

Skeleton-Based Pose Estimation of Human Figures

Dual Degree Project Report

Submitted in partial fulfillment of the requirements
for the degree of

**Bachelor of Technology
Master of Technology**

by

**Lakulish Antani
Roll No: 03D05012**

under the guidance of

Prof. Sharat Chandran



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Acknowledgement

I would like to thank **Prof. Sharat Chandran** for his guidance and support in making this project an informative and rewarding learning experience for me. I would also like to thank **Appu Shaji** for helping give direction to the project, and his invaluable input and help with the implementation, without which this work would not have been possible.

Lakulish Antani

Contents

Abstract	1
1 Introduction	1
1.1 Related Work	2
1.2 Challenges	3
1.3 Contributions	3
2 Pictorial Structures	5
2.1 The Model	5
2.1.1 The Definition	5
2.1.2 A Statistical Interpretation	6
2.2 Learning the Parameters	8
2.3 Computing the MAP Estimate	9
2.3.1 An Efficient Algorithm	9
2.3.2 The Generalized Distance Transform	10
2.4 Sampling the Posterior	10
3 A Bottom-Up Approach	13
3.1 Contour Detection	13
3.1.1 Probability of Boundary	13
3.1.2 Hysteresis Thresholding	16
3.1.3 Boundary Completion	16
3.2 Detecting Parts	18
3.3 Solving the IQP	18
3.3.1 Learning the Constraints	18
3.3.2 Constructing the IQP	19
4 The Skeleton	21
4.1 Closed Contours: Ratio Contour	22
4.1.1 A Polynomial Time Solution	23
4.1.2 Possible Improvements	25
4.2 Medial Axis Transform	25
4.3 Skeleton Pruning	26
5 Skeleton-Based Pose Estimation	28
5.1 Extracting Parts	28
5.1.1 Skeleton Fragments	28
5.1.2 Inverse Medial Axis Transform	29
5.2 Structuring the Part Set	29

5.2.1	Fragment Hierarchy	30
5.2.2	Splitting Fragments	31
5.3	A Top-Down Framework	32
5.3.1	Appearance Costs	32
5.3.2	Structure Costs	33
6	Results	36
6.1	Implementation Details	36
6.1.1	Dataset	36
6.1.2	Classic Pictorial Structures	36
6.1.3	CDT/IQP-based Algorithm	36
6.1.4	Skeleton-Based Pose Estimation	37
6.2	Experimental Results	37
6.2.1	Evaluation Strategy	38
6.2.2	Results	38
7	Conclusion	44
7.1	Discussion	44
7.2	Future Work	44
	Bibliography	46

List of Figures

1.1	Overview of the pose estimation process.	2
2.1	Tree structure of the human body	6
3.1	Stages of preprocessing	14
3.2	Example of a Pb map.	16
4.1	Utility of the skeleton as a shape descriptor.	21
4.2	Block diagram of the skeleton extraction process.	21
4.3	Benefit of using hysteresis thresholding.	22
4.4	Salient boundaries and simple alternate cycles.	23
4.5	Reduction 3 in Ratio Contour.	25
4.6	Example of a closed contour detected by Ratio Contour.	25
4.7	Pruning an example skeleton.	26
4.8	Complications caused by concave vertices.	27
5.1	Splitting skeletons at junction pixels.	29
5.2	Angle and length as cues for skeleton hierarchy	30
5.3	Two example splitting pixels.	32
5.4	Integral images	34
5.5	Joint positions for two parts	34
6.1	The learned human model.	37
6.2	Performance comparison	39
6.3	Results: test image 1	40
6.4	Results: test image 2	41
6.5	Results: test image 1 with background subtraction	42
6.6	Results: test image 2 with background subtraction	43

List of Tables

- 6.1 Comparison between our pose estimation algorithm and classic pictorial structures. 38
- 6.2 Comparison between skeleton-based part detection and CDT-based part detection. 39

Abstract

Pose estimation of human figures is a challenging open problem. Model-based approaches, which can incorporate prior knowledge of the structure and appearance of human figures in various poses are the most promising line of research in this area. Most models of the human body represent it in terms of a tree of interconnected parts. Given such a model, two broad classes of pose estimation algorithms exist: top-down and bottom-up. Top-down algorithms locate body parts in a top-down order with respect to the tree of parts, performing a structure-guided search. Bottom-up algorithms, on the other hand, first look for potential parts irrespective of which parts they may be (usually based on local image properties such as edges), and then assemble a human figure using a subset of these candidate parts. Both approaches have pros and cons, and there are compelling reasons to develop a hybrid approach.

We describe a model-based pose estimation algorithm that combines top-down and bottom-up approaches in a simple manner. We describe a bottom-up part detector based on the skeleton transform, and a skeleton computation pipeline that uses existing algorithms for computing a pruned skeleton transform of any image. We describe a top-down pose estimation algorithm based on pictorial structures which we combine with the skeleton-based part detector. We also describe a way of imposing a structure on the space of candidate parts by computing a hierarchy between skeleton fragments, and use this structure to facilitate pose estimation. We compare our pose estimation algorithm with the classic pictorial structures algorithm, and compare our skeleton-based part detector with another edge-based part detector, and provide ideas for improving our method.

Chapter 1

Introduction

An important open challenge in computer vision is the problem of building a reliable system that can track people and their poses. There are many difficulties that one faces when attempting to solve this problem. The motion and pose of a human figure is not easy to predict. Consider the problem of tracking a figure skater or gymnast; the motion is quite different from the more common human motions such as walking or running, and the pose may vary greatly from frame to frame. Moreover, people's appearances and clothing vary greatly from instance to instance, making the problem of locating the human figure in an image or image sequence complicated. A successful human tracking and pose estimation system has immense utility: the possible range of applications includes automated surveillance, providing input to structure-from-motion algorithms, and enabling novel approaches to human-computer interaction.

As a result, there is a rich body of literature devoted to this problem; such as the highly popular KLT tracking algorithm by Kanade et al. [11, 18] However the resulting systems typically require some constraints on the input, such as multiple cameras, special coloured markers or controlled backgrounds. The KLT tracker, for example, does not use any global information such as the structure or appearance of, say, a human figure, and hence it performs relatively poorly when attempting to track human motion. We would like a tracking system that requires input from only a single camera and performs tracking and pose estimation in previously unknown surroundings (even in the presence of a moving background). We would also prefer the ability to track multiple people in the same image sequence, as well as the ability to handle occlusions.

The standard approach to incorporate knowledge about humans into a pose estimation system is to use a model of the appearance and structure of the human body. Typically, the human body is described in terms of a set of parts and joints between them. The appearance of parts and the structural constraints at joints are represented in terms of cost functions defined on the parts and joints, respectively. These cost functions are in turn described in terms of some parameters, which we need to learn from a set of training images. Thus, at its core, any model-based pose estimation algorithm relies on machine learning techniques and a suitable training set.

In this work, we describe a pose estimation algorithm that allows us to infer the pose of human figures from a single image. While a repeated application of this algorithm does allow us (at least in theory) to track humans (and their poses) through an image sequence, we do not make use of inter-frame coherence or motion models to enhance the tracking output. We leave such modifications to future work.

Our basic method is as follows. We observe that the skeleton of a human shape is highly representative of pose. Therefore, we first compute a noise-free skeleton transform of the input image. We use this skeleton information as a part detector. (See Chapter 4 for details.) The

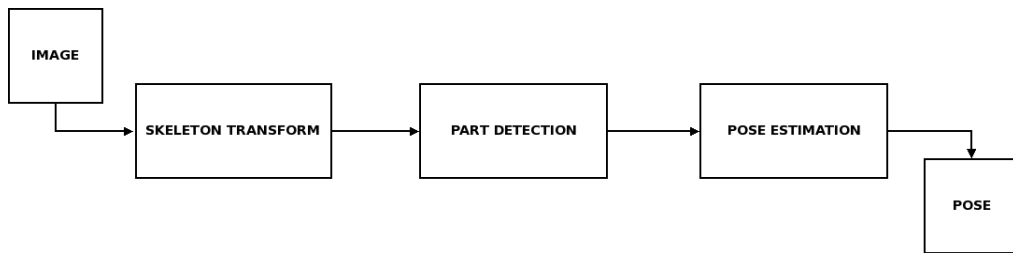


Figure 1.1: Overview of the pose estimation process.

candidate parts returned by this detector are used as input to a model-based pose estimation algorithm. (See Chapter 5 for details.) We also use the skeleton information to impose a structure on the space of candidate parts, which is then exploited by the pose estimation phase. Figure 1.1 shows an overview of our pose estimation process.

1.1 Related Work

Pose estimation algorithms for human figures are a widely studied and highly active area of research. Most algorithms treat the human body as a tree of connected parts, with the vertices representing parts and the edges representing joints. The root is typically taken to be the torso. Most existing algorithms can typically be classified as either top-down or bottom-up.

A top-down algorithm tries to locate the torso first, followed by the parts connected to the torso, and so on down the tree. One of the most popular such approaches is the pictorial structures algorithm [9], which we shall describe in detail. The advantage of such an approach is that the search is guided by the structure of the human body. However, one of the main disadvantages of such an approach is that since there is no knowledge of where potential parts exist in the image, the search space is typically very large, since a search must be performed over all positions, scales and orientations.

In contrast, a bottom-up algorithm begins by detecting potential parts (irrespective of *which* parts they may be) in the image, and then builds up pose estimates going upwards from the bottom of the tree. This process may be expressed as parsing an image [19, 15], or as solving some other form of optimization [16]. We shall describe a bottom-up approach that uses the constrained Delaunay triangulation for part detection and expresses pose estimation in terms of solving an integer quadratic program [16]. The main advantage of such an approach is that the optimization phase has a small search space. The disadvantage is that the search is not guided by structure, since no structure information is available after part detection.

Combining both approaches can lead to the best of both worlds. Many approaches have been documented in the literature [10, 4], several of them describing a combined segmentation and pose estimation system. However, we shall describe a simplistic combination of the top-down and bottom-up approaches: use a part detector to reduce the size of the search space, and then employ a structure-guided, top-down pose estimation algorithm.

Our part detector is based on the skeleton transform. For our purposes, we require a skeleton without spurious branches introduced by noise. Several skeleton pruning algorithms exist, some of which compute a multi-resolution hierarchy of skeletons [3, 20, 14]. We describe a skeleton pruning algorithm based on discrete curve evolution [3]. In order to use such an algorithm, we need a way to compute closed contours of salient objects in the image. Several algorithms exist to solve this problem [22, 6, 7], we describe one of them, Ratio Contour [22].

A completely different approach to pose estimation is the exemplar-based approach, where

a set of example poses are compared against the input image and the best match is found. Among existing algorithms that use this approach, some express the exemplars in terms of the shape skeleton [8, 5], and we can combine our skeleton pipeline with such an algorithm as well. Although such an approach has several unique advantages (in particular, 3-dimensional pose estimation is simplified greatly: each exemplar is associated with a 3-dimensional pose), we believe the disadvantage of explicitly storing all “possible” poses outweighs the advantages, and hence we do not pursue such an approach.

1.2 Challenges

A successful model-based pose estimation algorithm has to overcome several challenges. We shall now outline some of the issues that need to be addressed. Firstly, the search space which needs to be explored in order to determine the pose can grow to be very large. This makes many pose estimation algorithms which consider all possible part locations scale poorly with image resolution. In the case of exemplar-based algorithms, a large search space leads to correspondingly greater storage requirements as well.

We have already mentioned that a bottom-up part detector can be one way to reduce the size of the search space. This requires us to choose informative features using which to describe potential parts. Many pose estimation systems have been described in the literature which use edge-based part detectors. However, it is not easy to extract structural information about the candidate part based on edge information alone. Moreover, noise in the image can lead to faulty edge detection, which can greatly complicate the process of part detection.

At its core, a model-based pose estimation algorithm may be divided into a series of processing steps, each of which typically uses a machine learning technique to choose its output. For example, a part detector may use a classifier based on a learned part model in order to select a set of candidate parts given a set of edges in the image. Any such approach will lead to false positives and false negatives. In practice, the greater problem is that of false negatives, since they involve the loss of useful information. Therefore, we may be forced to design our processing steps in a manner which reduces false negatives at the cost of increasing false positives. Dealing with the resulting large number of false positives is another challenge in itself.

In the rest of this report, we shall explore several ways of tackling the above issues, and see to what extent they succeed.

1.3 Contributions

The main contributions of this work are as follows:

- We have built a system to construct pruned (and hence noise-free) skeletons based on salient boundaries in an image, and a parts detector based on the skeleton.
- We have combined this bottom-up detector with a structure-guided, top-down pose estimation algorithm (pictorial structures). The resulting algorithm is in some sense a hybrid between the top-down and bottom-up approaches.
- We have described a method to extract hierarchical relationships between skeleton fragments, in order to impose a structure on the search space for pose estimation. The pose estimation phase exploits this structure while locating parts.

- We have designed a pose estimation system based on pictorial structures that does not require background subtraction (a strong restriction) in order to compute the cost of a part configuration.
- We have compared the skeleton-based pose estimation algorithm with some existing pose estimation algorithms, with respect to performance and accuracy of the estimated pose.

The rest of this report is organized as follows. In Chapter 2, we describe the pictorial structures framework, which is one of the standard top-down pose estimation algorithms. Next, in Chapter 3, we describe a bottom-up pose estimation algorithm based on the constrained Delaunay triangulation (CDT) and integer quadratic programming (IQP). Before moving on to describe our pose estimation algorithm, we describe our pruned skeleton computation pipeline in Chapter 4. In Chapter 5, we detail the various aspects of our skeleton-based pose estimation algorithm. Finally, in Chapter 6, we describe our implementation of the algorithm and show some results, and compare performance with standard pictorial structures and the IQP-based algorithm.

Chapter 2

Pictorial Structures

One of the more popular frameworks for object recognition (particularly in the case of objects with articulated parts), is the pictorial structures approach [9]. We shall now describe the general features of the pictorial structure model as applied to human pose estimation. The basic pictorial structures algorithm involves an energy minimization (which can be described in terms of a *maximum a posteriori* (MAP) estimation problem). We shall describe an efficient algorithm for performing this energy minimization and estimating the pose of human figures in a single image. However, the energy minimization approach detects only the best match of the human model in an image (by design). Since we would like to estimate the pose of multiple human figures in a single image, we next describe an approach to sample multiple matches for the human model in order to locate multiple humans. However, we leave specifics of the model (in particular, the definitions of the cost functions) for a later chapter (Chapter 5).

2.1 The Model

The object to be detected is assumed to have multiple parts, with connections between some pairs of parts. The set of possible locations for any part is called the *location space* \mathbb{L} . (Note that \mathbb{L} need not be the actual space in which the object exists, such as \mathbb{R}^2 or \mathbb{R}^3 .) The location of the entire object is completely specified by the locations of all its parts. So for example, if the number of parts is n , then the location of the object is specified by (l_1, \dots, l_n) , where $\forall i \in [1, n], l_i \in \mathbb{L}$.

