algorithm due to bends or noise. By combining the two, a more confident measure may be made and a better result is obtained.

4.4.3.2 Torso Matching

The matching step is also fairly straightforward. The Hausdorff distance is typically computed using the algorithm in Figure 4.5. This algorithm runs in $O(nm)$, where n is the number of vertices in A, m is the number of vertices in B, and $\|\cdot\|$ is the Euclidean distance norm. The repeated calculations of the norm are expensive, even when using the squared distance.

A more efficient means of computing the Hausdorff distance is to first calculate the distance transform of the underlying set of vertices.The distance transform assigns an intensity value to each pixel in an input image corresponding to that pixel's distance from the contour of the figure. This results in a two-dimensional array of floating-point values.

To calculate the Hausdorff distance, then, the array is "probed" at each of the model's vertices. Summing the distance at each vertex produces the Hausdorff distance. See Figure 4.6.

By precomputing the distance transform, the Hausdorff distance may be computed in $O(m)$. When computing reverse distances, this represents a significant savings in computation time, since the distance transform of the model need only be computed once, though the reverse distance may be computed thousands of times. In practice, this decreased computation time on rather trivial data sets from 10 seconds per frame to about two and a half.

The matching process may be further improved using early acceptance methods. Two observations permit this here. First, there is only one figure in the frame. Second, the vertices of the torso are taken directly from the contour of the figure in question. Therefore, we may assume that the first very good match (e.g., with a distance of one pixel) is the torso and terminate the search. Since the torso is usually located near the upper-middle of the figure, this can halve the search time.

The final output is then smoothed using linear regression.

```
function Hausdorff (float[][] dt, list B)
  float dist ← 0
  foreach b in B
    dist = dist + dt(b.x, b.y)
  endfor
end
```

Figure 4.6: Hausdorff Distance using Distance Transform

4.5 Default Settings

Several system parameters may be toggled or altered to change performance.

σ	The standard deviation for noise in source images. Defaults to 1.5 standard deviations.
W_s	The number of frames in the observation window. Defaults to 7 frames.
W_m	The size on a side of the morphological fill operator. The default operator is a 5x5 square.
t	The intensity threshold. Defaults to an intensity value of 80.
tol	The tolerance, in standard deviations, the tracker is willing to accept when measuring frame confidence. Defaults to 1.0.

In practice, these settings are sufficient for most types of video and present an affordable trade-off between speed and accuracy. When movement is particularly sporadic or noise levels are high, it is appropriate to change these parameters.

The Flow Discretization segmentation software used also takes some parameters. We accepted the defaults for ease of use, with the exception of the option used to change the number of outputted segments to six as described above.

Input sequences were subjected initially to the default parameters. These parameters were then changed in the course of experimentation to examine their effect. Where results are reported, parameters that deviate from the normal are explicitly noted.

4.6 Problems Encountered

This section considers from a more practical standpoint some of the difficulties encountered when generating output. Covered are the three main stages of the algorithm: edge detection, segmentation, and identification.

4.6.1 In Edge Detection

Edge detection posed few problems. In initial testing, thresholding was sometimes too aggressive, which produced a sample too sparse for the segmenter to process. This was rectified by lowering the default threshold value from 128 to 80.

Lower values for σ may also require a lower threshold value.

4.6.2 In Segmentation

Fixing errors in segmentation proved somewhat more difficult than in edge detection.

A consistent means of breaking the segmentation process seems to be to obscure the inverted V-angle for the legs. This has the effect of turning the legs into one unit and perhaps fusing them to the torso, which in turn skews the location of the torso considerably. This effect may be seen in Figures 5.4 and 5.7.

Not all sequences in which this occurs had problems. The genie sequence, in particular, responded very well despite the noise in its input. As seen in many of the figures in this section and others, the legs were problematic throughout.

This difficulty was addressed by changing the noise distribution (lower values for σ kept more detail and therefore better segments) and by tightening the tolerance on each frame.

More problematic than obscuring angles was the tendency of the segmenter to produce a good torso in one frame that would be split into two segments in the next. Therefore, when the confidence in a frame was low, the largest and next-largest segments were combined and the confidence remeasured. If the confidence exceeded the required tolerance, the segment would be accepted.

In order to prevent the tracker from over-correcting, creating larger and larger segments, these synthesized segments were accepted at half the normal confidence. This produced good results in the tested sequences.

4.6.3 In Identification

The main problem encountered when identifying the torso was the drift associated with the tracker. Attempting to calculate a weighted average when working with a back window naturally pushes the results back in time. To combat this problem, the bounding box was shifted in both the x and y directions to align it more accurately with the contour of the figure. The distance between the bounding box and the contour in both directions was found; the box was shifted by half of that distance. This had the effect of eliminating most of the drift.

For reasons explained in Chapter 6, the model update step probably needs to be more sophisticated to identify the torso (or other segments) in more complicated data sequences as well.

