A real-world system for human motion detection and tracking

David Moore

California Institute of Technology

dcm@acm.org

June 5, 2003

Acknowledgements

The author wishes to thank Professor Pietro Perona for serving as the research advisor to this project. His mentorship and valuable advice were instrumental in the course of this research. Much thanks also goes to Yang Song for sharing the details and results of her research on human motion, and for her assistance in starting this project. The author would also like to acknowledge Professor Richard Murray for serving as second reader on this paper, and Kent Potter for organizing the senior thesis course and offering useful suggestions on the work.

Abstract

In this thesis we present an operational computer vision system for real-time detection and tracking of human motion. The system captures monocular video of a scene and identifies those moving objects which are characteristically human. This serves as both a proof-of-concept and a verification of other existing algorithms for human motion detection. An approach to statistical modeling of motion developed by Y. Song is coupled with a pre-processing stage of image segmentation and point feature tracking. This design allows a system that is robust with respect to occlusion, clutter, and extraneous motion. The results of experiments with the system indicate the ability to minimize both false detections and missed detections.

Contents

1 Introduction **3**
1.1 Motivation 4
1.2 Justification 5
1.3 Previous Work 6

2 Approach **7**
2.1 Understanding the recognition problem 8
2.2 Image segmentation 9
2.3 Point feature tracking 12
2.4 Motion model detection 16
2.5 Summary 19

3 Implementation **20**
3.1 Software overview 21
3.2 Background subtraction 22
3.3 Kanade-Lucas-Tomasi feature tracking 25
3.4 Human motion detection 27

4 Results **29**
4.1 Method of Performance Analysis 29
4.2 Results of Performance Analysis 31
4.3 Computational Cost 33

5 Conclusions **35**
5.1 Future Work 36

Chapter 1

Introduction

The field of computer vision is concerned with problems that involve interfacing computers with their surrounding environment through visual means. One such problem, object recognition, involves detecting the presence of a known object in an image, given some knowledge about what that object should look like. As humans, we take this ability for granted, as our brains are extraordinarily proficient at both learning new objects and recognizing them later. However, in computer vision, this same problem has proven to be one of the most difficult and computationally intensive of the field. Given the current state of the art, a successful algorithm for object recognition requires one to define the problem with a more specific focus.

In this thesis, we consider a sub-problem of object recognition: human motion detection, in which we are interested in recognizing humans based solely on the characteristic patterns of motion that they exhibit. This approach differs from other techniques for human detection, such as those that recognize humans based on shape, color, texture, or surface features. This thesis presents a fully realized system for human motion detection that can be deployed in the field. Its characteristics include real-time performance, insensitivity to background clutter and movement, and a modular design that can be generalized to other types of motion.

1.1 Motivation

The ability to reliably detect and track human motion is a useful tool for higher-level applications that rely on visual input. Interacting with humans and understanding their activities are at the core of many problems in intelligent systems, such as human-computer interaction and robotics. An algorithm for human motion detection digests high-bandwidth video into a compact description of the human presence in that scene. This high-level description can then be put to use in other applications.

Some examples of applications that could be realized with reliable human motion detection and tracking are:

- Automated surveillance for security-conscious venues such as airports, casinos, museums, and government installations: Intelligent software could monitor security cameras and detect suspicious behavior. Futhermore, human operators could search archived video for classes of activity that they specify without requiring manual viewing of each sequence. Having automated surveillance vastly increases the productivity of the human operator and increases coverage of the surveillance.

- Human interaction for mobile robotics: Autonomous mobile robots in the workplace or home could interact more seamlessly with the humans in their environment if they could reliably detect their presence. For example, robots to assist the elderly would know when assistance is needed based on the motion of a person.

- Safety devices for pedestrian detection on motor vehicles: Intelligent software on a camera-equipped car could detect pedestrians and warn the driver.

- Automatic motion capture for film and television: Producing computer-generated imagery of realistic motion currently requires the use of a motion-capture system that stores the exact 2-D or 3-D motion of a human body using visual or radio markers attached to each limb of an actor. With accurate algorithms for human motion tracking,

the same data could be acquired from any video without any additional equipment.

Currently, no algorithm exists that can perform human motion detection reliably and efficiently enough for the above applications to be realized. Although the problem as a whole remains unsolved, many of the tools necessary for a robust algorithm have been developed. By assembling these task-specific tools into a working system, this thesis will show that a robust system is not far from realization.