The pictorial structures framework is quite general, and is not restricted to human pose estimation. For example, it has been used successfully for face detection [9]. For our purposes, however, we shall assume that the human body is made of 10 parts (as shown in Figure 2.1), arranged in a tree structure with the torso at the root.

2.1.1 The Definition

Roughly speaking, a pictorial structure is a graph with a set of energy functions associated with it. A more formal definition follows.

Definition 1 Consider a graph $G = (V, E)$. With each vertex $v_i \in V$ associate a function $m_i : \mathbb{L} \rightarrow \mathbb{R}$, and with each edge $e_{ij} = (v_i, v_j) \in E$ associate a function $d_{ij} : \mathbb{L} \times \mathbb{L} \rightarrow \mathbb{R}$. The graph G combined with the associated functions are collectively termed a *pictorial structure*.

Each vertex v_i corresponds to a part of the object, and each edge $e_{ij} = (v_i, v_j)$ corresponds to a connection between the parts corresponding to vertices v_i and v_j . The value $m_i(l_i)$ is the

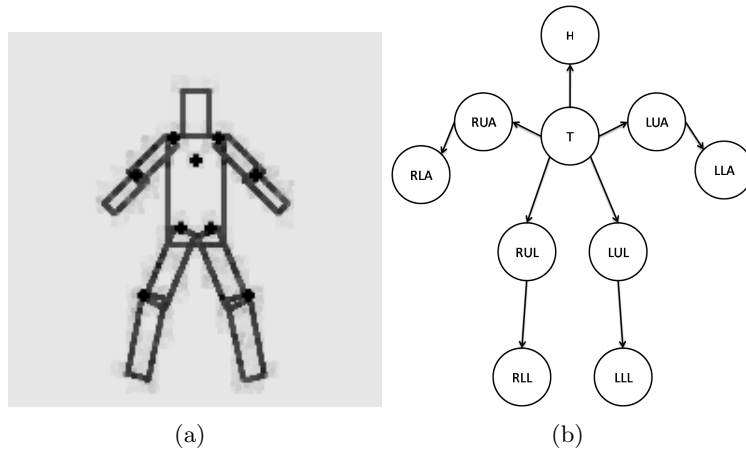


Figure 2.1: The tree structure of the human body. (a) Rectangles represent parts, and adjacent rectangles are connected parts. (b) This naturally leads to a tree structure with the root at the torso.

cost of locating part i at location l_i , based on the appearance of the image at l_i . The value $d_{ij}(l_i, l_j)$ is the cost of locating parts i and j at l_i and l_j respectively, based on their relative locations. (Typically, this is some measure of distance between l_i and l_j .)

Again, suppose the number of parts is n (i.e., $|V| = n$). Then the location of the object can be specified by an n -tuple $L = (l_1, l_2, \dots, l_n)$. To perform object recognition given an image, we must find the “best” location L^* for the object. In the pictorial structure framework, this involves solving the following energy minimization:

$$L^* = \arg \min_L \left(\sum_{i=1}^n m_i(l_i) + \sum_{(v_i, v_j) \in E} d_{ij}(l_i, l_j) \right) \quad (2.1)$$

In the next section, we will describe a statistical framework within which the above energy minimization can be rewritten as a MAP estimation problem.

2.1.2 A Statistical Interpretation

The above energy minimization view has some major drawbacks:

- There is no simple way to specify the cost functions.
- There is no obvious method of learning these functions given a set of training examples.
- There is no way of detecting multiple good instances of the model in the same image, since the minimization locates a single “best” match of the model.

All of these issues can be resolved by introducing a statistical framework in which to interpret the pictorial structure model. Since we want to determine locations L given an image I , we consider the following probability distribution:

$$\Pr(L | I, \theta) \propto \Pr(I | L, \theta) \Pr(L | \theta) \quad (2.2)$$

where $\theta \in \Theta$ is a set of parameters describing the pictorial structure model. Using standard Bayesian terminology [13], $\Pr(L | I, \theta)$ is the *posterior* distribution, $\Pr(I | L, \theta)$ is the *likelihood*

and $\Pr(L | \theta)$ is the *prior*. The likelihood models the probability of the image given an object configuration L , based on the appearance of the image, much like the m_i functions described earlier. The prior models the probability distribution of all possible object configurations, in a way performing the role of the d_{ij} functions described earlier. The posterior models the probability distribution of object configurations given the input image, and hence is the distribution that is most useful to us.

The above proportionality can be rewritten as follows:

$$-\log \Pr(L | I, \theta) = -\log \Pr(I | L, \theta) - \log \Pr(L | \theta) + \text{constant} \quad (2.3)$$

Putting $-\log \Pr(I | L, \theta) = \sum_{i=1}^n m_i(l_i)$ and $-\log \Pr(L | \theta) = \sum_{(v_i, v_j) \in E} d_{ij}(l_i, l_j)$, it can be seen that:

- The energy minimization problem reduces to a MAP estimation problem.
- The parameters for the distributions can be obtained by computing the max-likelihood estimator of θ .
- Multiple good matches for the model can be obtained by repeatedly sampling the posterior distribution.

It is assumed that the likelihood parameters and the prior parameters are independent: $\theta = (u, c, E)$. Here, u is a set of per-part appearance parameters, c is a set of per-edge connection parameters and E is the set of edges in the graph.

It is also assumed that the individual part likelihoods are independent, i.e.:

$$\Pr(I | L, u) = \prod_{i=1}^n \Pr(I | l_i, u_i) \quad (2.4)$$

Note that this is only an approximation, and can be bad if parts overlap, in which case multiple parts generate the same portion of the image. From the above, it can be seen that $m_i(l_i) = -\log \Pr(I | l_i, u_i)$.

It is also assumed that the graph G is a tree. (If it is not a tree, we can compute a tree-structured edge set using some standard minimum spanning tree algorithm, as described in Section 2.2.) In terms of a tree-structured Markov random field, the prior can now be written as:

$$\Pr(L | \theta) = \frac{\prod_{(v_i, v_j) \in E} \Pr(l_i, l_j | \theta)}{\prod_{v_i \in V} \Pr(l_i | \theta)^{\deg(v_i)-1}} \quad (2.5)$$

The distribution over absolute locations is assumed to be uniform, and hence the denominator can be ignored. This gives:

$$\Pr(L | c) = \prod_{(v_i, v_j) \in E} \Pr(l_i, l_j | c_{ij}) \quad (2.6)$$

From the above, it can be seen that $d_{ij}(l_i, l_j) = -\log \Pr(l_i, l_j | c_{ij})$.

In the next section, we will see how to use this statistical framework for the learning parameters of a pictorial structure model, and an efficient algorithm for computing the MAP estimate.

2.2 Learning the Parameters

Before we can use a pictorial structure model for object recognition, we need to learn its parameters from a set of training examples. We will now outline the basic approach for doing so, without going into the specifics of the parameters themselves.

The problem is as follows. We are given a set of m training instances, in the form of a set of images $\{I^1, \dots, I^m\}$ and a corresponding set of object locations $\{L^1, \dots, L^m\}$. We need to compute the parameters $\theta = (u, c, E)$. These parameters are computed using the maximum likelihood estimation technique. Since the likelihood and prior parameters are independent, the MLE technique can be applied separately to the likelihood (to compute u) and the prior (to compute c and E).

For the appearance parameters u , this gives:

$$u^* = \arg \max_u \prod_{k=1}^m \Pr(I^k | L^k, u) \quad (2.7)$$

Since the individual part likelihoods are assumed to be independent, it is easy to see that the individual parts' appearance parameters can be computed separately, as follows:

$$u_i^* = \arg \max_{u_i} \prod_{k=1}^m \Pr(I^k | l_i^k, u_i) \quad (2.8)$$

Similarly for the connection parameters c :

$$c^* = \arg \max_c \prod_{k=1}^m \Pr(L^k | c) \quad (2.9)$$

Again, the c_{ij} 's can be computed independently:

$$c_{ij}^* = \arg \max_{c_{ij}} \prod_{k=1}^m \Pr(l_i^k, l_j^k | c_{ij}) \quad (2.10)$$

The edge set E needs to describe a tree. In case the connections between parts of the object naturally form a tree structure, it is not necessary to learn E^* : the natural edge set can be used. However, if the connections form a more general kind of graph, it is necessary to learn an edge set E^* which forms a tree. To do this, each edge e_{ij} in the original graph is assigned the weight:

$$w_{ij} = \prod_{k=1}^m \Pr(l_i, l_j | c_{ij}^*) \quad (2.11)$$

The minimum spanning tree of the pictorial structure graph is then computed using any standard algorithm. E^* is defined to be the edge set of the resulting spanning tree.

Note that for our purposes, E is fixed, since we explicitly model the human body as a set of parts and known joints between them. Therefore, we do not need to perform the above minimum spanning tree computation when learning the model parameters.

In the next section, we will describe an efficient algorithm to compute the MAP estimate for the pictorial structure model posterior.

2.3 Computing the MAP Estimate

For the sake of brevity, we will describe the procedure of computing the MAP estimate in terms of the original energy minimization problem. Solving the energy minimization problem for general graphs is an NP-hard problem. However, in the special case where the graph is a tree, the minimization can be performed in polynomial time using a dynamic programming algorithm.

2.3.1 An Efficient Algorithm

We shall now see how the minimization can be performed efficiently. $B_j(l_i)$ is defined to be the cumulative energy for the best locations of vertex v_j and the subtree rooted at it given the position l_i for its parent v_i . This includes both the appearance score for v_j and the connection score for the edge to its parent. Therefore, if v_j is a leaf node:

$$B_j(l_i) = \min_{l_j} (m_j(l_j) + d_{ij}(l_i, l_j)) \quad (2.12)$$

For any internal node v_j other than the root, this gives the following recursive expression:

$$B_j(l_i) = \min_{l_j} \left(m_j(l_j) + d_{ij}(l_i, l_j) + \sum_{v_c \in C_j} B_c(l_j) \right) \quad (2.13)$$

where C_j denotes the set of children of vertex v_j . Finally, for the root v_{root} :

$$B_{root}(l_i) = \min_{l_{root}} \left(m_{root}(l_{root}) + \sum_{v_c \in C_{root}} B_c(l_{root}) \right) \quad (2.14)$$

It is assumed that there exists a procedure to compute not only $B_j(l_i)$ but $l_j^*(l_i)$, which is the best position of v_j given location l_i for its parent v_i , or in other words, the argmin of the above expressions. (We will see shortly that there is indeed such a procedure.) The following strategy can then be used to perform the energy minimization:

1. Perform a post-order traversal of the pictorial structure tree. Upon visiting each node v_j , compute B_j and l_j^* (for all possible values of l_i). Once this is complete, we will have obtained the minimum overall value for the energy function.
2. Locate the optimal location of the root by computing the location corresponding to the optimal score for the root.
3. Perform a pre-order traversal of the pictorial structure tree. Upon visiting each node v_j , set the optimal location of the node to be l_j^* for the optimal location of the parent. Note that this is not necessary for the root.

Upon completion, the above procedure gives the optimal locations for each part in the pictorial structure. A simplistic implementation of this algorithm runs in $O(h^2n)$ time, where n is the number of nodes in the pictorial structure and h is the number of locations considered in the location space (in practice, the location space is divided into a grid with h locations, and minimization is performed by iterating over the discrete locations in the grid; considering for each location of the parent, every location of the child). In the next section, we will see how to improve the running time of this algorithm.

2.3.2 The Generalized Distance Transform

The running time of our algorithm can be improved by employing the generalized distance transform. It is defined as follows.

Definition 2 Consider a grid \mathbb{G} . Suppose $\rho : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{R}$ is some measure of distance between points on the grid. Further suppose that $f : \mathbb{G} \rightarrow \mathbb{R}$ is some arbitrary function defined on grid points. Then the function $D_f : \mathbb{G} \rightarrow \mathbb{R}$ given by:

$$D_f(x) = \min_{y \in \mathbb{G}} (\rho(x, y) + f(y)) \quad (2.15)$$

for any point $x \in \mathbb{G}$ is defined as the generalized distance transform given f .

The idea is to rewrite the general form of B_j in the form of a generalized distance transform. It can easily be seen that f corresponds to the appearance (m_j) and recursive (B_c) terms. Moreover, ρ corresponds to d_{ij} if the latter is expressed in the form of a Mahalanobis distance. A Mahalanobis distance is a “scale-invariant” distance measure, defined as follows:

$$\rho(x, y) = (x - y)^T \Sigma^{-1} (x - y) \quad (2.16)$$

where x and y are vectors in some space, and Σ is a general covariance matrix for data points in the space. In our case, d_{ij} can be expressed as a Mahalanobis distance after transforming the locations:

$$d_{ij}(l_i, l_j) = (T_{ij}(l_i) - T_{ji}(l_j))^T D_{ij}^{-1} (T_{ij}(l_i) - T_{ji}(l_j)) \quad (2.17)$$

where T_{ij} and T_{ji} are transformations in the underlying space of which the grid is a discretization; and D_{ij} is a diagonal covariance matrix. Thus, we can write:

$$B_j(l_i) = D_f(T_{ij}(l_i)) \quad (2.18)$$

where

$$f(y) = \begin{cases} m_j(T_{ji}^{-1}(y)) + \sum_{v_c \in C_j} B_c(T_{ji}^{-1}(y)) & \text{if } y \in \text{range}(T_{ji}) \\ \infty & \text{otherwise} \end{cases} \quad (2.19)$$

Efficient algorithms exist which compute the distance transform in $O(k)$ time, where k is the number of grid locations. Therefore, by computing B_j using the distance transform, the running time of the algorithm can be reduced to $O(kn)$, where k is the number of grid locations in the transformed grid. Since in practice k is usually $O(h)$, we get a reduction of running time to $O(hn)$.

2.4 Sampling the Posterior

The algorithm described in the previous section locates the *best* match of the model in a given image. However, real-world images may contain multiple human figures, and we would like to detect all of them. Moreover, the appearance and structure models for human figures that we use are imprecise. We now describe the modifications necessary to address these issues.

Instead of performing MAP estimation (or equivalently, energy minimization), we now consider precomputing the posterior distribution and sampling from it in order to get the best few matches of the model in the image. The idea is that since good matches will have a much higher

posterior than bad matches, this approach will locate multiple human figures in a single image. The basic approach is quite similar to the MAP estimation case, except that the minimizations in the recursive equations are replaced with summations.

We have previously seen that the posterior distribution is given by

$$\Pr(L | I, \theta) \propto \left(\prod_{i=1}^n \Pr(I | l_i, u_i) \prod_{(v_i, v_j) \in E} \Pr(l_i, l_j | c_{ij}) \right) \quad (2.20)$$

Suppose the vertex r is the root of the pictorial structure tree. To begin, the marginal distribution $\Pr(l_r | I, \theta)$ is computed. From this distribution, a location for the root is sampled. Then for each child c of the root, the distribution $\Pr(l_c | l_r, I, \theta)$ is computed and a location for the child relative to the location of its parent is sampled. Recursion proceeds down the tree in this manner, and locations are sampled for each of the parts.