Chapter 5

DISCUSSION OF RESULTS

The goal of our algorithm is to describe a means by which reasonably general motion tracking may be accomplished. This tracking should be done relatively quickly, without relying on unnecessary or domain-specific information. It should be robust against noise and various kinds of movement.

To that end, the model chosen for the human figure reduced the search space by decreasing the number of degrees of freedom. The algorithm relied on relatively simple image processing and cost metrics to identify and locate the torso throughout each frame. Example output frames are found in Section 5.3.

This chapter presents a critical discussion of the results, examining them in light of the stated objectives in Section 3.1, including successes and failures. The concluding remarks discuss means of continuing the work presented in this thesis, including how to overcome the obstacles encountered.

5.1 Concerning Objectives

5.1.1 Automatic Tracking

A very difficult problem facing most current solutions to the tracking problem is that they are not automatic; sometimes substantial user intervention is required to set up the tracker. Usually this setup involves some kind of machine learning or training.

When we speak of "automatic" or "general" tracking, it is with this ideal in mind: that the figure of interest may be disambiguated from its background and reliably followed through changing conditions without difficulty, and without user intervention.

The use of a low-level model is a necessary element of an automatic tracker; high-level models are too specific and compromise the generality of trackers. Low-level models separate conscious semantic meaning from the tracking process, thereby permitting more flexibility in the face of changing conditions and parameters. At the same time, they simplify the metrics used to locate the torso [15].

Unfortunately, low-level models contain too little detail to provide high-level motion descriptions. By deriving semantic information from low-level models, however, a higher-level description can be obtained without compromising generality. This is demonstrated to an extent by our algorithm, in which a variety of motions are tracked using simple statistical processes.

In particular, Figure 5.1 and Figure 5.3 show the utility of simple metrics in clean and noisy reference frames. Figure 5.5 demonstrates a "broken" frame in which the torso is completely occluded, noise has corrupted the figure, and the torso is moving quickly. However, there is a graceful degradation of the tracker—it simply adjusts to following the biggest segment. (In this particular case, the torso happens to be close to the largest segment; in general, this is not necessarily true.)

5.1.2 Limited Optical Cues

Another concern is that the algorithm handle as many different types of data as possible. The lowest common denominator in visual input for computers is grayscale (or binary) imagery. Old films and surveillance footage are often black and white, and so the algorithm requires no additional color information to function. Moreover, no depth information is required either, though it would be useful in identifying the torso when it is completely occluded.

To a degree, this lack of information may hamper tracking. Lack of depth information, especially, makes occlusion extremely difficult to resolve, witnessed by the multitude of cameras used in motion capture. (Ohio State's Advanced Computing Center for the Arts and Design motion capture lab uses fourteen.) Some researchers have used certain simplifying assumptions and machine learning to attempt to address occlusion with various degrees of success [15, 17, 21, 23]. Lack of color eliminates secondary cues that might be useful in segmenting regions of the body (skin from clothing, for example as in [21]).

In the present solution, occlusion is handled as a case of noise. Color is not considered at all.

5.1.3 Robustness to Noise

A feature of human tracking is substantial robustness to noise in the visual input. The ability to "ignore" extraneous detail is probably assisted by the complex biochemistry of the eye and the brain's ability to make sense of vast visual bandwidth.

It has been suggested that certain visual pathways may have a Gaussian response, so the Gaussian smoothing filter is useful in attempting to emulate the action of the eye. In practice, however, the input is more likely to be corrupted by impulse noise, *i.e.*, spikes in the input signal or salt and pepper noise. Median smoothing filters are often more appropriate to handle this kind of noise.

Holes (background-color discontinuities in the input image) arose from background subtraction and occlusion, where limbs would form an internal boundary on the figure. To combat these features of the input, morphological operations are used to close holes after Gaussian smoothing had been applied.

The result was relatively promising, though not sufficient for production. Large holes require a large structuring element to fill, which results in global figure degradation, as in the dance sequence (Figure 5.5). It would be better to use a more local operation, such as morphological filling [11]. Smaller holes responded well (Figure 5.2). Figure 5.3 demonstrates the corrective effect of morphological operations. The genie sequence was tracked extremely well once morphological operations were used to cover up the holes induced by occlusion and noise.

5.1.4 Speed

The speed of the tracker is difficult to bound tightly. The Hausdorff matching technique used finds cells of interest in the input sequence, and the area of the cells varies widely within a sequence. On a single input image, the algorithm runs in roughly $O(hw)$, where h is the height of the input image and w is its width. Typical processing times ranged between three to seven seconds per frame. The time reported is subject to several different performance bottlenecks.

5.1.4.1 Edge Detection

A significant bottleneck in processing occurs during edge detection, since convolution is computationally expensive. Using the Difference of Gaussians operator improves computation time by reducing smoothing and edge-detection operations to one dimension.