1.2 Justification

Detection of a human based only on motion may, at first, seem far-fetched. Do the motion of the limbs contain enough information to infer the presence of a human? Experiments performed by Johansson in the 1970's demonstrated the answer to be 'Yes'. Johansson filmed moving humans in a pitch-black room, the only visual indicator being a white point of light attached to each limb. He showed that a viewer watching the film could easily identify human motion, despite the absence of visual cues such as shape, texture, brightness, and color [1]. An example of these Johansson points is shown in Figure 1.1. It has been further demonstrated that specific individuals or genders can be recognized in the same manner [2, 3]. Given that the human brain can effortlessly recognize this motion, it is conceivable that a computer algorithm could do the same. In addition, single points of motion as used in the Johansson experiment can be efficiently represented on a computer. Unlike pure image processing, which must deal with large numbers of pixels at each time step, this Johansson motion can be specified by a handful of points, each represented by a 2-D position and a 2-D velocity at any given time. This gives us hope that a simple, effective algorithm is achievable.

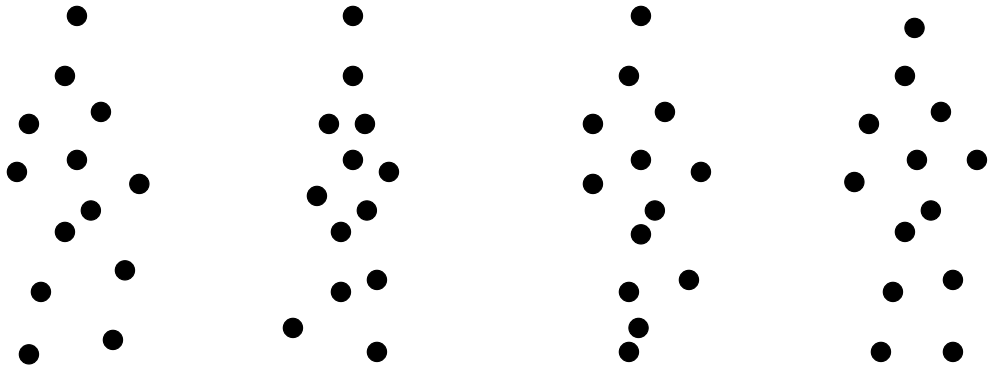


Figure 1.1: An example sequence of Johansson points that show the side view of a walking human. Taken one image at a time, the shape of the human figure is not completely apparent. However, when considered in sequence, the images clearly depict a human.

1.3 Previous Work

The work of this thesis is an extension of the algorithms developed by Song for model-based human motion detection [4]. Her algorithms for unsupervised learning and model detection provide the groundwork for the real-world system described herein. There are a variety of other techniques that have been developed for human motion detection, usually falling into one of two categories: monocular detection or 3D reconstruction. Monocular detection techniques recognize human motion from a single viewpoint, either using statistical modeling or shape-based analysis [5, 6, 7]. 3D reconstruction techniques first triangulate the volume of the human before applying a recognition algorithm or use multiple viewpoints to increase the accuracy of a monocular approach [8].

Chapter 2

Approach

The Johansson experiment shows us that human detection from the motion of point features is a realistic goal. With this premise in mind, we can divide the system into several sub-problems:

1. Distilling full-frame video into the motion of individual point features.
2. Finding a model of motion that accurately represents human motion.
3. Apply that model to a detector that can determine if a cluster of moving points is representative of human motion.

Problems 1 and 3 are the subject of this thesis. Problem 2, finding a model for human motion, has been studied by Y. Song, who has developed unsupervised learning algorithms for developing a model of human motion from video [4]. The statistical models used in that research have been borrowed for use in the this project. In addition, the detection algorithm used in the Song research has been adapted to this project and is discussed in greater detail in Section 2.4.

2.1 Understanding the recognition problem

The approach taken by Song in recognizing human motion is a probabilistic one, in which the common features and variances of the human gait are encapsulated in a single statistical model [4]. This model is stored as a graph containing vertices and edges. Each vertex constitutes a point feature of motion somewhere on the human figure. The vertex is represented by a Euclidean position (x, y) indicating the mean position of the feature on the human body, as well as a mean velocity (v_x, v_y). In addition, there is a covariance matrix that relates these four parameters to those of every other vertex. Any two vertices connected by an edge are considered to be probabilistically dependent on each other, and those unconnected are considered independent. Although the details of the detection algorithm that uses this model are discussed in Section 2.4, it is important to note here that its runtime efficiency is $\mathcal{O}(M\,N^3)$, where M is the number of triangles in the graph and N is the number of point features we are evaluating. Thus, for efficiency reasons, it is in our best interest to keep the number of point features in each evaluation to a minimum.