The marginal distribution for the root location is:

$$\Pr(l_r | I, \theta) \propto \sum_{l_1} \cdots \sum_{l_{r-1}} \sum_{l_{r+1}} \cdots \sum_{l_n} \left(\prod_{i=1}^n \Pr(I | l_i, u_i) \prod_{(v_i, v_j) \in E} \Pr(l_i, l_j | c_{ij}) \right) \quad (2.21)$$

Since the pictorial structure graph is a tree, we can write

$$\Pr(l_r | I, \theta) \propto \Pr(I | l_r, u_r) \prod_{v_c \in C_r} S_c(l_r) \quad (2.22)$$

where the S_j functions give a recursive formulation similar to the B_j functions in the MAP estimation case:

$$S_j(l_i) \propto \sum_{l_j} \left(\Pr(I | l_j, u_j) \Pr(l_i, l_j | c_{ij}) \prod_{v_c \in C_j} S_c(l_j) \right) \quad (2.23)$$

For any other node j with parent i :

$$\Pr(l_j | l_i, I, \theta) \propto \Pr(I | l_j, u_j) \Pr(l_i, l_j | c_{ij}) \prod_{v_c \in C_j} S_c(l_j) \quad (2.24)$$

So the algorithm proceeds as follows:

1. Perform a post-order traversal of the pictorial structure tree. Upon visiting each node v_j , compute S_j (for all possible values of l_i).
2. Sample a location for the root from the marginal distribution of root locations, $\Pr(l_r | I, \theta)$.
3. Perform a pre-order traversal of the pictorial structure tree. Upon visiting each node v_j , sample a location from the node from $\Pr(l_j | l_i, I, \theta)$, where l_i is the sampled location for its parent. Note that this is not necessary for the root.

The process is quite similar to the MAP estimation case. The key difference is that a new approach is needed for computing the S_j functions efficiently.

Due to the particular form of d_{ij} that is used (compare with Equation 2.17 and note that $d_{ij}(l_i, l_j) = -\log \Pr(l_i, l_j | c_{ij})$), the prior can be written in the form of a Gaussian distribution:

$$\Pr(l_i, l_j | c_{ij}) \propto \mathcal{N}(T_{ij}(l_i) - T_{ji}(l_j), 0, D_{ij}) \quad (2.25)$$

This allows the S_j function to be rewritten as a Gaussian convolution:

$$S_j(l_i) \propto (F \otimes f)(T_{ij}(l_i)) \quad (2.26)$$

where F is a Gaussian filter (whose covariance is given by D_{ij}) and f is the following function:

$$f(y) = \begin{cases} \Pr(I | T_{ji}^{-1}(y), u_j) \prod_{v_c \in C_j} S_c(T_{ji}^{-1}(y)) & \text{if } y \in \text{range}(T_{ji}) \\ 0 & \text{otherwise} \end{cases} \quad (2.27)$$

Since the covariance matrix is diagonal, the Gaussian filter is separable. This fact is used to approximate the Gaussian convolution using a cascade of uniform filters using methods described in [23]. This gives us an overall running time of $O(hn)$, where n is the number of parts and h is the number of transformed grid locations.

In practice, the posterior values for some incorrect poses tend to be large, comparable to the peaks in the posterior corresponding to the “correct” pose(s). Therefore, the posterior sampling algorithm is typically used to generate a large set of candidate poses (around 100-200 poses), from which the correct pose is located by computing the Chamfer distance between the object shape under the candidate pose and the binary image obtained through background subtraction.

We defer discussion of our implementation of the pictorial structures algorithm until a later chapter (Chapter 6), where we shall compare it with our skeleton-based pose estimation algorithm.

Chapter 3

A Bottom-Up Approach

The pictorial structures approach is a top-down pose estimation algorithm. In this chapter we shall describe a contrasting bottom-up pose estimation algorithm [16]. To locate potential body parts, we use a part detector which characterizes parts by a pair of parallel edges in the image. This requires a boundary detection algorithm that does not return too many spurious boundaries; we shall describe such a system, based on the *Pb* operator [12]. The actual pose estimation is expressed as an integer quadratic program (IQP), and we shall describe how to express the pose estimation problem as an IQP, and how to solve it.

3.1 Contour Detection

Parts are assumed to be characterized by a pair of parallel line segments (which form the boundary of the part). The motivation for this assumption comes from the Gestalt school of psychology [16]. This characterization means that the parts classifier need concern itself only with edges in the image. Therefore, the first step of preprocessing an image would be to perform edge detection.

3.1.1 Probability of Boundary

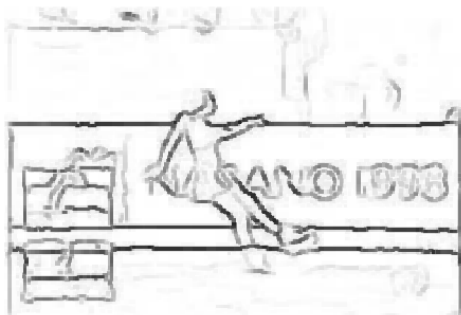
Instead of using a standard edge detector, we employ a method for computing the *probability of boundary* at each pixel in the image. Termed the *Pb* operator, this method [12] analyzes the brightness, colour and texture around each image pixel and computes the posterior probability that the pixel is part of a boundary. It is important to note the distinction between “edge” and “boundary” here. An edge (the output of, say, the Canny detector) is simply a contour across which local image features such as intensity change abruptly. However, a boundary is a contour separating distinct objects in the image. This is much more useful for vision applications, since it takes into account issues such as texture. An edge detector would detect a large number of edgels in a highly textured region, whereas the *Pb* map returns low probabilities in such regions. Hence closed contours extracted from a *Pb* map are very likely to contain distinct objects.

The basic idea is as follows. For each pixel, consider an image patch centered around the pixel. Based on this image patch, a feature vector is computed (as explained shortly). This feature vector is then given as input to a classifier, whose output is used to estimate the probability of a boundary passing through the pixel. (In practice, the classifier used is a logistic classifier, or one of its variants.) The output of such a classifier directly gives the probability we seek.

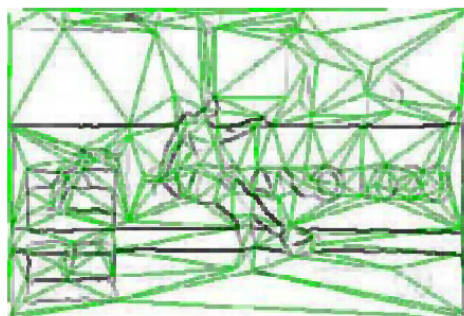
In order to compute the feature vector components for a pixel, a circular patch is constructed, centered at the pixel. It is divided into 2 halves along a diameter at an angle θ to the horizontal.



(a) Sample input image.



(b) Edges detected in the image using the Pb operator.



(c) Constrained Delaunay triangulation obtained from the edge map. (Constrained edges are shown in black.)

Figure 3.1: Stages of preprocessing employed in [16].

This process is repeated over eight orientations and three scales σ to compute the entire feature vector. The feature vector used by the classifier contains the following components:

Oriented Energy

For any given orientation θ and scale σ , the *orientation energy* $OE_{\theta,\sigma}$ is computed as follows:

$$OE_{\theta,\sigma} = (I * f_{\theta,\sigma}^e)^2 + (I * f_{\theta,\sigma}^o)^2 \quad (3.1)$$

where I is the input image and $f_{\theta,\sigma}^e$ and $f_{\theta,\sigma}^o$ are even- and odd-symmetric filters, respectively. (In particular, the even-symmetric filter is a Gaussian second derivative, and the odd-symmetric filter is its Hilbert transform [24].) For any pixel (x, y) , the oriented energy is simply $OE_{\theta,\sigma}(x, y)$.

Brightness Gradient

For each half of the patch, the luminance values of each of its pixels (the L component of the CIELAB colour space) are binned into a histogram. The difference between the two histograms corresponding to the two halves of the patch is then computed using the χ^2 difference operator:

$$\chi^2(h_1, h_2) = \frac{1}{2} \sum_i \frac{(h_1 - h_2)^2}{h_1 + h_2} \quad (3.2)$$

The resulting difference, the *brightness gradient*, is denoted by BG .

Colour Gradient

In a similar manner as the brightness gradient, a *colour gradient* in CIELAB space can be computed by taking the difference of two two-dimensional histograms (one for each half of the patch) generated by binning the (a, b) values of the corresponding pixels. However, this measure (CG^{ab}) is approximated by first computing two separate gradients, one for the a term and one for the b term, and then adding them:

$$CG^{a+b} = CG^a + CG^b \quad (3.3)$$

Since the a and b terms correspond to the perceptually orthogonal red-green and yellow-blue colour opponents, this is considered to be a reasonable approximation.

Texture Gradient

To describe the texture around a pixel, its responses to a bank of 13 filters is computed. This is done for each pixel in the two halves of the circular patch under consideration. Since computing a difference between two 13-dimensional histograms would not be feasible, the following approach is used.

If we consider the 13 responses of a pixel as a 13-dimensional vector, it turns out that the response vectors are not uniformly distributed in 13-dimensional space. Instead, they exist in clusters. Therefore, given a set of response vectors, they are first clustered (using the k -means algorithm). Then for each cluster, its center is referred to as a *texton*, and is given a unique identifier. With any pixel is associated the texton closest to its corresponding response vector. The resulting mapping between pixels and textons is referred to as a *texton map*. The texton map is to some extent a description of texture variation in the image.

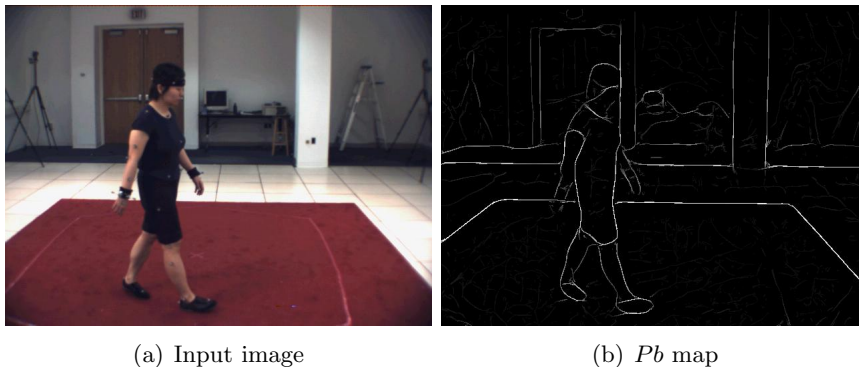


Figure 3.2: Example of a Pb map. Brighter pixels indicate higher probability.

Once the texton map is computed, a histogram of texton values for each pixel in the two halves of the circular patch is computed. Then the difference between the two 1-dimensional histograms is computed, and denoted by TG .

Figure 3.2 shows an example of a Pb map computed from a colour image.

3.1.2 Hysteresis Thresholding

Having obtained the Pb map for the image, we still need to derive a binary edge map from it. One possible way of doing so would be to apply a threshold to the probability values, setting only those pixels in the edge map whose Pb value is greater than some threshold T . However, due to imperfections in the computed probability values, this process may lead to a large number of gaps in the recovered edge map. One standard method of overcoming this problem (to a degree) is *hysteresis thresholding*.

Hysteresis thresholding works as follows. We are given two thresholds, T_1 and T_2 , such that $T_1 \geq T_2$. To construct the edge map, we first choose all pixels whose probability is at least T_1 . Next, we locate all pixels whose probability is at least T_2 and which are connected via a sequence of such pixels to a pixel whose probability is at least T_1 . Here, two pixels are considered to be connected if they are neighbours of each other.

In practice, this process tends to reduce the number of gaps in the edge map as compared to the usual single threshold method. However, it does not entirely eliminate the problem. In the next section, we shall describe a method of recovering closed contours given the edge map recovered via hysteresis thresholding.

3.1.3 Boundary Completion

The above process returns piecewise linear contours which may contain gaps due to issues with the Pb computation. Therefore, a method to fill in the gaps in a piecewise linear contour is needed. The *constrained Delaunay triangulation* is employed for this purpose, due its beneficial scale-invariance properties [17].

Constrained Delaunay Triangulation

The constrained Delaunay triangulation is defined as follows:

Definition 3 *The constrained Delaunay triangulation (CDT) of a set of line segments E is a triangulation T of the endpoints of each edge in E such that:*

1. every edge in E is present in T ; and
2. T is as close as possible to the Delaunay triangulation of the edges' endpoints, subject to the above requirement.

A CDT is constructed with the contour edges as constraint edges. This process fills gaps in the contours and leaves us with a triangulation over the image domain. However, it also leads to a large number of non-contour edges, which would unnecessarily increase the search space of the part detector. In the next section, we describe a method to discard such non-contour edges by learning a contour model.

Contour Completion using the CDT

The basic idea is as follows: we want to compute the probability that any given edge is part of the contour (to be more precise, is part of a piecewise linear approximation to the “correct” contour). We use the following local continuity model [17]. With each edge e_i , associate a random variable X_i , which is 1 if e_i is a contour edge, and 0 otherwise. Consider a pair of edges e_1 and e_2 , sharing a common endpoint. Assume that either both X_1 and X_2 are 1, or both are 0. (This based on the observation that in practice, most contours are closed.)

This lets us associate a random variable Y_{12} with this edge pair. With each such edge pair (sharing a common endpoint), associate a feature vector \mathbf{x}_{ij} , having the following components:

- \overline{Pb}_{ij} , the average Pb value along the edge pair;
- θ_{ij} , the angle between the edges;
- $G_{ij} = G_i G_j$, where $G_i = 1$ if edge e_i is a constraint edge in the CDT, 0 otherwise.

A logistic classifier is trained to classify edge pairs with a common endpoint. A logistic classifier is a simple modification of logistic regression, which works as follows. Suppose we have a feature vector \mathbf{x} of independent variables, and we wish to predict the value of y , the dependent variable. The relation between \mathbf{x} and y is assumed to have the form of the sigmoid function:

$$y = \frac{e^{\beta^T \mathbf{x} + \beta_0}}{1 + e^{\beta^T \mathbf{x} + \beta_0}}$$

where β and β_0 give the coefficients of a hyperplane, which have to be learned from training data. (β_0 is often absorbed into β by suitably modifying the feature vector.) Now, if y is constrained to take on exactly two values (say $+1$ and -1), then we have a logistic classifier.

The probability that edge e_0 (between endpoints u and v) is a contour edge is computed as follows:

$$\Pr(e_0) = \max_{e_1 \in E_u} \Pr(Y_{01} = 1) \max_{e_2 \in E_v} \Pr(Y_{02} = 1) \quad (3.4)$$

where E_u is the set of edges sharing endpoint u with e_0 (and similarly for E_v). Only those edges are then chosen whose probabilities are above some threshold. Once this process is complete, we have a set of edges which lie along object boundaries. The next step is to choose pairs of approximately parallel edges in order to detect potential parts.

3.2 Detecting Parts

A logistic classifier is used for bottom-up part detection. Given a pair of edges in the CDT graph, a feature vector is derived with the following elements: (Here, L is the length of an edge, α its orientation, \vec{C} its center and Pb the average contrast on it; \vec{T} and \vec{N} are the average tangent and normal directions, respectively.)