When applying the operator, the delta between the two smoothing distributions was fixed at 0.5. The larger the delta, the wider the detected edges and the more pronounced the smoothing effect. Using a delta of 0.5 produced pixel-width lines.

The morphological operators require a single pass with a two-dimensional convolution kernel. In practice, a 5x5 kernel worked well. Closure first performs erosion and then dilation, so two total passes are required. So the hybrid operator requires four passes with a linear Gaussian kernel and two with a quadratic morphological operation. Other means of improving the performance are suggested in the concluding remarks (see section 6.2).

5.1.4.2 Torso Matching

As previously stated, the matching process runs in $O(m)$ when using the distance transform method [16]. In practice, matching requires the bulk of processing time because the search is brute-force. Even using early acceptance and other techniques, the matching process is computationally intensive.

5.2 Memory Complexity

In general, the memory complexity of the tracker is reasonable. The tracker examines a window of frames to perform updates and therefore must only retain data for that many frames.

A frame consists of a list of segments and an associated confidence. Each segment contains a list of vertices on the contour of the figure and a precomputed axis-aligned bounding box calculated from the vertices. A frame requires space on the order of magnitude of the vertices of the object. Therefore, the memory complexity of the data is linear with respect to the number of vertices.

The input and output images are PPM files, not known for their spatial efficiency. However, the image files are not retained for any length of time in memory and so only contribute to on-disk storage.

5.3 Example Output

Figures 5.1 to 5.7 show various stages of output from the implementation, with appropriate captioning to describe the results shown. Generally speaking, all figures contain input images with output from various stages of the tracking process, including COCONE segmentation and final output.

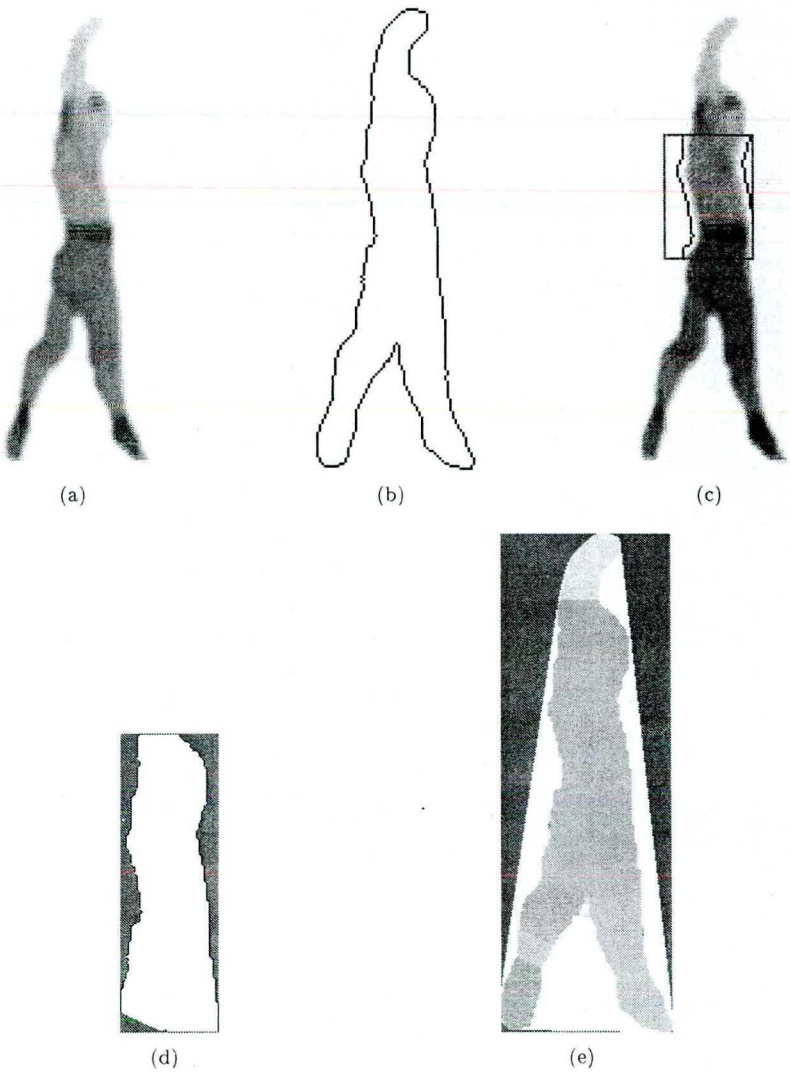


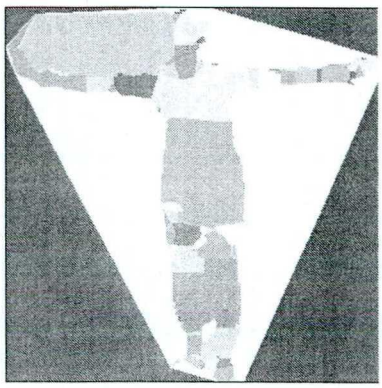
Figure 5.1: Example stages (1).