Reducing the number of point features in each model evaluation requires us to be more intelligent about choosing them in the first place. It would be impractical to simply find the point motion throughout the entire image, and feed all resulting features to the model in a single evaluation. First of all, the number of features would be very large, giving poor performance. Secondly, we would be supplying a large number of extraneous point features, such as those that are part of the background. In order to avoid wasting time on the background, we use *image segmentation* to separate foreground portions of the scene from the background.

Image segmentation is the action of any algorithm that separates regions of an image in a way that resembles how a human would naturally perceive them. Since we are interested in motion, a natural approach is to segment those regions of the image that are moving relative to the background. This process is called *background subtraction*, and is discussed further in

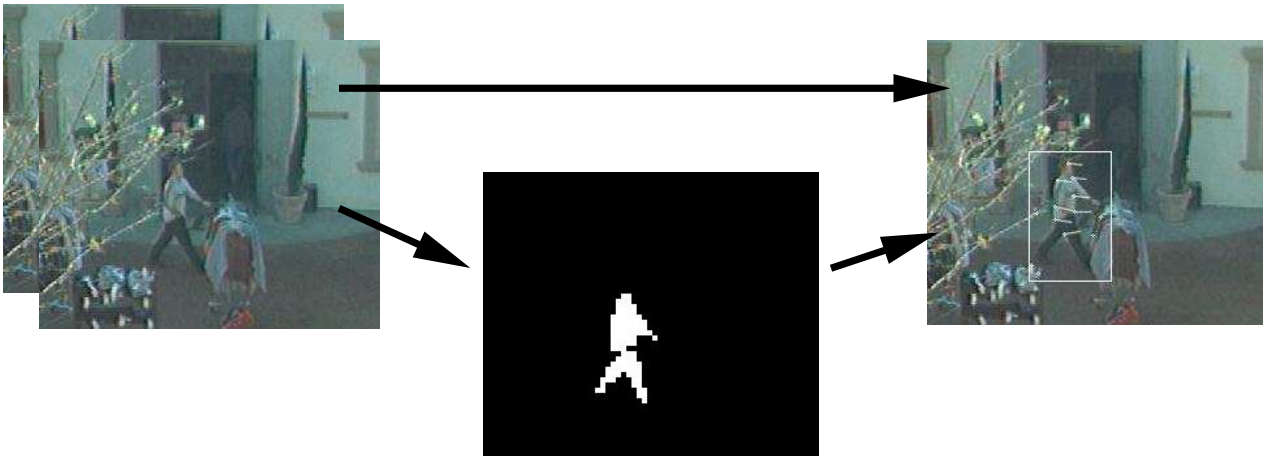


Figure 2.1: The first two stages, image segmentation and point feature tracking, of our human motion detection system. On the left is an input video sequence, which is used to mask those regions in motion, center. The resulting image segments are then used for point feature tracking, shown on the right. A box encloses the region of the image considered in motion.

Section 2.2.

Once we have segmented the image according to motion, point features can be tracked separately in each region. This allows us to run the model evaluation individually for each region, rather than on the entire image. Since the model evaluation has runtime $\mathcal{O}(M\,N^3)$, it runs faster on two regions of 20 points each than on one region of 40 points. The point tracking algorithm is discussed in Section 2.3.

These two basic steps, image segmentation and point feature tracking, constitute the first half of a system for human motion detection and tracking. An example of these algorithms running in the field is shown in Figure 2.1.

2.2 Image segmentation

Our goal in image segmentation is to separate background areas of the image from foreground regions of motion that are of interest for human tracking. In this project, we make the fundamental assumption that the background will remain stationary. This assumption

necessitates that the camera be fixed and that lighting does not change suddenly. It is possible to achieve accurate image segmentation without this assumption, but such generality would require more computationally expensive algorithms. Given our assumption, the algorithm of choice is *background subtraction*, in which we compute a model of the image background over time. For any given frame of video, we can subtract this background image from it. Those pixels with a result near zero are treated as background and those pixels with a larger result are treated as foreground. Thus, once we have the model of the background image, this algorithm is simple, efficient, and easy to implement.