- **orientation consistency**, $|\alpha_1 - \alpha_2|$.
- **length consistency**, $|L_1 - L_2|/|L_1 + L_2|$.
- **low-level contrast**, $|Pb_1 - Pb_2|$ and $|Pb_1 + Pb_2|$.
- **distance between centers**, in the tangent direction ($|(\vec{C}_1 - \vec{C}_2) \cdot \vec{T}|/|L_1 + L_2|$) and in the normal direction ($|(\vec{C}_1 - \vec{C}_2) \cdot \vec{N}|/|L_1 + L_2|$).
- **intervening contour**, which is the maximum Pb value on all edges intersecting the straight line connecting \vec{C}_1 to \vec{C}_2 .

Before the classifier can be used to detect parts, it must be trained using a training set of (\mathbf{x}, y) pairs where the feature vectors have been classified by hand (or, to be more precise, whose class labels are known beforehand). Another important part of the training phase is learning the parameters to the various cost functions, which we shall describe next.

3.3 Solving the IQP

We shall now describe the cost functions used in expressing the pose estimation problem in terms of the IQP problem, and the method used to learn the parameters of these cost functions.

3.3.1 Learning the Constraints

Since this approach deals with pose estimation from single images, we use only appearance and structure cost functions. In this section, we detail the various cost functions that we use to evaluate candidate matches.

The appearance cost functions we use are:

- **aspect ratio**, the ratio of part length to part width.
- **low-level classifier score**, the score returned by the logistic classifier for a given part.
- **scale consistency** between a pair of candidate parts. If the candidate part widths are w_1 and w_2 , and the ideal widths of the corresponding parts are W_1 and W_2 , then we compare $r = (w_1 - w_2)/(w_1 + w_2)$ to $R = (W_1 - W_2)/(W_1 + W_2)$. The goal of this cost function is to capture the relative sizes of body parts in a scale-invariant manner.
- **orientation consistency** between a pair of candidate parts. If the orientations of the candidate parts are α_1 and α_2 , then we compute $|\alpha_1 - \alpha_2|$.
- **appearance consistency** between a pair of candidate parts is computed by comparing the average colour of each candidate part in *Lab* colour space.

Before we describe the structure cost functions we use, we need the definitions of a couple of functions computed over pairs of edges in the CDT graph:

- $dist(e_1, e_2)$ Given two edges e_1 and e_2 in the CDT graph, this function returns the total weight of all edges along the shortest path from e_1 to e_2 . Note that non-constraint edges are assigned thrice the weight of constraint edges.
- $gap(e_1, e_2)$ Given two edges e_1 and e_2 , this function returns the number of non-constraint edges along the shortest path from e_1 to e_2 .

A pair of candidates c_1 and c_2 are described by edge pairs (a_1, b_1) and (a_2, b_2) . A correspondence between edges in each candidate is defined by choosing a_1 and a_2 to be the pair of edges with the smallest value of $dist(a_1, a_2)$. The other two edges are then b_1 and b_2 .

The structure cost functions can now be described. They include:

- **limb-limb separation**, which is computed in terms of the $dist$ measure for pairs of connected limbs (such as upper right arm and lower right arm).
- **limb-torso separation**, which is computed in terms of the $dist$ measure between the torso and a limb connected to the torso.
- **V-shape between upper legs**, which is also computed in terms of the $dist$ measure. This cost function tries to capture the fact that in most poses, the upper legs and torso are arranged in a characteristic V-shaped configuration.
- **upper-arm/upper-leg connectivity**, which is computed in terms of the gap measure between corresponding upper arms and upper legs. This cost function tries to capture the fact that (for example) the right upper arm and right upper leg form a smooth contour with the right edge of the torso.

Each of the cost functions is assumed to take values from independent Gaussian distributions. The mean and variance of these distributions is computed based on a set of training examples. We shall now see how these cost functions are put to use in the IQP framework.

3.3.2 Constructing the IQP

Combining candidate parts into a match for the model as a whole can be formulated in terms of finding a labelling of candidate parts: $\pi : \{l_i\} \rightarrow \{c_i\}$ (where l_i are labels and c_i are candidate parts). The cost functions then take the form $f(l, c)$ (in the unary case) and $f(l_1, c_1, l_2, c_2)$ (in the binary case).

Suppose a constraint f_k is defined by Gaussian parameters (μ_k, σ_k) . Define $\bar{f}_k = (f_k - \mu_k)^2 / \sigma_k^2$. Now it can be seen that finding the maximum likelihood assignment π^* is equivalent to minimizing the following expression:

$$\sum_{l_1, l_2} \sum_k \bar{f}_k(l_1, \pi(l_1), l_2, \pi(l_2)) + \sum_l \sum_k \bar{f}_k(l, \pi(l))$$

An assignment is represented by a vector \mathbf{x} , which is defined as follows. Consider first a vector \mathbf{v} where the element v_i is the label assigned to the i^{th} candidate. Here, labels are integers in the range $[1, N]$ (where N is the number of labels) or 0 to indicate no label. The number of elements in such a vector is equal to the number of candidate parts. Now replace each v_i with a sequence of N binary values, such that if v_i is some non-zero label k , then the k^{th} bit in the sequence is set to 1 while all the others are 0; and if v_i is zero, then all bits are zero. The number of elements in the resulting vector is then N times the number of candidate parts. This vector is the vector \mathbf{x} that we shall be using in the rest of this discussion.

Also, $c(i)$ is defined to be the candidate part corresponding to index i , and $l(i)$ to be the label corresponding to the index i . It should be clear from the above discussion that

$$\begin{aligned} c(i) &= \left\lfloor \frac{i}{N} \right\rfloor \\ l(i) &= i \bmod N \end{aligned}$$

Note that such a vector must satisfy the constraint that for each part label L :

$$\sum_{i:l(i)=L} x_i = 1$$

i.e., each part label is assigned to exactly one candidate and each candidate is assigned to exactly one part. The authors do not mention it, but the following constraint must also hold for correctness. For each candidate C :

$$0 \leq \sum_{i:c(i)=C} x_i \leq 1$$

i.e., each candidate part is assigned at most one label.

The above two equations are essentially an instance of the classic Integer Quadratic Programming problem, and are rewritten as follows:

$$\begin{aligned} \mathbf{x}^* &= \arg \min(\mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{c}^T \mathbf{x}) \\ \text{subject to } \mathbf{A} \mathbf{x} &\leq \mathbf{b} \end{aligned}$$

where

$$\begin{aligned} c_i &= \sum_k \bar{f}_k(l(i), \pi(l(i))) \\ H_{ij} &= \sum_k \bar{f}_k(l(i), \pi(l(i)), l(j), \pi(l(j))) \end{aligned}$$

where $\mathbf{A} \mathbf{x} \leq \mathbf{b}$ captures the constraints mentioned above. A lower bound to the solution to the IQP is given by the solution to the following linear programming problem:

$$\begin{aligned} q_i &= \min \sum_j H(i, j) x_j \\ \mathbf{x}^* &= \arg \min \sum_i (q_i + c_i) x_i \\ \text{subject to } \mathbf{A} \mathbf{x} &\leq \mathbf{b} \end{aligned}$$

as explained in [16]. Since the above method gives only an approximate solution to the IQP, further processing is needed. Starting from the assignment that minimizes the above LP, a greedy local search is used to improve the solution. Note however, that this local search may still fail to the global optimum.

We defer discussion of our implementation of the above algorithm until a later chapter (Chapter 6), where we shall compare it with our skeleton-based pose estimation algorithm.

Chapter 4

The Skeleton

One of the most well-known descriptors of the shape of an object is its *skeleton*. Roughly speaking, the skeleton transform of an object is a set of curves which lie in the “middle” of the object. As can be seen from the examples in Figure 4.1, the skeleton is a very intuitive shape descriptor. Moreover, the *medial axis transform* (MAT) (a procedure used to compute the skeleton) can be used to reconstruct the original shape in its entirety. Thus the skeleton has remained a popular, powerful method for shape analysis and object detection problems.

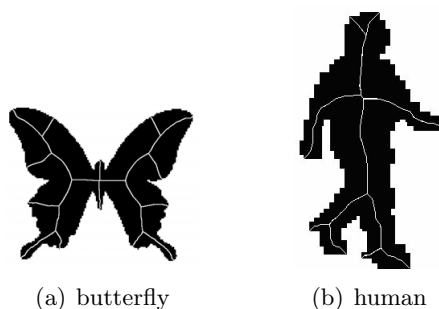


Figure 4.1: Utility of the skeleton as a shape descriptor. The shapes are the black silhouettes, and the skeletons are shown in white. [3] These are pruned skeletons, unaffected by discretization issues in, say, the human example.

In the rest of this report, we propose a method to detect humans and estimate their pose from a single image using the skeleton transform. While the details of this process will be described in Chapter 5, in this chapter we shall describe the method we use to compute the skeletons of the objects in a given image. To begin with, we need an estimate of the boundaries of the objects whose skeletons we wish to compute. We also need to ensure that these boundaries are closed. Once this is done, we may compute the skeleton transforms. A block diagram of this process is shown in Figure 4.2. In the subsequent sections, we shall describe each step of this process in detail.

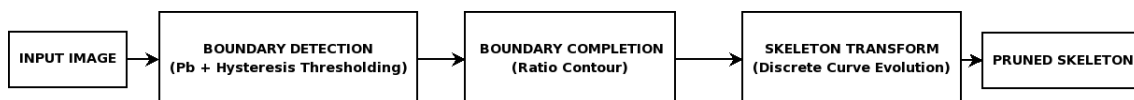


Figure 4.2: Block diagram of the skeleton extraction process.

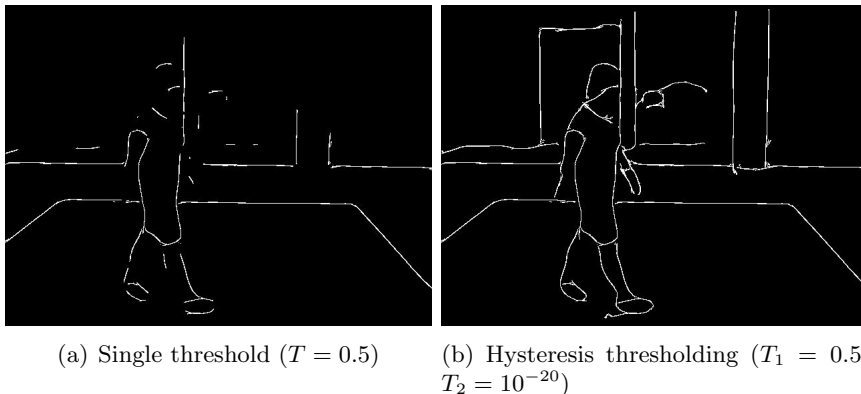


Figure 4.3: Benefit of using hysteresis thresholding. Notice the gaps, such as those in the legs, which are filled by hysteresis thresholding.

4.1 Closed Contours: Ratio Contour

The first step in our process is to detect the boundaries of the objects in the image. We employ the Pb operator followed by hysteresis thresholding for this purpose (as explained in Chapter 3). Although hysteresis thresholding attempts to reduce the number of gaps in the detected boundaries (see Figure 4.3), it fails to completely eliminate them. We shall now examine a graph-based algorithm, Ratio Contour [22], for filling in these gaps, and returning closed boundaries of objects in the image. We have chosen this algorithm as opposed to the CDT-based algorithm described in Chapter 3 primarily because of its lack of dependence on a training set of any sort.

To begin, the binary edge map is converted into a set of edge fragments, which may or may not be connected to each other. Edge fragments thus recovered are referred to as *real fragments*, to distinguish them from *virtual fragments*, which we shall describe shortly. Since the edge map may contain issues such as intersections between edge fragments, we first perform a preprocessing step. In this step, edge fragments are split at intersection points and points of high curvature. This technique is used to handle intersections, false attachments between ideally separate edge fragments, and closed edge fragments in the edge map. Note that at each point where a fragment is split, two overlapping “endpoints” are created, so that no two real fragments share a common endpoint.

The next step is to construct potential gap completions, which are referred to as virtual fragments. To construct a virtual fragment between any two real fragment endpoints, they are first connected by a straight line segment. Then consider the *combined fragment* which is made up of the line segment and one half of each of the real fragments (the half containing the endpoints under consideration). A spline-based algorithm is then used to smooth the resulting curve. Removing the halves of the real fragments from the smoothed combined fragments leaves us with the final virtual fragment.

Given the combined set of real and virtual fragments, the next step is to extract salient closed boundaries. Note that we are concerned with *nondegenerate boundaries*, i.e. boundaries which do not self-intersect. For any closed boundary given as a sequence of fragments (real and/or virtual), the following cost is defined:

$$\Gamma(B) = \frac{W(B)}{L(B)} = \frac{\int_B [v(t) + \lambda\kappa^2(t)]dt}{\int_B dt} \quad (4.1)$$

where t is an arc length parameter, B is a closed boundary, $\kappa(t)$ is the curvature of the

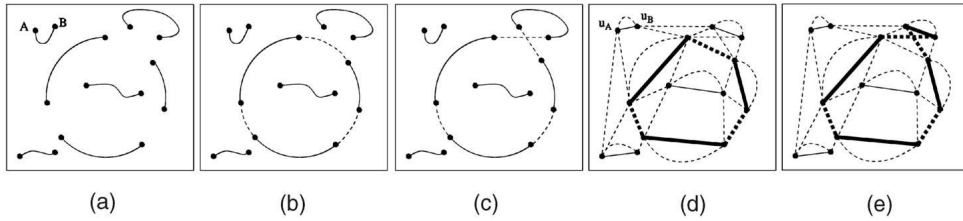


Figure 4.4: Salient boundaries and simple alternate cycles. (a) A set of real fragments. (b) A subset of real and virtual fragments giving a nondegenerate, non-self-crossing boundary. (c) A nondegenerate but self-crossing boundary. (d) The solid-dashed graph corresponding to (b), with the boundary shown in bold. (e) The solid-dashed graph corresponding to (c), with the boundary shown in bold.

boundary at t , $v(t) = 1$ if the point at t lies in a virtual fragment (and is 0 otherwise), and λ is a pre-determined weight parameter. Our goal therefore is to locate the closed boundary which minimizes the above cost. We shall now describe a graph-based algorithm to perform this optimization efficiently.

Consider a graph $G = (V, E)$ whose vertices correspond to fragment endpoints, and whose edges correspond to fragments. There are two kinds of edges: solid and dashed, corresponding to real and virtual fragments, respectively. Such a graph is referred to as a *solid-dashed graph*. To describe closed boundaries with respect to such a graph, we need the following definitions:

Definition 4 A *simple cycle* in a graph $G = (V, E)$ is a cycle that does not traverse any vertex more than once.

Definition 5 An *alternate cycle* in a graph $G = (V, E)$ is a cycle that alternately traverses solid and dashed edges.