In this example, $\sigma = 0.5$, so difference of Gaussians was applied without modification. Images have been edited for inclusion in the document; all input and output images are 320x240. (a) shows the input to the tracker, (b) the output, (c) the detected contour, (d) the torso recommended by the segmenter, and (e) the segmentation of the entire body. Note that the recommended torso in (d) contains the head of the dancer, but that the output from the tracker does not.

(a) Input

(b) DoG contour($\sigma = 0.5, t = 128$)

(c) Output induced by DoG



(d) COCONE segmentation (DoG induced)

(e) Torso recommendation (DoG induced)

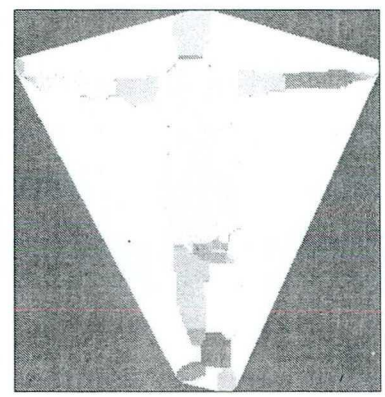
Figure 5.2: Example stages (2).

Note the presence of image artifacts in (a) and how they are detected in (b). (b) is the result of using the DoG operator. The presence of extra noise in the upper-left corner causes the effect seen in (d) and results in a skewed response in (e). Note how it drags the location up to the face!

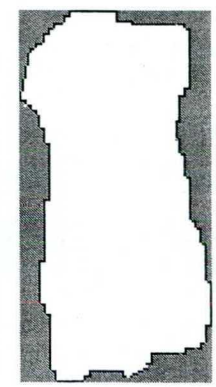
(a) Input image

(b) DoG with morphological operators ($\sigma = 2.0, size = 5 \times 5, t = 128$)

(c) Morphologically induced output.



(d) COCONE-segmented body



(e) Recommended Torso

Figure 5.3: Example stages (3).

The input image (a) is the same as in figure 5.2. Using morphological operators and a relatively aggressive thresholding value produces the "pointilistic" contour of (b). Note that the noise that wrecked the prior example is now gone. (d) demonstrates that COCONE very effectively reconstructs the body contour from these points. (c) shows the corrected result—a significant improvement over the previous attempt.

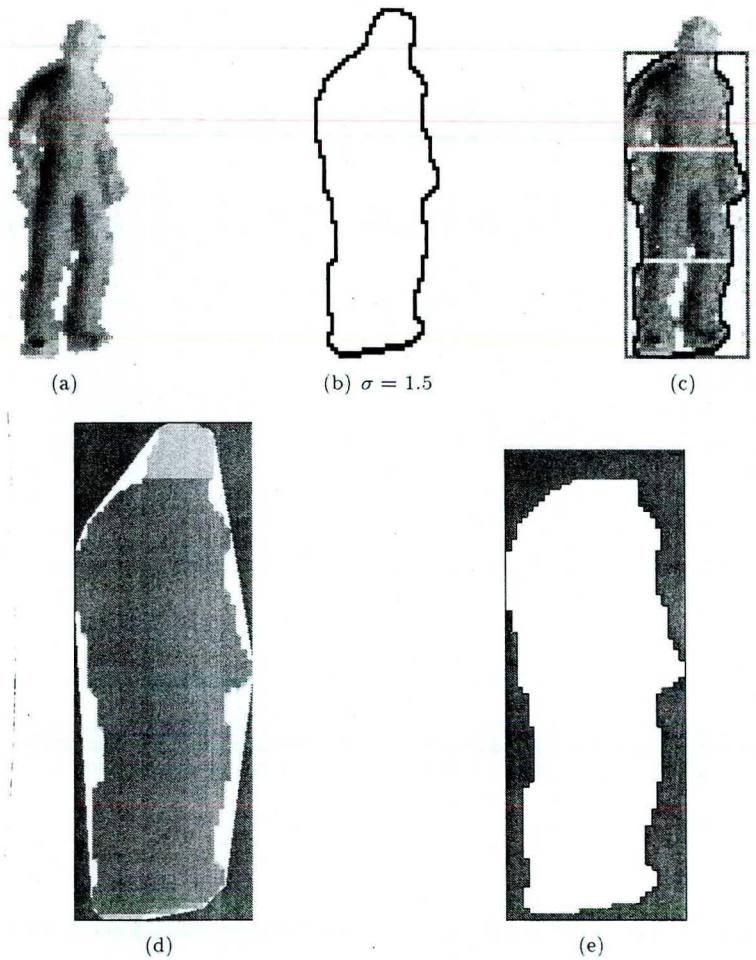


Figure 5.4: Change of depth effect.