Acquiring the background model, on the other hand, is more complicated. The most straightforward approach would be to simply set up the camera, empty the scene of any moving objects, and take a snapshot. Although this approach is simple, it is always impractical in real scenes because backgrounds can change over time, it can be difficult to empty a scene, lighting can change subtly, and the camera position can drift. A more practical approach is one that can adapt to a slowly changing background in real-time, which we will now describe [9].

Consider the time-varying value of a pixel at position (x, y) of a grayscale video sequence. We will refer to this value as $V_{x,y}(t)$. We can treat the value as a random process of variable X_t,

$$X_t = V_{x,y}(t).$$

Now, suppose we can model the probability of observing the current pixel value as a mixture of K Gaussian distributions. This probability is,

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t}\, \eta(X_t, \mu_{i,t}, \Sigma_{i,t})$$

where $\omega_{i,t}$ is an estimate of the weight of the i^{th} Gaussian, and η is the evaluation of a

standard Gaussian with mean $\mu_{i,t}$ and covariance matrix $\Sigma_{i,t}$:

$$\eta(X, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_t)^T\Sigma^{-1}(X-\mu_t)}.$$

Since the background is assumed to be static, the value of pixels which are part of the background can be represented by one or more Gaussians with a small variance due to image noise alone. More than one Gaussian is a possibility for bimodal scenes such a trees swaying in the wind or a flashing light. Furthermore, in most scenes, the background will be visible more often than foreground at any given pixel, so the Gaussian with the largest weight ω is likely the background.

These ideas now enable the following approach to background subtraction:

- For each pixel in a frame of video:
    - Consider the last N values taken by the pixel.
    - Find the K Gaussians and weights that best fit this sample of N values using an algorithm such as K-Means or Expectation Maximization (EM).
    - Choose the Gaussian with the largest weight ω and store its mean as the value of the background image for that pixel.
- Subtract the background image from the frame.
- In the resulting difference image, any value larger than three standard deviations from the mean is considered foreground, and any other value is considered background.

The preceeding algorithm is too computationally intensive for real-time use, especially the step of fitting K Gaussians to the data for each pixel and every frame. To simplify, the background image itself need only be recomputed every N frames. Thus, for most time steps, values of each pixel are simply collected and stored for later processing that only occurs once every N frames. The disadvantage of this approach is some lag time before the background

Figure 2.2: An example of the Kanade-Lucas-Tomasi point tracking algorithm in operation. Features being tracked are shown as a white dot, with the inter-frame displacement shown as a white line. The length of the line is exaggerated to show motion. This frame was captured from a sequence of video running at 30 frames per second.

can adapt to new stationary objects. A few more approximations that result in speed gains are described in Section 3.2, which presents the exact implementation of the algorithm used in this project.

2.3 Point feature tracking

The image segmentation step allows us to separate foreground objects from the scene background. However, we are still working with full images, not the individual points of motion desired for human motion detection. The problem of computing the motion in an image is known as finding the *optical flow* of the image. There are a variety of well-understood techniques for doing so, but the Kanade-Lucas-Tomasi method stands out for its simplicity and lack of assumptions about the underlying image [10]. A simple example of this algorithm in operation is shown in Figure 2.2.

The most naive algorithm for point feature tracking between two frames of video is

- Choose a small window, say 7 pixels on a side, around a pixel of interest in frame 1. This pixel of interest will be called pixel A.
- For each pixel near A in frame 2, call it pixel B, and perform the following:
    - Subtract the value of each pixel in the 7 by 7 region around pixel A from each pixel in the 7 by 7 region around pixel B. Square the result of the difference, and sum these 49 values to produce a 'dissimilarity' for this choice of pixel B.
- The pixel B in frame 2 with the smallest dissimilarity is considered to be the new location of pixel A in frame 1.

Figure 2.3: Naive algorithm for computing the displacement of a point feature between two images

outlined in Figure 2.3. Although this algorithm would give us a new position and velocity for the feature represented by pixel A, it would suffer from several flaws. First, it would be slow, requiring about a hundred computations for each iteration, and potentially hundreds of iterations depending on how far we want to search. Second, the algorithm would only give us the position and velocity of the feature to the nearest whole pixel. If the feature actually moved by one and a half pixels, we would compute either one or two. The Kanade-Lucas-Tomasi algorithm alleviates these problems by using the image's gradients to predict the new location of the feature—iterating until the new location is converged upon. Since this approach is based on a Taylor series expansion, it makes no assumptions about the underlying image.