Based on the above definitions, it can be seen that a nondegenerate closed boundary in an edge map is equivalent to a simple alternate cycle in its corresponding graph. With each edge e_i in the graph, associate a weight $w(e_i) = W(B(e_i))$ and a length $len(e_i) = L(B(e_i))$, and on any boundary, define the following cost function:

$$\Gamma(B) = \frac{\sum_{e_i \in B} w(e_i)}{\sum_{e_i \in B} len(e_i)} \quad (4.2)$$

where $B(e_i)$ denotes the fragment corresponding to the graph edge e_i . Thus we see that the problem of computing the most salient closed boundary reduces to computing the *minimum ratio alternate cycle* in the graph G . (See Figure 4.4 for an example.)

4.1.1 A Polynomial Time Solution

We shall now see a series of reductions that will let us reformulate the problem in a manner which can be solved in polynomial time.

Reduction 1: Modifying Edge Weights of Solid Edges

By construction, two solid edges can never share a common vertex. So consider a solid edge e . To every dashed edge adjacent to e , add the weight $\frac{1}{2}w(e)$ and the length $\frac{1}{2}l(e)$, and set the weight and length of e to 0. This reassignment is performed for every solid edge in the graph.

Since any alternate cycle which passes through e must necessarily pass through two dashed edges adjacent to e , the total weight and length of any alternate cycle in the graph remains unchanged. In other words, this reduction does not alter the minimum ratio alternate cycle (or its cost).

Reduction 2: Detecting Negative Weight Alternate Cycles

Consider the following transformation of edge weights:

$$w'(e) = w(e) - b \cdot l(e) \quad (4.3)$$

Since the above transformation offsets the ratio costs of any alternate cycle by b , the transformation does not alter the MRA cycle. This fact allows us to use the above transformation to reframe the MRA problem as follows. Suppose we have a subroutine to locate a *negative weight alternate cycle* if one exists. Let the cost of the NWA cycle thus obtained be b . The above transformation is then performed, offsetting the cycle ratio by b . This process is repeated until no NWA exists. It can easily be seen that in this case, the cost of the MRA cycle is 0.

The last NWA cycle located must necessarily be the MRA cycle. Hence, in order to locate the MRA cycle, all we need is a method to locate NWA cycles.

Reduction 3: Finding Minimum Weight Perfect Matchings

We will now describe how an NWA cycle detector can be constructed based on a *minimum weight perfect matching* (MWPM) detector. We first need the following definition:

Definition 6 A *perfect matching* in G denotes a subgraph of G which includes each vertex of G but in which each vertex has exactly one incident edge.

The MWPM is the perfect matching in G which has the minimum total edge weight. Since each solid edge has weight 0, the set of solid edges forms a trivial perfect matching, with total weight 0. It follows that the MWPM has nonpositive total weight. Now, given any perfect matching P , a subgraph of G which contains only a set of disjoint alternate cycles can be constructed as follows:

1. From P , remove all solid edges and their incident vertices. Denote the resulting subgraph by P' .
2. To P' , add all solid edges in G which were not in P . Denote the resulting subgraph by P'' . This is the required subgraph.

The proof that the above process works is as follows. Each vertex in G is incident on exactly one solid edge. Since no two solid edges can share a vertex, step 2 implies that each vertex in P'' has degree 2. Also, by construction, each vertex in P'' is incident on exactly one solid edge and exactly one dashed edge. Therefore, it can be seen that each vertex in P'' is part of a simple alternate cycle, implying that P'' is a set of disjoint alternate cycles.

The authors show that each cycle in P'' has nonpositive total weight. Therefore, detecting an NWA cycle is equivalent to computing an MWPM, carrying out the above two-step process, and locating the cycle in P'' which has minimum cost. Figure 4.5 illustrates an example of this reduction process.

Based on the above three reductions, and the fact that there are known polynomial time algorithms for computing the MWPM, we can obtain a polynomial time algorithm for computing the MRA cycle. Figure 4.6 shows an example closed contour recovered by the Ratio Contour algorithm.

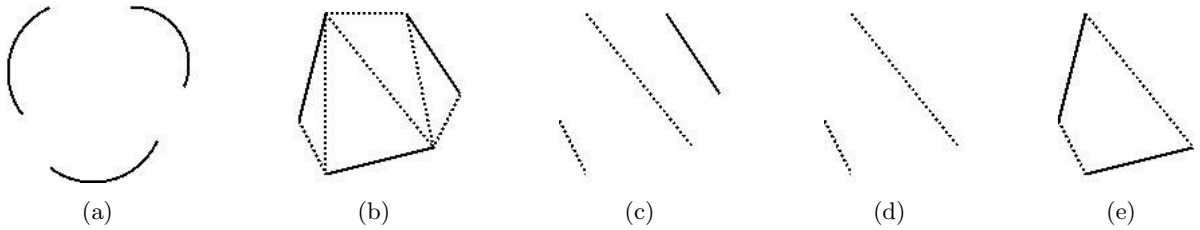


Figure 4.5: Reduction 3 in Ratio Contour. (a) A set of real fragments. (b) The corresponding solid-dashed graph G , with some of the dashed edges shown. (c) A perfect matching P in G . (d) Removing solid edges from P , giving P' . (e) Adding solid edges in G which are absent in P , giving P'' , a simple alternate cycle.

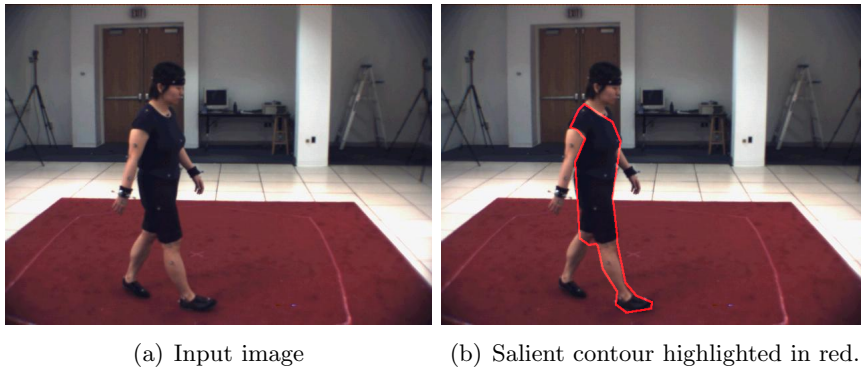


Figure 4.6: Example of a salient contour detected by Ratio Contour.

4.1.2 Possible Improvements

Ratio Contour is not necessarily the optimal boundary completion algorithm for our purposes. Due to the nature of its cost function, it tends to favour more circular contours. This leads to trouble in distinguishing between, for example, the two legs of a human figure. Many other approaches to this problem exist in the literature [6, 7], and their utility for our purposes needs to be assessed.

One of the promising ideas involves incorporating appearance information in the contour completion process [7]. Basically, the algorithm tries to favour boundaries which best separate visually distinct parts of the image. In the common case where (say) the appearance of the right and left legs is symmetric, this can lead to a boundary which properly distinguishes between the two legs. We leave such analysis to future work.

4.2 Medial Axis Transform

Before we describe the skeleton, we need the following definition:

Definition 7 For any 2-dimensional shape S and its corresponding boundary ∂S , a **maximal circle** is defined as a circle which lies inside S , and which is tangential to ∂S in at least 2 distinct points.

The skeleton and the medial axis transform are defined as follows:

Definition 8 The **skeleton** $Sk(S)$ of a shape S is defined as the locus of the centres of all maximal circles contained in S .

Definition 9 The *medial axis transform* of a shape S is the skeleton of S combined with a function $f : Sk(S) \rightarrow \mathbb{R}$, which gives the radius of the maximal circle associated with each skeleton point.

Some of the standard methods of computing the skeleton (or MAT) are:

- **Morphological thinning:** Suppose we fill the interior of the detected closed contour. We employ a form of successive morphological thinning which preserves line segment endpoints. If we keep performing the thinning operation until no further thinning is possible, the remaining pixels approximate the skeleton.
- **Distance transform:** If we compute the distance transform of the contour image, then the skeleton lies along ridges in the distance transform.
- **Voronoi poles:** If the boundary points are sampled densely enough, then it can be shown [1] that the medial axis is approximated by the *poles* of the Voronoi diagram of the boundary points (the furthest vertices of the Voronoi cells of each boundary point which lie inside the boundary).

Since we are dealing with image data, we choose to employ the first method. However, due to noise and other errors in the detected boundaries, the skeleton may contain a large number of spurious branches (see Figure 4.7). To address this issue, we describe a method for pruning a skeleton in the next section.

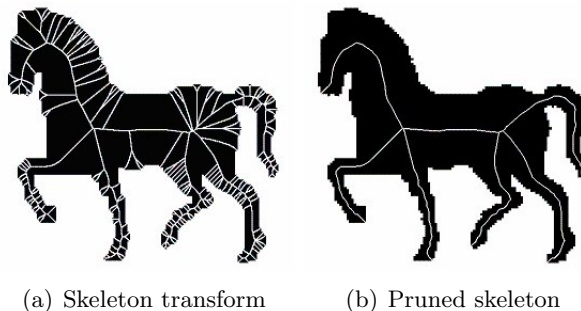


Figure 4.7: Pruning an example skeleton. In both cases, the horse shape is the input. Due to noise, the skeleton has many spurious branches, which are removed by the pruning process as shown in (b) [3].

4.3 Skeleton Pruning

At this stage, we are given a closed contour curve and its corresponding skeleton. The objective is to remove those points in the skeleton which are “noise,” or in other words, which do not contribute significantly to the shape of the object. (See Figure 4.7(b) for an example.) We shall now describe a skeleton pruning process, beginning with the following definitions:

Definition 10 For any skeleton point, the points on the boundary which are tangential to its maximal circle are called its *generating points*.

The intuitive notion that noise in the skeleton corresponds to short, “localized” features on the boundary is formalized as follows. Define a sequence of control points p_1, \dots, p_n on the

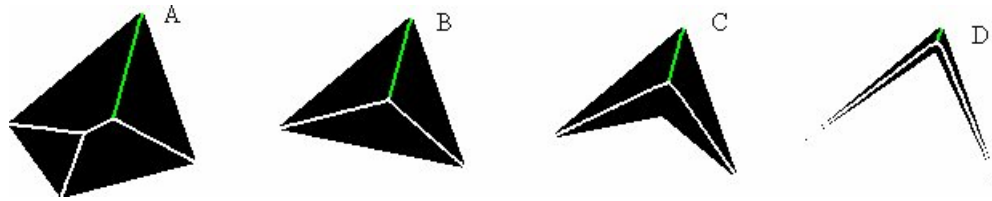


Figure 4.8: Complications caused by concave vertices in the skeleton pruning process. The same convex vertex may generate skeleton branches with different importances. [3]

boundary. Boundary arcs between two consecutive control points are called *boundary segments*. Then use the following pruning strategy: remove all skeleton points whose generating points all lie within the same boundary segment. This requires that we compute a segmentation that is able to distinguish between boundary “features” and “noise.”

We use the *discrete curve evolution* [3] technique to compute a boundary segmentation that is optimal for our purposes. It is assumed that the boundary is a closed, piecewise linear curve. (This is true due to the discrete nature of our input.) The initial set of control points is defined as $P_0 = \{v_1, \dots, v_n\}$, where v_i are the vertices of the boundary curve. From this set, the vertex v which has the minimum value of the following cost function is removed:

$$K(v) = \frac{\beta(s_1, s_2)l(s_1)l(s_2)}{l(s_1) + l(s_2)} \quad (4.4)$$

where s_1 and s_2 are the two polygon sides incident at vertex v , $\beta(s_1, s_2)$ is the turn angle at vertex v , and $l(s)$ denotes the length of the segment s normalized by the total length of the boundary. The resulting set is denoted by P_1 . This process is repeated until some stopping condition is reached. (In our case, we simply stop iterating when the number of control points reaches a target k_0 .) If this process takes k iterations, then P_k is the final contour segmentation. Note that in order to speed up the process of contour simplification, only convex vertices are considered in the partitioning and simplification process.

However, additional complications can be introduced by concave vertices, as shown in 4.8. To make sure that skeleton branches such as those in the figure are assigned a low cost, vertices with a low value of $D_l(v)$ are also removed, where $D_l(v)$ is the distance between v and the nearest concave vertex u such that the line segment vu is inside the shape.

Instead of a two-step process where we first compute the skeleton and then prune it, the authors describe a combined algorithm, where the skeleton is grown recursively [3]. Points are added along the ridges of the distance transform of the boundary, subject to the constraint that any added point must have generating points in at least two distinct segments of the simplified boundary.

In the preceding sections, we have seen how to compute pruned skeletons of salient objects in any image. The next step is to use this information in order to locate human figures and infer their pose. In the next chapter, we shall describe our approach for performing pose estimation using the skeleton.

Chapter 5

Skeleton-Based Pose Estimation

Having obtained a (reasonably) good set of skeletons from the input image, our next step is to perform pose estimation given the skeleton information. One approach to doing this would be to treat the skeleton as a parts detector and plug its output into a bottom-up or top-down pose estimation algorithm. Doing so would require a method to recover the image region corresponding to any skeleton fragment, which we shall see below.

An alternative approach would be to employ a classifier and throw away skeletons which are not “human-like,” based on some appropriate definition of human-like. (Such a definition would typically involve topological constraints on the number of fragments and simple geometric relationships between them.) We could use (among others) either a logistic classifier for this purpose, or a forest of randomized decision trees (RDT). The RDT classifier has been used to recognize typewritten symbols [2], and has the unique advantage that it can deal with a virtually infinite feature space.

However, the part detector approach is simpler, and unlike the skeleton classifier approach, does not require us to stitch together broken skeleton fragments as a preprocessing step. Hence, we choose to not use a classifier-based approach, and instead use the skeleton information as a set of candidate parts. This set of candidate parts is then plugged into a top-down pictorial structures algorithm, which gives us the pose estimate. In the next few sections, we shall explore various issues involved in each step of such a process.

5.1 Extracting Parts

The first step in extracting a set of candidate parts from a skeleton is obviously to break up the skeleton into a set of *skeleton fragments*, each of which represents a single candidate part (see Figure 5.1). Note that since we may have multiple skeletons in a single image (due to there being multiple salient closed boundaries detected in the image), we need to extract parts from each skeleton. The next step is to reconstruct the image region corresponding to each skeleton fragment, using an inverse medial axis transform. We shall describe these processes in more detail in the next subsections.

5.1.1 Skeleton Fragments

To describe the method we use to recover skeleton fragments, we will need the following definition:

Definition 11 *A pixel in a binary image I is a **junction** if, when visiting each of its 8 neighbours in clockwise (or counter-clockwise) order, the number of transitions between 0 and 1 is 3*

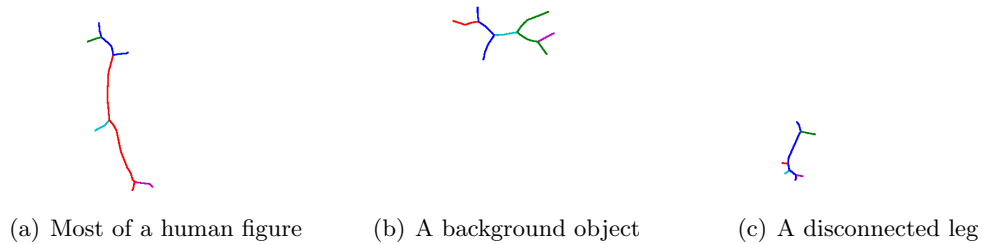


Figure 5.1: Result of splitting skeletons at junction pixels, for three skeletons in the same image. Each resulting fragment is shown in a different color.

or more.