(a) is the input image, and (b) is the output from the hybrid morphological operators-difference of Gaussians edge detection. (d) and (e) reveal that this removes too much information for correct segmentation. (c) is the output image with the minimal-area metric (light) and smoothed-corners metric (darker). The latter is obviously a poor decision in this case.

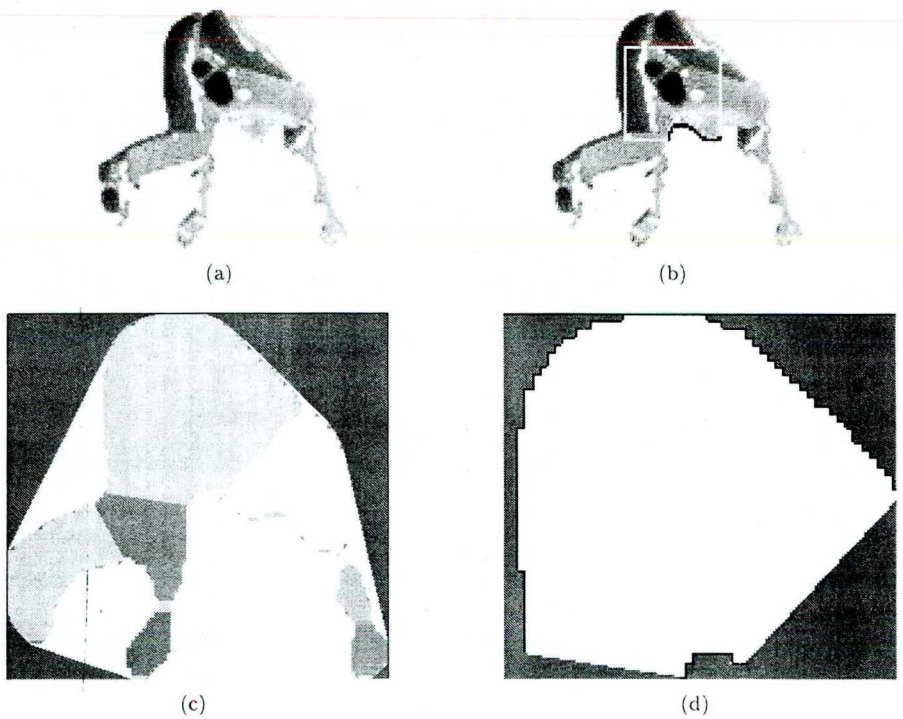


Figure 5.5: Breakdancing.

(a) is the input image, and (b) is the output. While the bounding box corresponds mostly to the location of the torso, it is quite clear from (c) and (d) that the torso is not visible, so the location is rather fortuitous. This sequence in particular was very problematic for tracking because of the noise which corrupted the figure and the silhouette contortions, which make it difficult to isolate necessary detail to extract the torso.

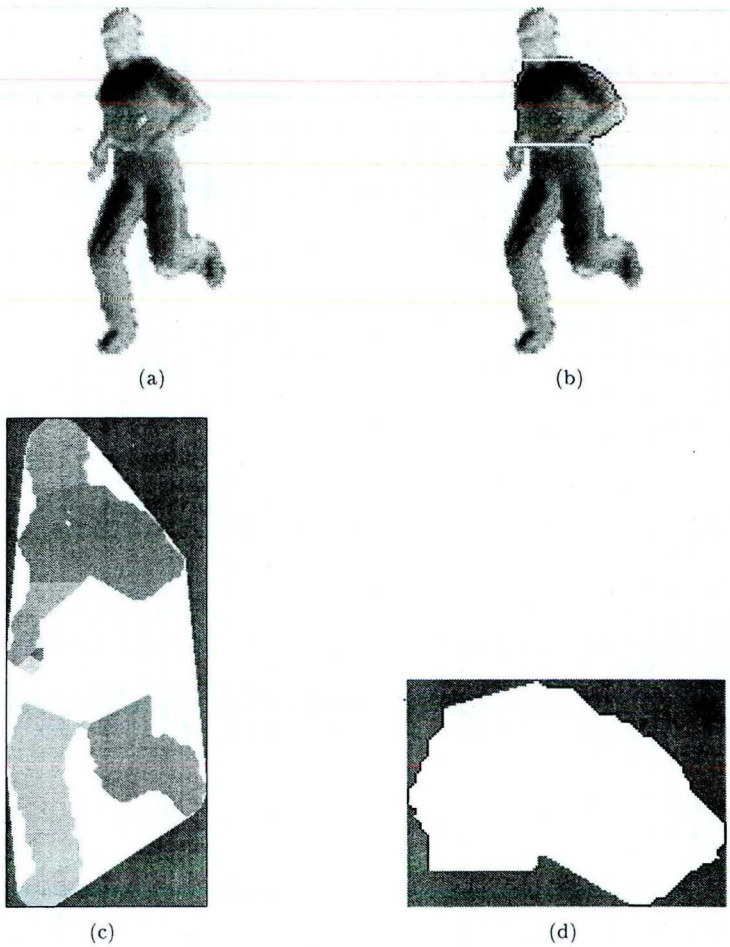


Figure 5.6: Hopping.