The following derivation summarizes the iterative step of the Kanade-Lucas-Tomasi algorithm [11]. Consider two images, I and J, represented as continuous functions in two dimensions. We want to track a feature of known location $\mathbf{x}' = [x, y]^T$ in image I to image J, finding its displacement $\mathbf{d} = [d_x, d_y]^T$. Given a window W, we can compute the dissimilarity ϵ between the new and old feature as

$$\epsilon = \iint_W \left[J(\mathbf{x}') - I(\mathbf{x}' - \mathbf{d})\right]^2 \, d\mathbf{x}'.$$

We can make this relationship symmetric by making the substitution $\mathbf{x}' = \mathbf{x} + \frac{\mathbf{d}}{2}$:

$$\epsilon = \iint_W \left[J(\mathbf{x} + \frac{\mathbf{d}}{2}) - I(\mathbf{x} - \frac{\mathbf{d}}{2}) \right]^2 d\mathbf{x}.$$

Given this expression for dissimilarity, we want to solve for the value of $\mathbf{d}$ that minimizes ϵ. Thus, we find the value of $\mathbf{d}$ that solves the equation,

$$\frac{\partial \epsilon}{\partial \mathbf{d}} = 0 = 2 \iint_W \left[J(\mathbf{x} + \frac{\mathbf{d}}{2}) - I(\mathbf{x} - \frac{\mathbf{d}}{2}) \right] \left[\frac{\partial J(\mathbf{x} + \frac{\mathbf{d}}{2})}{\partial \mathbf{d}} - \frac{\partial I(\mathbf{x} - \frac{\mathbf{d}}{2})}{\partial \mathbf{d}} \right] d\mathbf{x}. \tag{2.1}$$

In order to make it possible to solve for $\mathbf{d}$, we can express the value of the displaced images by their Taylor series expansion, approximating terms of second-order or higher derivatives as zero in

$$J(\mathbf{x} + \frac{\mathbf{d}}{2}) \approx J(\mathbf{x}) + \frac{d_x}{2} \frac{\partial J}{\partial x}(\mathbf{x}) + \frac{d_y}{2} \frac{\partial J}{\partial y}(\mathbf{x})$$

and,

$$I(\mathbf{x} - \frac{\mathbf{d}}{2}) \approx I(\mathbf{x}) - \frac{d_x}{2} \frac{\partial I}{\partial x}(\mathbf{x}) - \frac{d_y}{2} \frac{\partial I}{\partial y}(\mathbf{x}).$$

Equation (2.1) can now be approximated as:

$$\frac{\partial \epsilon}{\partial \mathbf{d}} \approx \iint_W \left[J(\mathbf{x}) - I(\mathbf{x}) + \frac{1}{2} \mathrm{g}^T(\mathbf{x}) \mathbf{d} \right] \mathrm{g}(\mathbf{x})\, d\mathbf{x} = 0,$$

where

$$\mathrm{g} = \begin{bmatrix} \frac{\partial}{\partial x}(I + J) \\ \frac{\partial}{\partial y}(I + J) \end{bmatrix}.$$

Terms can be rearranged as follows:

$$\begin{aligned} \iint_W \left[J(\mathbf{x}) - I(\mathbf{x}) + \frac{1}{2} \mathrm{g}^T(\mathbf{x}) \mathbf{d} \right] \mathrm{g}(\mathbf{x})\, d\mathbf{x} &= 0 \\ \iint_W [J(\mathbf{x}) - I(\mathbf{x})] \mathrm{g}(\mathrm{x})\, d\mathbf{x} &= -\iint_W \frac{1}{2} \mathrm{g}^T(\mathbf{x}) \mathbf{d} \mathrm{g}(\mathbf{x})\, d\mathbf{x} \\ \iint_W [J(\mathbf{x}) - I(\mathbf{x})] \mathrm{g}(\mathrm{x})\, d\mathbf{x} &= -\frac{1}{2} \left[\iint_W \mathrm{g}(\mathbf{x}) \mathrm{g}^T(\mathbf{x})\, d\mathbf{x} \right] \mathbf{d}. \end{aligned}$$

Thus, we have simplified the expression to a 2×2 matrix equation,

$$\mathrm{Z}\mathbf{d} = \mathbf{e}, \tag{2.2}$$

where Z is a 2×2 matrix,

$$\mathrm{Z} = \iint_W \mathrm{g}(\mathbf{x})\mathrm{g}^T(\mathbf{x})\, d\mathbf{x}$$

and $\mathbf{e}$ is a 2×1 vector,

$$\mathbf{e} = 2 \iint_W [I(\mathbf{x}) - J(\mathbf{x})]\mathrm{g}(\mathrm{x})\, d\mathbf{x}.$$

Equation (2.2) allows us to solve for the approximate displacement of a feature, given its starting location and the two images. Furthermore, the computed displacement has sub-pixel accuracy. Since we are dealing with a discrete image composed of pixels, the above definitions for Z and $\mathbf{e}$ are computed with a summation over the window rather than an integral. The x and y image derivatives are approximated by convolving the images with a Sobel operator.