We now define a skeleton fragment as follows:

Definition 12 A *skeleton fragment* is a set of connected pixels belonging to a skeleton in which no pixel other than possibly the first and/or the last pixel are junctions.

(Here, two pixels are connected if they are neighbours of each other.) With this definition, the process of recovering skeleton fragments is simple to describe: we simply trace the skeleton pixels, starting and stopping only at junctions. The result of this process is shown in Figure 5.1.

5.1.2 Inverse Medial Axis Transform

Once we have obtained a set of skeleton fragments, the next step is to recover their corresponding image regions. This is accomplished using the *inverse medial axis transform*. This process requires that for each skeleton point, we store the radius of the maximal circle centered at that point. We accomplish this by computing the distance transform of the closed contour corresponding to the skeleton, and then masking it with the pruned skeleton.

After this information is available, the reconstruction process is simple. For each skeleton pixel, we draw a filled circle centered at the pixel and whose radius is the corresponding maximal circle's radius. We combine the pixels from all circles drawn for a given skeleton fragment (by ORing them together), and the resulting set of pixels serves as a candidate part.

5.2 Structuring the Part Set

The major problem with the above approach is that the process of extracting skeleton fragments can lead to a large number of small, insignificant fragments in place of a large, perceptually significant fragment. This is because we only look at whether or not a pixel is a junction. What is required is an inspection of the skeleton branches incident at junction pixels. In the next sections, we will describe a method of extracting skeleton fragments that takes properties of incident skeleton branches into account, and computes a *hierarchical relationship* between skeleton fragments (a fragment lower in the hierarchy is considered to “branch off” from its parent).

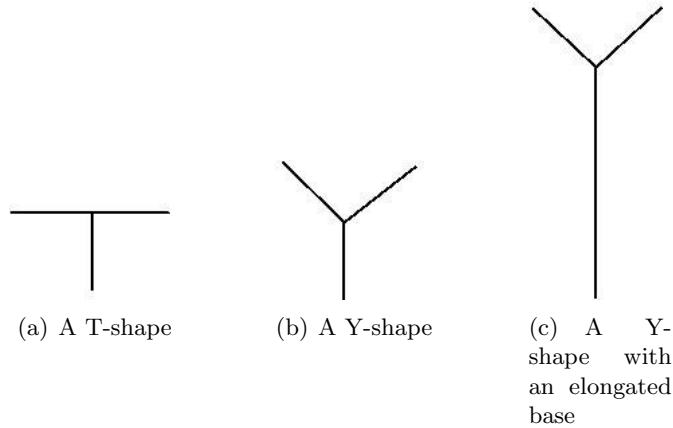


Figure 5.2: Angle and length as cues for skeleton hierarchy. In (a), the vertical segment can be considered to branch off from the longer horizontal segment. In (b), no one segment branches off from any other, since the angles and lengths are all approximately equal. In (c), even though the angles are the same as in (b), the vertical segment appears to be the parent of the other two, due to its larger length.

5.2.1 Fragment Hierarchy

As depicted in Figure 5.2, angle and branch length are two major cues that define a “hierarchy” between skeleton branches. Our system only considers angles between skeleton fragments, ignoring branch lengths. The results are nonetheless satisfactory. We shall now explain the method we use to compute this hierarchy information from a skeleton.

Outline of the Method

We start by computing a set F of skeleton fragments, and a set J of junction points from the skeleton, using the method described in Section 5.1.1. Our goal is to compute a set of fragments F_{leaf} which are the lowest in the hierarchy out of all fragments in F . Further steps of the method require the following definitions:

Definition 13 A skeleton fragment $f \in F$ is said to be **disconnected** if none of its endpoints are junctions in J .

Definition 14 A skeleton fragment $f \in F$ is said to be a **branch** if exactly one of its endpoints is a junction in J .

Since disconnected fragments have no “parent” in F , we add all disconnected fragments f_d to F_{leaf} and remove them from F . Next we consider branch fragments. For any branch f_b , we decide whether or not it should be considered as a leaf in the hierarchy (as explained shortly). We then add all fragments f_b which should be considered as leaves to F_{leaf} and remove them from F . If no such branch exists, then we add all fragments in F to F_{leaf} and remove them from F .

We then repeat the entire process (including computing the sets F and J) with the pixels corresponding to fragments in F_{leaf} reset to 0. This continues until there are no more fragments left.

After each iteration, every fragment in F_{leaf} is labelled with the current iteration number. This allows us to compute the depth in the hierarchy of each fragment. Note that we do not need

explicit parent-child relationships between fragments. As we shall see, our approach requires that we know only the depth of each node in the hierarchy.

Classifying a Fragment

We now examine the details involved in deciding whether a branch should be considered as a leaf in the hierarchy. Denote the branch under consideration by f_b . Denote the set of all fragments (not necessarily branches) which share the junction with f_b by F_j . Now for each $f_i \in F_j$, we compute the angle θ_i between f_b and f_i . If the largest such angle, $\max_i \theta_i$ is *not* within some threshold of π , we treat f_b as a leaf. Otherwise, we consider f_b to not be a leaf.

This process will obviously lead to misclassifications in cases such as the one shown in Figure 5.2(c). To remedy this problem, we would need to incorporate fragment lengths into the analysis. For now, however, we leave this as future work.

5.2.2 Splitting Fragments

Since the above process erases fragment pixels between iterations, it can lead to the merging of fragments corresponding to distinct parts (as shown in Figure 5.1(a)). To address this issue, we examine each fragment returned by the hierarchy computation process and locate pixels within the fragment where the fragment may need to be split.

We have identified the following four criteria for determining whether a fragment f should be split at a pixel p :

- **Junctions:** If p is a junction in the original skeleton, we should split f at p . This addresses the problem that when a branch is erased, junction pixels may no longer be junction pixels in the resulting binary image.
- **Turn Angle:** We first compute a piecewise linear approximation of f . Next, if p is a vertex in this polyline and if the turn angle at p is below some threshold θ_{max} , then we split f at p . This enables us to locate joints between upper and lower limbs, since in such cases the joint is characterized by sudden turn in the skeleton fragment.
- **Middle Pixel:** If p is the middle pixel in the sequence of pixels that make up f , then we may split f at p . This enables us to locate joints between upper and lower limbs in the case when there is no bend at the joint.
- **Discontinuity in Maximal Circle Radius:** If p is the location of a sudden change in the radius of the maximal circle centered at p , we split f at p . The justification for doing so is that such sudden changes typically mark a transition from part to another. For example, if the head and torso are both upright, then the sudden change can locate the pixel which separates torso from head.

Figure 5.3 shows two example splitting pixels.

Once all the n splitting pixels have been located, we compute each of the $\binom{n}{2}$ subfragments of f , formed by considering subfragments between each possible pair of splitting pixels (including the original endpoints of f). Once this is done for each skeleton fragment, the resulting subfragments are used to reconstruct the corresponding image regions (using the inverse MAT), which are in turn used in the final step of the algorithm.

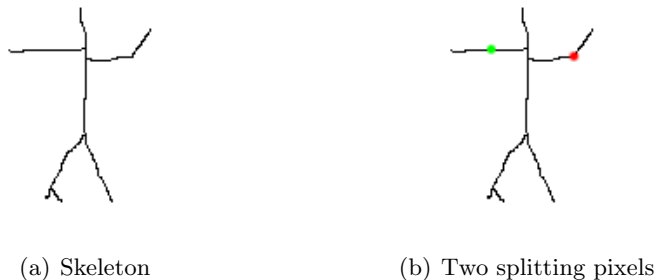


Figure 5.3: Two example splitting pixels. (a) The original skeleton (hypothetical). (b) Two splitting pixels (size exaggerated). The red pixel splits an arm at what is possibly the elbow (based on turn angle). The green pixel splits the other arm at the midpoint, attempting to separate the upper and lower arm.

5.3 A Top-Down Framework

Having obtained a set of candidate parts, we perform pose estimation using a modified pictorial structures algorithm. We use the posterior sampling framework described in Section 2.4. However, unlike the original algorithm, our location space is not a 4-dimensional grid describing a discretization of the space of all possible positions, scales and orientations. Instead, our location space is simply the set of candidate parts.

This allows us to make use of the hierarchy information computed previously (see Section 5.2.1). When considering locations for the torso, we restrict ourselves to candidate parts with depth 0. When considering locations for the upper limbs and the head, we consider candidate parts with a depth of at most 1, and for the lower limbs, we consider candidate parts with a depth of at most 2. Thus the hierarchy information allows us to further prune the search space used by the pictorial structures algorithm.

In the next few sections, we will describe the specific cost functions used by our version of the pictorial structures algorithm. All cost functions are computed based on *best-fit rectangles* computed for each candidate part.

5.3.1 Appearance Costs

The likelihood is modelled as follows. Each pixel in the image is assumed to be independently generated. Pixels within part i 's rectangle are foreground pixels with probability q_{1i} . Intuitively, this value should be close to 1, modeling the fact that most pixels within a part's rectangle are foreground pixels. Pixels within a border area around this rectangle are foreground pixels with probability q_{2i} . This number models the fact that parts tend to be surrounded by background pixels. All other pixels are equally likely to be foreground or background. Thus, we get the following expression:

$$\Pr(I | l_i) = q_{1i}^{c_{1i}} (1 - q_{1i})^{A_{1i} - c_{1i}} q_{2i}^{c_{2i}} (1 - q_{2i})^{A_{2i} - c_{2i}} (0.5)^{A - A_{1i} - A_{2i}} \quad (5.1)$$

where A_{1i} is the area of part i 's rectangle, c_{1i} is the number of foreground pixels in it, A_{2i} is the area of the border around part i 's rectangle, c_{2i} is the number of foreground pixels in it, and A is the total area of the image. Therefore, $u_i = (q_{1i}, q_{2i})$.

Learning these parameters is simple: we simply compute the average fraction of foreground pixels in each part’s rectangle over all training instances:

$$q_{1i} = \frac{\sum_{k=1}^m c_{1i}^k}{\sum_{k=1}^m A_{1i}^k} \quad (5.2)$$

$$q_{2i} = \frac{\sum_{k=1}^m c_{2i}^k}{\sum_{k=1}^m A_{2i}^k} \quad (5.3)$$

When computing the likelihood in practice, we use a dilated version of the input image while computing c_{1i} and an eroded version of the input while computing c_{2i} .

To compute the foreground counts, we need to apply a rectangular mask to the input image. This operation can be performed very efficiently using the technique of *integral images* [21]. The integral image of an image I is defined as follows:

Definition 15 *The integral image of I is the image with the same dimensions as I which contains at each location (x, y) the sum of values of pixels above and to the left of (x, y) in I :*

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \quad (5.4)$$

where $i(x', y')$ is the value of the pixel at (x', y') in I .

The integral image can be computed in one pass over I by using the following recurrence:

$$s(x, y) = s(x, y - 1) + i(x, y) \quad (5.5)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \quad (5.6)$$

It can now be seen that the number of foreground pixels within a rectangle can be computed by only 4 array lookups, as explained in Figure 5.4.

5.3.2 Structure Costs

The prior is modelled as follows. With each edge e_{ij} , we associate two points $p_{ij} = (x_{ij}, y_{ij})$ and $p_{ji} = (x_{ji}, y_{ji})$. p_{ij} is the ideal position of the joint between parts i and j in the coordinate system of part i . p_{ji} is the ideal position of the joint between parts i and j in the coordinate system of part j . Ideally, when transformed to the common image coordinate system, these two points should coincide. (See Figure 5.5.)

Roughly speaking, the prior takes into account the separation between these joint positions for a candidate configuration and the relative orientation of the parts:

$$\Pr(l_i, l_j) = \mathcal{N}(x'_i - x'_j, 0, \sigma_{x_{ij}}^2) \mathcal{N}(y'_i - y'_j, 0, \sigma_{y_{ij}}^2) \mathcal{M}(\theta_i - \theta_j, \mu_{ij}, k_{ij}) \quad (5.7)$$

The first two terms capture the difference in joint positions when transformed to the common image coordinate system. Note that (x'_i, y'_i) is the transformed version of p_{ij} , and (x'_j, y'_j) is the transformed version of p_{ji} :

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + R_{\theta_i} S_i \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} \quad (5.8)$$

$$\begin{bmatrix} x'_j \\ y'_j \end{bmatrix} = \begin{bmatrix} x_j \\ y_j \end{bmatrix} + R_{\theta_j} S_j \begin{bmatrix} x_{ji} \\ y_{ji} \end{bmatrix} \quad (5.9)$$

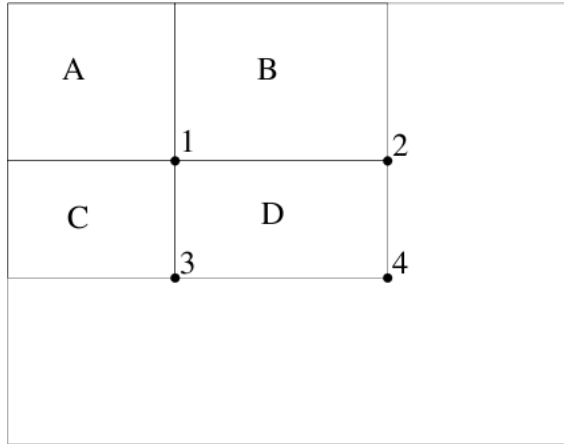


Figure 5.4: The sum of the pixel values in rectangle D can be computed in 4 array lookups. Denote the value of the integral image at marked point i in the above figure ($1 \leq i \leq 4$) by x_i and the sum of pixel values in one of the above rectangles by the name of the rectangle as marked in the figure. We have $x_1 = A$, $x_2 = A + B$, $x_3 = A + C$, $x_4 = A + B + C + D$. Therefore, the sum within rectangle D is $x_4 + x_1 - x_2 - x_3$. This requires only 4 lookups.

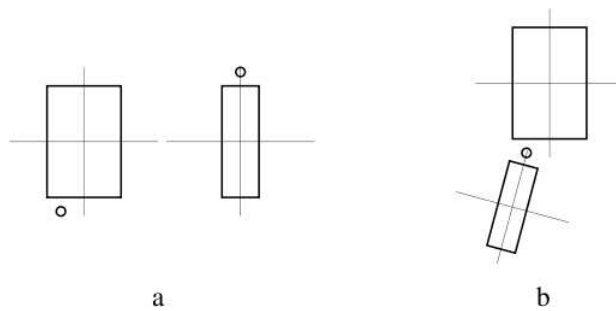


Figure 5.5: Joint positions for two parts of the pictorial structure. (a) depicts them in their own coordinate systems. (b) depicts them in the common image coordinate system.

where R_θ denotes a standard rotation matrix corresponding to a rotation by angle θ ; and S_i is a scaling matrix which scales the ideal dimensions of part i (learnt from training data) to match the dimensions of the candidate part under consideration (and similarly for S_j).