(a) is the input image; note that there is good separation between the legs. A good segmentation follows in (c) and (d), resulting in the output seen in (b).

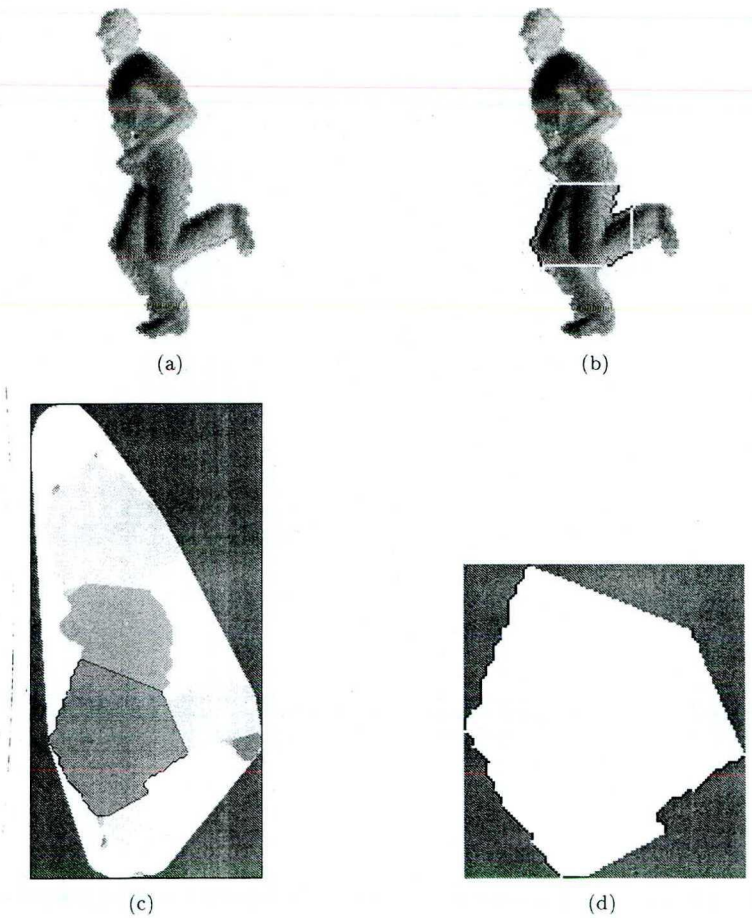


Figure 5.7: Hopping (2).

(a) is the input and (b) the resulting output. This evidences the difficulty mentioned in section 4.6.2, namely that the lack of separation between the legs results in a skewed torso location. In this particular frame, a solution such adaptive torso selection might help, or altering the smoothing parameter (here set at default).

Chapter 6

CONCLUSION

We have presented a novel solution to the problem of tracking the human torso through video. Rather than requiring potentially costly learning processes or user intervention, we accommodate a loose model of the human figure (*i.e.*, that the torso has the greatest area of any other segmented body part) and analytically choose the best approximation to it throughout a sequence of images.

This approach requires little user interaction and performs comparably to both Chen, et al. and Sminchisescu and Triggs. The implementation is straightforward and permits a wide range of inputs due to the generality of the model. There remains room for improvement, however.

Of first importance is that it be extended to full-body tracking. While the speed of the tracker is reasonable, real-time solutions are necessary for many applications (such as surveillance or defense). Moreover, it is unlikely that tracking single figures will be as useful as tracking an arbitrary number. Finally, extremely complex movements are difficult for this tracker. To handle more interesting data, it will need to be extended.

6.1 Full Body Tracking

While tracking the torso is a practical problem, it is likely insufficient for most applications that use tracking technology, especially motion capture systems. To that end, extensions to the algorithm that would accommodate full body tracking are discussed.

We have presented a system that will track the human torso by using statistical methods combined with area-based segmentation information. Using the area confidence is more general than stick figures, blobs, or other models, and may be used fairly effectively to find the torso.

The following sections describe how to track other segments using different criteria and how to track multiple segments at a time.

6.1.1 Location Confidence

The algorithm may be generalized when applied with a corresponding feature: the location confidence. While the torso is usually the largest segment—or at least a part of the largest segment—the relationship between areas is not always consistent. The head, for example, is reliably segmented in almost every single frame of the walk sequence, but the area of the head in relation to the area of the other segments varies quite a bit, and so it is difficult to assign it a consistent position in the list of segments. It may appear last in one frame and fourth in another.

This motivates the need for a location confidence, similar to the area metric already described. In fact, it may be calculated exactly the same way. The location is determined by the centroid of the segment, which may be found either by applying Heron's method or by calculating the centroid of the bounding box of the segment. The former is usually more reliable, but the latter is more efficient. A window of observations is found, the standard error of the mean calculated, and the location confidence assigned in the same way as the area confidence:

$$lc_i = \begin{cases} 1 - SEM/\mu_l & , l_i \leq \mu_l \pm t * \sigma_l \\ 0 & , l_i \geq \mu_l \pm t * \sigma_l \end{cases} \tag{6.1}$$

where lc_i is the location confidence for the i-th frame, μ_l is the mean location for the window of frames, and σ_l is the standard deviation of the locations. The variable t retains its normal interpretation—the tolerance in standard deviations we are willing to extend to outliers.

6.1.2 Segment Integration

It is unlikely that the location confidence by itself will provide enough information to perform full body tracking. Each of the segments should be considered, because a correlation should be made between the segment location and area so that disparate recordings may be reconciled.

Using the area confidence alone permitted relying upon the sorted area to correctly locate the torso (after some statistical adjustments and combinations, of course). Such a strategy will not work as effectively for full body tracking because the recorded area of the segments may vary enough that selecting the smallest segment consistently (*e.g.*, assuming that the head is the smallest segment) could lead to incorrect results.

Instead, it is necessary to examine each of the segments by location and area and select the one from the list that provides the best confidence in a match. In other words, the *position in which a segment occurs in the list should not be an immediate factor in the analysis, despite the fact that the segments are sorted by area.* The relative importance of the metrics may vary from application to application, and how to combine them is not immediately clear. However, it seems logical to treat the two features as dependent probabilities and treat them accordingly.

6.1.3 Initialization

Tracking other segments using both confidences is a little more difficult; additional constraints may be needed to initialize the frame. For example, if one wished to track the head, the initial segment location may have the largest y-value. Deciding which segment is the head based on area is more difficult. Additionally, it may be necessary to assume that the figure is standing erect with arms at the side.

Such constraints are not particularly binding and do not affect the model at all. Moreover, they vary from application to application. It is equally plausible to create a more interactive tracker initialized by the user. So, while initialization may be somewhat more complicated, the tracker should remain fairly flexible.

6.1.4 Multiple Segment Tracking

Once it is possible to track an arbitrary segment, tracking them all at once is not particularly difficult. Statistical information about each segment must be collected and stored. Record-type data structures should be sufficient to store this information.

In particular, because the area of the segment may vary frame to frame, it is important to search through each segment in the list to find an appropriate match, since the ranking of each segment will vary from frame to frame. This is caused by changes in pose, noise, and other considerations.

The relative size of the segment and quality of the sample will influence how easy it is to track, and adjustments to the tolerance will probably be necessary in difficult situations.

6.2 Real-Time Location

Real-time tracking (in this case, locating the torso in real time) proved difficult due to the nature of the image processing operations required. There are several possible solutions to this problem. The most viable is to implement as many operators in graphics hardware as possible. Real-time image processing libraries such as Intel's OpenCV [1] may be used to enhance performance. Matching is also a bottleneck, but Huttenlocher and Rucklidge suggest several heuristics that may improve its performance, namely early rejection and acceptance [16].

An intriguiging possibility for leveraging graphics hardware to perform Sobel edge detection is outlined in [2]. Since the Canny detector finds edges by using the Sobel kernels initially, it seems likely that it could be easily adapted to this routine.

The principle difficulty with implementing these routines is that they require additional input in the form of a depth map or normal map, neither of which is likely to be available for the input images. (If the depth map is available, 3-D motion capture data should be straightforward. The normal map must usually be specified with each of the coordinates—it is not derivative information.)

It seems, however, that implementing the difference of Gaussians operator in hardware should be considerably easier. Input images and the kernels could be passed as textures to a GPU shader, many of which have a hard-wired convolution function. Computing the difference should proceed quickly enough.

6.3 Tracking Multiple Figures

The algorithm assumes that only one figure appears in the input sequence. While useful for some applications, it is likely that most applications would capture interaction. In order to do this, the algorithm needs to be extended in two ways. The first is to identify separate figures in the input sequence; the second is to handle occlusions.

6.3.1 Figure Classification

The first problem is motivated by the implementation the Flow Discretization Algorithm, which assumes that every vertex reported is on the contour of a single figure. When the contours are separate, as in Figure 4.1.f, a segment is found that bridges the contours. Not only is this inaccurate, when the figures are sufficiently separated, the "phantom segment" may be the largest one, and therefore incorrectly interpreted to be the torso.

A simple solution to this is to use connected components labeling or another pixel classification algorithm to create two binary image masks. The two figures can be extracted and analyzed separately.