The first two terms are Gaussian distributions with 0 mean, however the third term is a von Mises distribution with mean μ_{ij} and circular variance k_{ij} . The connection parameters are therefore:

$$c_{ij} = (\sigma_{x_{ij}}, \sigma_{y_{ij}}, \mu_{ij}, k_{ij}) \tag{5.10}$$

Learning these parameters is simple: computing the variances of the Gaussian terms is straightforward; similarly there are simple ways of computing the mean and circular variance of a von Mises distribution given a set of training instances. We assume fixed values for the ideal relative joint positions. However, in practice, it is desirable to learn these from training instances. This can be done in the form of a linear least squares problem. In the interests of simplicity, we do not perform this in our implementation.

In the next chapter, we shall describe our implementation of the algorithm described in the preceding two chapters, and comment on its performance and quality of output.

Chapter 6

Results

In this chapter, we shall describe our implementation of the skeleton-based pose estimation algorithm described in Chapter 5. We shall consider the performance of our implementation as compared with our implementation of the classic pictorial structures algorithm [9], and shall compare their quality of output by comparison with ground truth data.

6.1 Implementation Details

We have implemented the classic pictorial structures algorithm, the IQP-based bottom-up algorithm, and two variants of our skeleton-based pose estimation algorithm, all in MATLAB. We now describe some of the important details of these implementations.

6.1.1 Dataset

The dataset we used was the Brown University HumanEva dataset¹. The same dataset was used to train our implementations of the classic pictorial structures algorithm, as well as the IQP-based pose estimation algorithm [16].

The ground truth data present in the HumanEva dataset describes each part as a line segment, and gives its orientation in image space. In order to obtain rectangles corresponding to parts, we have chosen widths for each part by hand based on training images and part masks provided along with the ground truth.

6.1.2 Classic Pictorial Structures

Due to the fact that the classic pictorial structures algorithm involves searching over all possible locations, significant downsampling of the image is required in order to bring the running time within acceptable limits. We also have to discretize the range of orientations used by the algorithm. In our implementation, we consider only angles which are multiples of 45 degrees.

We also make the simplifying assumption that the variation is foreshortening between parts is negligible, and hence fix the value of the s parameter in location space for all parts.

6.1.3 CDT/IQP-based Algorithm

Our implementation has the following major components:

¹<http://vision.cs.brown.edu/humaneva/>

1. **Boundary Detection.** We use the Berkeley Segmentation Engine² for computing Pb maps. Hysteresis thresholding is performed using MATLAB's Image Processing Toolbox.
2. **Constrained Delaunay Triangulation.** We use the open-source Computational Geometry Algorithms Library (CGAL)³ for computing CDTs. All code dealing with CDTs and CDT-based graphs is written in C++.
3. **Logistic Classifier.** The logistic classifier used for detecting contour edges and candidate parts is implemented using the NETLAB Toolbox⁴. The classifiers are trained using a hand-labeled dataset consisting of 10 images from the HumanEva dataset.
4. **Integer Quadratic Programming.** Finally, the LP approximation to the IQP problem is solved using MATLAB's Optimization Toolbox.

6.1.4 Skeleton-Based Pose Estimation

We have implemented most of the pose estimation algorithm in MATLAB. For the skeleton computation phase, we use the Berkeley Segmentation Engine for computing Pb maps, an existing C implementation of Ratio Contour⁵, and an existing MATLAB implementation of DCE-based skeleton pruning⁶.

We have also implemented a version of our skeleton-based algorithm which uses background subtraction to locate salient boundaries, instead of the Pb operator followed by Ratio Contour. We use this version of our algorithm primarily for comparison against the classic pictorial structures algorithm, which also relies on background subtraction.

6.2 Experimental Results

We learn the human model from a set of 62 training images. The total time required for training is approximately 1 minute. Figure 6.1 shows the learned model. Before we describe our experimental results, we shall outline our strategy for comparing our method with existing methods.

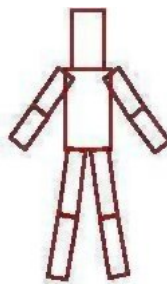


Figure 6.1: The learned human model.

²<http://www.cs.berkeley.edu/~fowlkes/BSE/>

³<http://www.cgal.org/>

⁴<http://www.ncrg.aston.ac.uk/netlab/>

⁵<http://www.cse.sc.edu/~songwang/>

⁶<http://xiang.bai.googlepages.com/softwareforskeletonizationandskeletonpru>

Algorithm	Resolution	Time (s)	Accuracy	Remarks
PS	160×120	3491.69	0.49	
Sk-RC	160×120	109.06	0.38	Does not include part detection.
Sk-BG	160×120	11.69	0.69	Does not include part detection.

Table 6.1: Comparison between our pose estimation algorithm and classic pictorial structures.

6.2.1 Evaluation Strategy

We have compared our pose estimation algorithm with the classic pictorial structures algorithm with respect to performance and quality of the estimated pose. We have also compared our skeleton-based part detector with the CDT-based part detector with respect to performance. Our methodology is as follows.

We took a set of 3 test images from the HumanEva dataset. Using this test set, we measured (in MATLAB) the average time required for processing an image using the classic pictorial structures algorithm (PS), our skeleton-based pose estimation algorithm with our skeleton-based part detector (Sk-RC), our skeleton-based part detector with background subtraction (Sk-BG), and the CDT-based part detector (CDT).

In order to compare the quality of the estimated pose, we computed a binary image of the ground truth pose, and compared it with a binary image of the estimated poses using normalized cross-correlation, and used the score of the highest-scoring pose.

6.2.2 Results

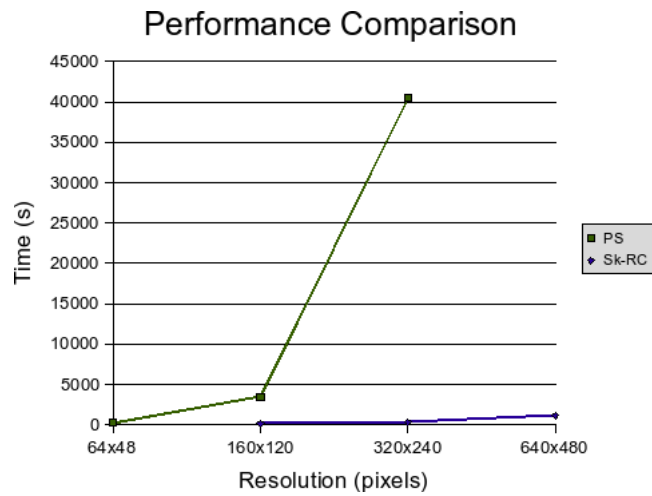
Two examples of the output at each step of the pose estimation process are shown in Figures 6.3 and 6.4. In order to assess the usefulness of the boundary detection and completion phases, we compared our skeleton pipeline with a two step process consisting of background subtraction followed by pruned skeleton computation. Two examples of each step of this process are shown in Figures 6.5 and 6.6.

Table 6.1 compares the average running times and quality of both versions of our skeleton-based pose estimation algorithm and the classic pictorial structures algorithm. Table 6.2 compares the average running times of both versions of our skeleton-based part detector and the CDT-based part detector.

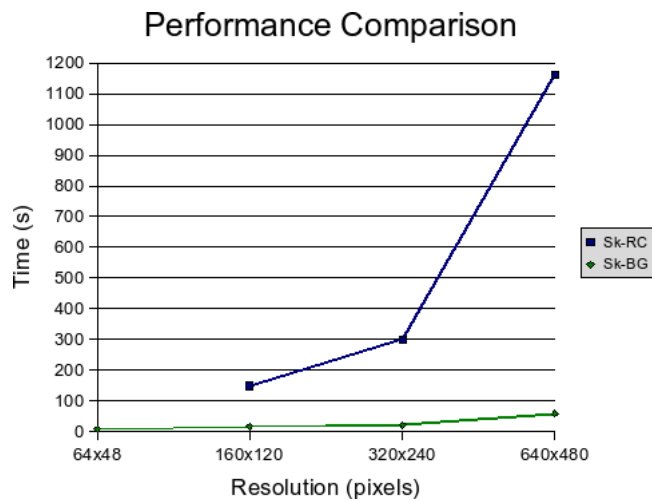
Figure 6.2 shows how the performance of the classic pictorial structures algorithm and both versions of our skeleton-based pose estimation algorithm varies with resolution. Due to the fact that classic pictorial structures involves a search over all locations, using images beyond a resolution of, say, 160×120 leads to unacceptably high running times. At the other extreme, our skeleton-based algorithm with background subtraction handles resolutions upto 640×480 with ease. (Note that in our implementation, background subtraction is performed at 640×480 irrespective of the resolution that the rest of algorithm uses. The results of the background subtraction are then scaled appropriately.) Performance and memory issues prevented testing of classic pictorial structures at 640×480 . Also, we had trouble recovering useful contour information using Ratio Contour at 64×48 , which is the cause of the missing data point in the plots.

The results highlight the advantages of our algorithm over existing methods. However, some issues may be noted from the results.

There is difficulty in distinguishing between upper and lower limbs (which is why lower limbs are not shown in the figures, they were detected in exactly the same configuration as the upper limbs). This occurs due to the fact that the skeleton branches do not distinguish between upper



(a)



(b)

Figure 6.2: Comparison between classic pictorial structures and both versions of our skeleton-based algorithm based on performance.

Algorithm	Resolution	Time (s)
CDT	160 × 120	93.74
Sk-RC	160 × 120	39.64
Sk-BG	160 × 120	4.29

Table 6.2: Comparison between skeleton-based part detection and CDT-based part detection.