6.3.2 Figure Occlusion

The second problem, when figures occlude one another, is somewhat more problematic and would probably require some form of machine learning. Even then, without 3-D model data, it will be extremely difficult to solve this problem in any extended sequences. It seems likely that automatic motion capture will be achieved before handling figure interaction in two dimensions.

6.4 Multiple Movement Types

As noted in Section 4.3, complicated movements may cause breakdowns in the tracking process. In the dance sequence, for instance, the torso moves quickly in all three dimensions and is fully occluded in several frames.

It is likely that complex sequences like this one will require machine learning. Ho's technique[15] could be applied fairly well to these sequences; it should also be possible to use a more complicated update technique that relies less on the bounding box of the contour vertices and more on the vertices themselves.

6.5 Summary

Despite the simplicity of this algorithm, it recommends a new way of thinking about the tracking problem. By limiting the specificity of the model and deriving information from the geometry of the contour, we have demonstrated that it is possible to track the human torso without semantically

identifying it as such. This allows substantially more flexibility than many current solutions whose models are so specific as to limit them to a particular domain [21] or that require extensive learning processes for each motion learned [17].

Bibliography

[1] Opencv. http://sourceforge.net/projects/opencvlibrary/.

[2] T. Akenine-Möller and E. Haines. *Real-time Rendering*. A K Peters, Natick, Massachusetts, USA, 2002.

[3] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[4] J. Brey. Markerless based human motion capture: A summary. http://www.visicast.co.uk/members/move/Partners/Papers/MarkerlessSurvey.pdf.

[5] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), November 1986.

[6] Y. Chen, J. Lee, R. Parent, and R. Machiraju. Markerless monocular motion capture using image features and physical constraints. *Computer Graphics International 2005*, June 2005. To appear.

[7] D. Coombs and C. Brown. Real-time binocular smooth pursuit. *Interjatioanl Journal of Computer Vision*, 11(2):147–164, 1993.

[8] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *Proc. Conf. Computer Vision and Pattern Recognition*, volume 2, pages 1144–1149, 2000.

[9] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In *Workshop on Algorithms and Data Structures (WADS)*, 2003.

[10] T. K. Dey and S. Goswami. Tight cocone: A water tight surface reconstructor. In *Proceedings of the 8th ACM Symposium on Solid Modeling Applications*, pages 127–134, 2003.

[11] B. Fisher, S. Perkins, A. Walker, and E. Wolfart. Morphology - dilation, 2003. http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm.

[12] D. Fox, J. Hightower, L. Liao, D. Shulz, and G. Borriello. Bayesian filters for location estimation. *IEEE Pervasive Computing*, 2003.

[13] P. Fua, A. Gruen, N. D'Apuzzo, and R. Plänkers. Markerless full body shape and motion capture from video sequences. *International Archives of Photogrammetry and Remote Sensing*, 34(5):256–261, 2002.

[14] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[15] J. Ho, K.-C. Lee, M.-H. Yang, and D. Kriegman. Visual tracking using learned linear subspaces. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[16] D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the hausdorff distance, 1992. http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cs/TR92-1321.

[17] M. Isard and A. Blake. Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 1998.

[18] O. Javed, K. Shafique, and M. Shah. A hierarchical approach to robust background subtraction using color and gradient information. In *MOTION '02: Proceedings of the Workshop on Motion and Video Computing*, page 22, Washington, DC, USA, 2002. IEEE Computer Society.

[19] K. Kim. Umd background subtraction program, October 2003. http://www.umiacs.umd.edu/ knkim/UMD-BGS/.

[20] M. W. Lee and I. Cohen. Proposal maps driven mcmc for estimating human body pose in static images.

[21] V. Mamania, A. Shaji, and S. Chandran. Markerless motion capture from monocular videos.

[22] D. Pritchard and W. Heidrich. Cloth motion capture. *EuroGraphics*, 22(3), 2003.

[23] F. Remondino and A. Roditakis. Human figure reconstruction and modeling from single image or monocular video sequence. *4th International Conference on 3-Dimensional Imaging (3DIM)*, 2003.

[24] H. Sidenbladh, M. J. Black, and D. J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. *European Conference on Computer Vision*, 2000.

[25] C. Sminchisescu. Consistency and coupling in human model likelihoods. *IEEE Conference on Autoamtic Face and Gesture Recognition*, 2002.

[26] C. Sminchisescu and B. Triggs. Estimating articulated human motion with covariance scaled sampling. *International Journal of Robotics Research*, 22(6):371–393, 2003.

[27] D. Tweed and A. Calway. Tracking many objects using subordinated condensation. 2003.

[28] "Eric W. Weisstein". "gram-schmidt orthonormalization". "http://mathworld.wolfram.com/Gram-SchmidtOrthonormalization.html".

[29] "Eric W. Weisstein". "heron's formula". "http://mathworld.wolfram.com/HeronsFormula.html".

[30] G. Welch and G. Bishop. An introduction to the kalman filter, August 2004. http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.