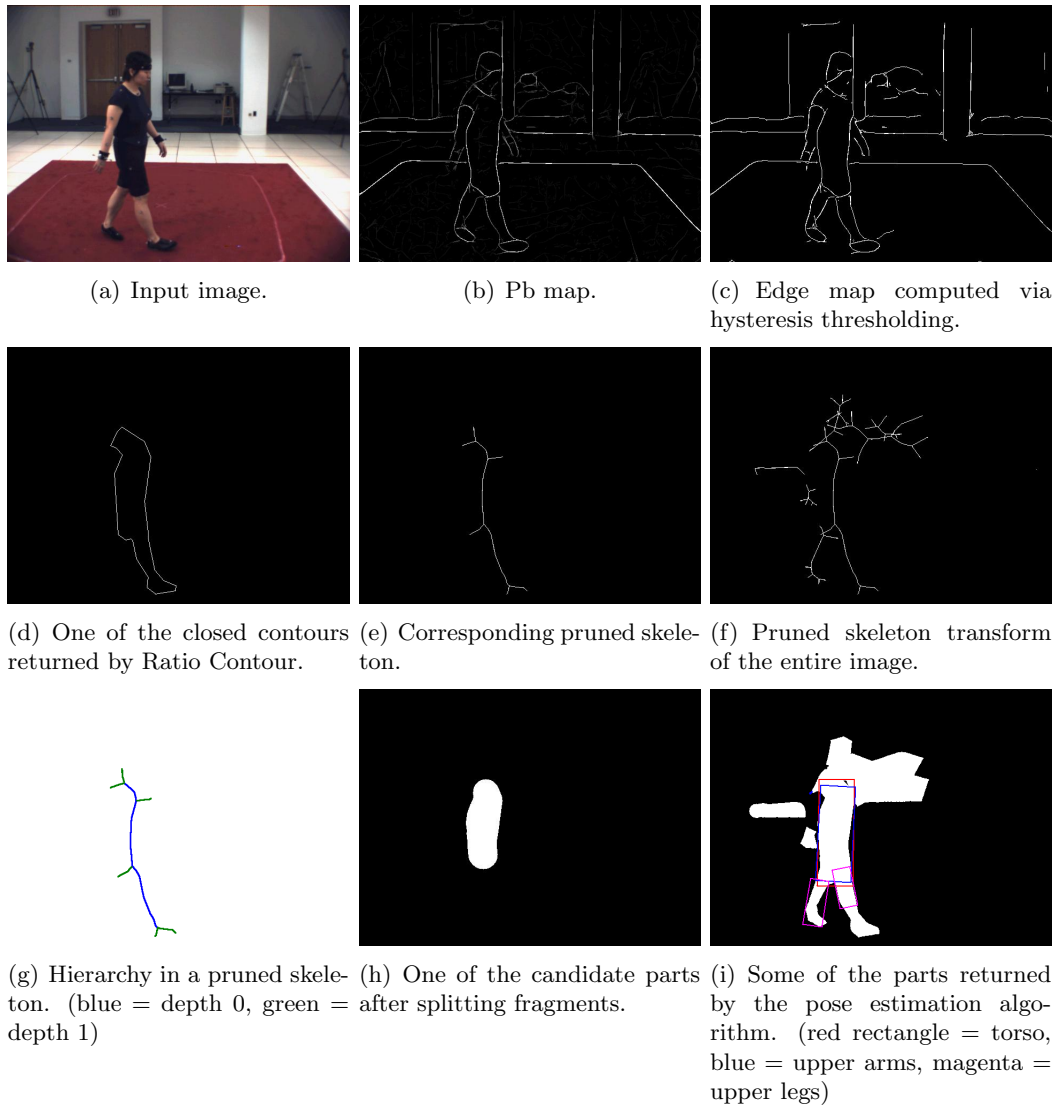


Figure 6.3: Results: test image 1

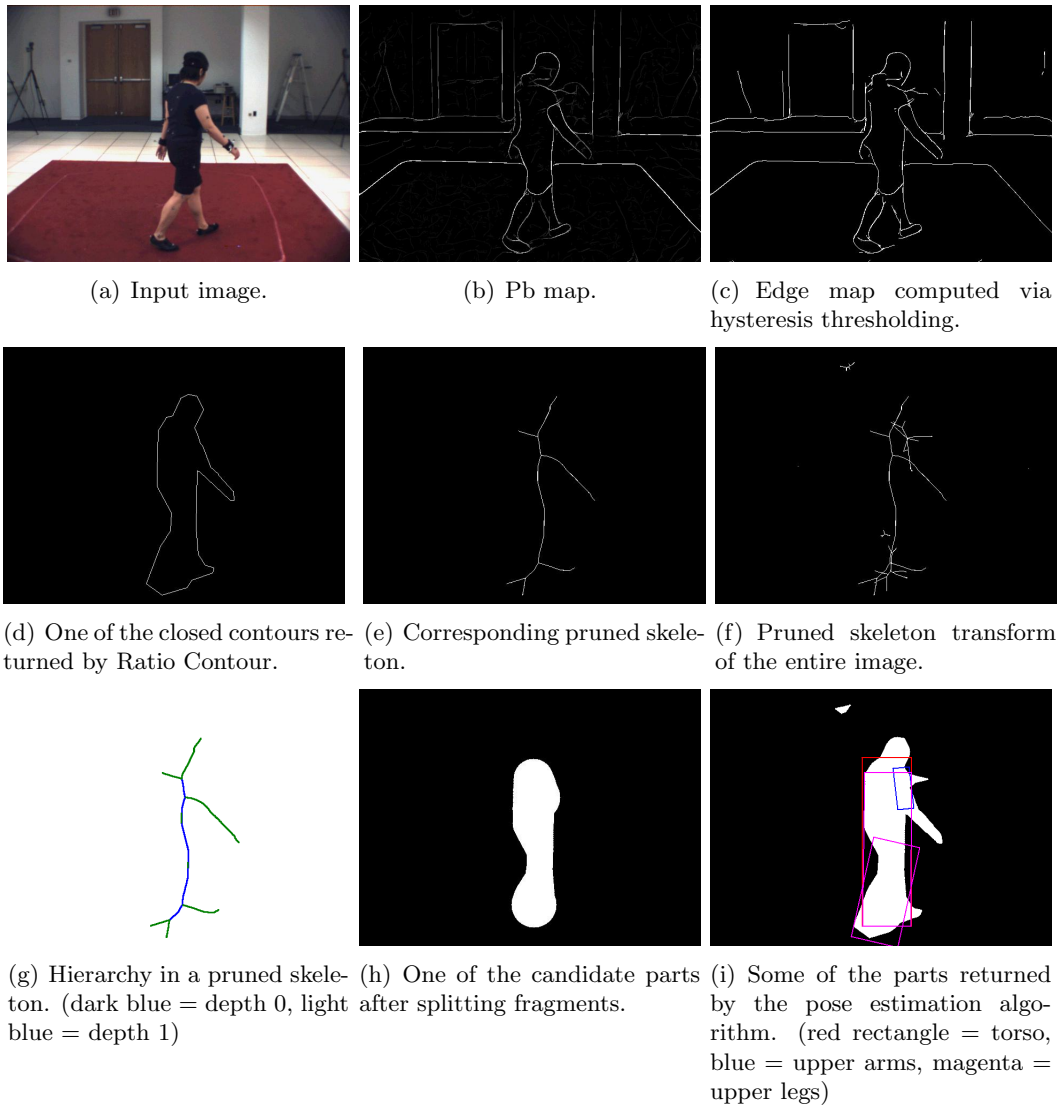


Figure 6.4: Results: test image 2

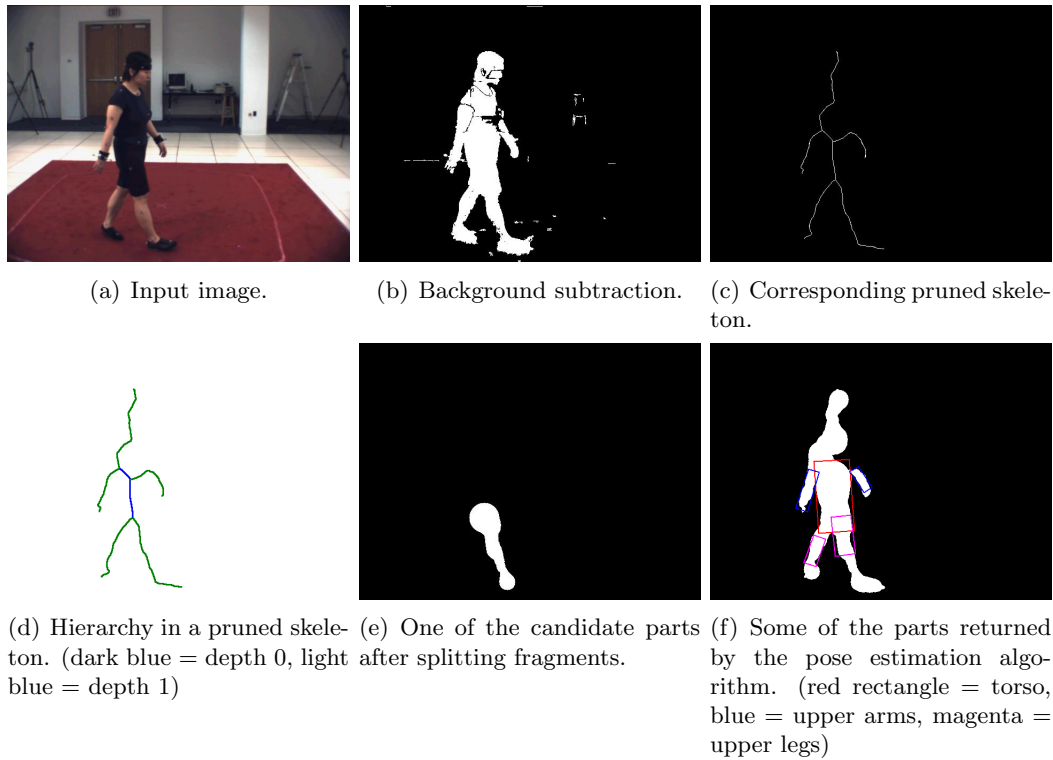


Figure 6.5: Results: test image 1 with background subtraction

and lower limbs, and we have to split the branches in a somewhat arbitrary manner in order to attempt to make the distinction.

Also note how in Figure 6.4, Ratio Contour has merged the two legs. This is a natural consequence of its cost function, and leads to issues with the corresponding skeleton transform (the torso is too elongated and legs are too short and at unnatural angles).

Although the quality of the results seems to be better, requiring background subtraction is too strong a restriction. We therefore believe that it would be worthwhile to consider improvements in the boundary detection and completion phases of our skeleton pipeline.

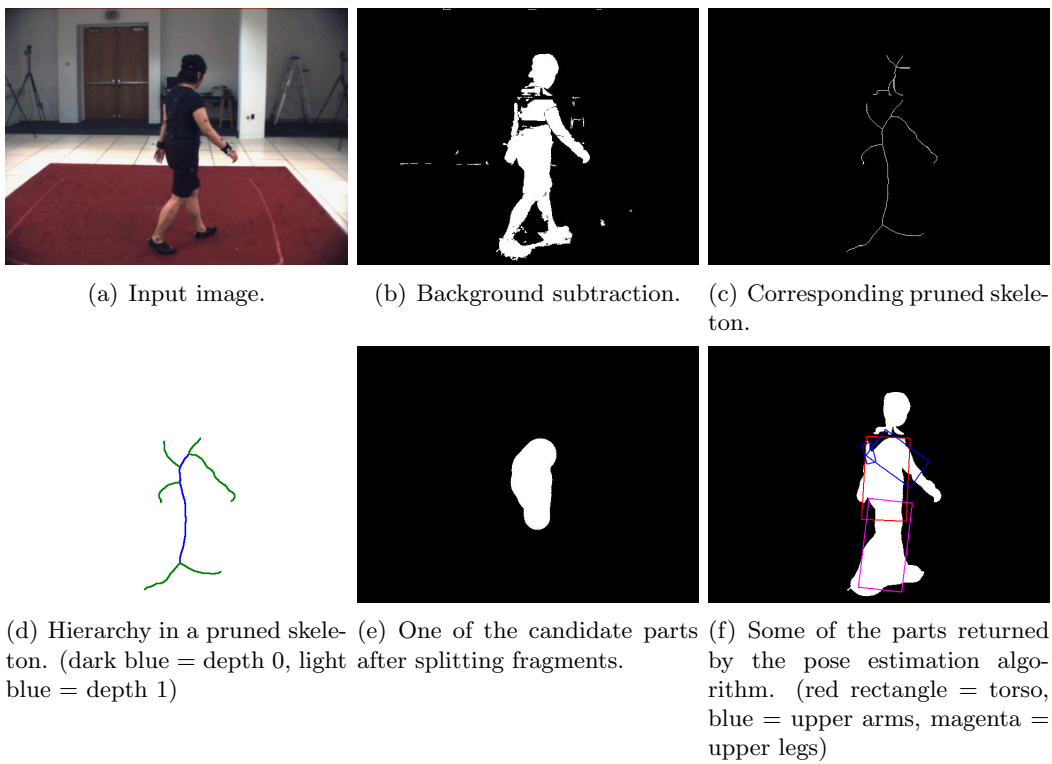


Figure 6.6: Results: test image 2 with background subtraction

Chapter 7

Conclusion

In the preceding chapters, we have described a skeleton-based pose estimation algorithm and compared it with existing top-down and bottom-up approaches. In this chapter, we shall summarize our findings and outline ideas and improvements for future work.

7.1 Discussion

We have seen the clear advantage of combining top-down and bottom-up approaches to human pose estimation. In particular, we have seen how using a bottom-up part detector noticeably speeds up top-down pose estimation by significantly reducing the search space. This allows us to deal with input images at a much higher resolution than is possible with, say, the pictorial structures algorithm. It also allows us to incorporate the structure of the human body in our search for the correct pose.

Another important conclusion of our work is that a skeleton-based part detector is in several ways more preferable to an edge-based part detector. Intuitively speaking, this is because the skeleton captures the structure of an object in a more explicit manner than the boundary, and this makes it easier for us to locate parts. Moreover, the implicit structure can be extracted (to some extent), and can be used to reconstruct hierarchical relationships between candidate parts. This allows us to impose a structure on the part space, which can be further exploited by a top-down pose estimation algorithm.

However, several issues remain to be addressed, and there is much potential for improvement. In the next section, we shall briefly outline a few such ideas.

7.2 Future Work

Although our pose estimation framework is a complete system, there remain a few issues that need to be addressed, and there is much scope for improvement and further study. We shall now briefly outline our ideas for future work in this direction.

As can be seen from the output (see Figure 6.4(d)), Ratio Contour is not the best choice of contour completion algorithm for our purposes. We have already discussed the issues with Ratio Contour in Section 4.1.2, and have briefly described one possible alternative [7]. This new approach incorporates appearance into the boundary cost function. The benefit of such a cost function is that for poses where the legs are close together, an appearance-based cost function will correctly return a V-shaped concavity in the boundary (thus distinguishing between the legs), whereas a purely geometric cost function such as the one used by Ratio Contour will tend to join edge fragments from both legs into a more circular shape.

Another approach to computing improved boundaries (and hence improved skeletons) would be to combine the boundary detection and skeleton computation steps. For any candidate closed boundary, we can compare its corresponding pruned skeleton with some skeletal model (stick figure) of the human body, and modify the candidate boundary based on the comparison. Since such an algorithm by definition is more likely to return human-like boundaries and skeletons, we believe that this approach is worth studying.

The appearance and structure cost functions that we use can be improved upon. In particular, the binary appearance cost functions used by the IQP-based pose estimation algorithm [16] might make a good addition to our current set of cost functions. However, its incorporation will lead to the pictorial structure model no longer being a tree, but a directed acyclic graph. Of course, investigation of other, entirely new cost functions is another possibility.

An important issue with our procedure to extract hierarchy information from a pruned skeleton is the fact that it makes use of angle-based cues only. As we have previously shown (Figure 5.2), incorporating length in this process can help resolve ambiguities. We would need to carefully choose which fragments' lengths we use in the comparison, and this would make for an interesting direction of study.

Although we have chosen not to use a classifier to discard pruned skeletons which are not "human-like", this possibility may also be explored. We would need a way to stitch together multiple skeleton fragments which individually may not look human-like, but may be two halves of a human skeleton. However, this requirement may be alleviated to some extent by an improved contour completion algorithm.

Finally, although we can perform tracking over multiple images by repeated use of our algorithm, we do not exploit inter-frame coherence in any way. Adding such a capability to our algorithm would be one of the most important areas of future work.

Bibliography

- [1] AMENTA, N., CHOI, S., AND KOLLURI, R. K. The power crust, unions of balls, and the medial axis transform. *Computational Geometry* 19, 2-3 (2001), 127–153.
- [2] AMIT, Y., AND GEMAN, D. Shape quantization and recognition with randomized trees. *Neural Computation* 9, 7 (1997), 1545–1588.
- [3] BAI, X., LATECKI, L. J., AND LIU, W.-Y. Skeleton pruning by contour partitioning with discrete curve evolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 3 (2007), 449–462.
- [4] BRAY, M., KOHLI, P., AND TORR, P. H. S. Posecut: Simultaneous segmentation and 3D pose estimation of humans using dynamic graph-cuts. In *ECCV (2)* (2006), pp. 642–655.
- [5] EFROS, A. A., BERG, A. C., MORI, G., AND MALIK, J. Recognizing action at a distance. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision* (Washington, DC, USA, 2003), IEEE Computer Society, p. 726.
- [6] ESTRADA, F. J., AND ELDER, J. H. Multi-scale contour extraction based on natural image statistics. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop* (Washington, DC, USA, 2006), IEEE Computer Society, p. 183.
- [7] ESTRADA, F. J., AND JEPSON, A. D. Robust boundary detection with adaptive grouping. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop* (Washington, DC, USA, 2006), IEEE Computer Society, p. 184.
- [8] FATHI, A., AND MORI, G. Human pose estimation using motion exemplars. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* (14-21 Oct. 2007), 1–8.
- [9] FELZENSZWALB, P. F., AND HUTTENLOCHER, D. P. Pictorial structures for object recognition. *Int. J. Comput. Vision* 61, 1 (2005), 55–79.
- [10] LEVIN, A., AND WEISS, Y. Learning to combine bottom-up and top-down segmentation. In *ECCV (4)* (2006), pp. 581–594.
- [11] LUCAS, B., AND KANADE, T. An iterative image registration technique with an application to stereo vision. In *IJCAI81* (1981), pp. 674–679.
- [12] MARTIN, D. R., FOWLKES, C. C., AND MALIK, J. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 5 (2004), 530–549.
- [13] MITCHELL, T. *Machine Learning*. McGraw-Hill, 1997.

- [14] OGNIWICZ, R. Automatic medial axis pruning by mapping characteristics of boundaries evolving under the euclidean geometric heat flow onto voronoi skeletons, 1995.
- [15] RAMANAN, D. Learning to parse images of articulated bodies. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, Cambridge, MA, 2007, pp. 1129–1136.
- [16] REN, X., BERG, A. C., AND MALIK, J. Recovering human body configurations using pairwise constraints between parts. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 824–831.
- [17] REN, X., FOWLKES, C. C., AND MALIK, J. Scale-invariant contour completion using conditional random fields. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 1214–1221.
- [18] SHI, J., AND TOMASI, C. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)* (Seattle, June 1994).
- [19] SRINIVASAN, P., AND SHI, J. Bottom-up recognition and parsing of the human body. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).
- [20] TELEA, A., SMINCHISESCU, C., AND DICKINSON, S. Hierarchical skeleton abstraction.
- [21] VIOLA, P., AND JONES, M. Robust real-time object detection. *International Journal of Computer Vision* (2002).
- [22] WANG, S., KUBOTA, T., SISKIND, J. M., AND WANG, J. Salient closed boundary extraction with ratio contour. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 4 (2005), 546–561.
- [23] WELLS, W. M. Efficient synthesis of gaussian filters by cascaded uniform filters. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1986), pp. 8(2):234–239.
- [24] WIKIPEDIA. Hilbert transform (en.wikipedia.org/wiki/Hilbert_transform).