Since the above computation for displacement is only an approximation, it is useful to repeat the procedure for more than one iteration. If the displacement does not converge towards zero after several iterations, the feature is considered lost. For features displaced by a large amount, the approximation also breaks down because the Taylor series approximation becomes less accurate. To handle such a case, it is best to perform several iterations on versions of the images re-sampled to a coarser resolution, followed by several iterations on the full-resolution images.

A final consideration with the Kanade-Lucas-Tomasi algorithm is the choice of initial features. It is wasteful to track all pixels of the starting image to the destination image. A more useful approach is to track only those pixels which represent sharp, well-defined features. In fact, it has been shown that the best features to track are exactly those features which can be handled well by Equation (2.2) [12]. The eigenvalues of Z give us an indication

of how successful the tracking will be for a given feature. Large eigenvalues indicate a feature that is more well-defined than the image noise and can thus be tracked reliably. Thus, when choosing features to track, we sort the pixels in descending order of their minimum eigenvalue and pick the first N from the list, where N is the number of features we wish to track.

2.4 Motion model detection

Given a set of moving points, each with a position and velocity, our goal in motion model detection is to decide if the set of points is representative of a predefined model. In this project, we use the approach taken by Y. Song of storing the model as a probability density function, and finding a labeling for the data that maximizes the probability [4]. We will now summarize that approach.

We first define a set of body parts, each corresponding to a point feature that could be tracked. Let $\mathcal{S}_{\text{body}} = \{\text{LW}, \text{LE}, \text{LS}, \text{H}, \ldots, \text{RF}\}$ be the set of body parts, where LW is the left wrist, H is the head, RF is the right foot, etc. We will also refer to these as the set of possible labels. Since the model for the human is learned without supervision, the actual mapping of body parts to point features is undetermined and does not correspond exactly to individual limbs. Each body part also has a vector of observed measurements consisting a position and velocity, which we will denote as X_{LW}, X_{LE}, etc. The model of motion is stored as a probability density function P, which can be evaluated for a given set of observed data:

$$P_{\mathcal{S}_{\text{body}}}(X_{\text{LW}}, X_{\text{LE}}, X_{\text{LS}}, X_{\text{H}}, \ldots, X_{\text{RF}}). \tag{2.3}$$

Now, consider a vector of N observed points $\overline{X} = [X_1, X_2, \ldots, X_N]$ where the correspondence between points and labels is not known ahead of time. We want to find the permutation of points that maximizes the probability density function, Equation (2.3). Put another way, we want to find some labeling $\overline{L} = [L_1, L_2, \ldots, L_N]$ where $L_i \in \mathcal{S}_{\text{body}}$ is the label

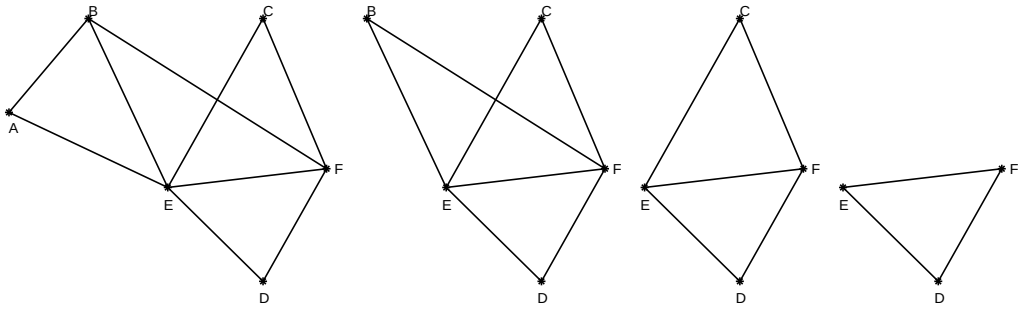


Figure 2.4: An example of a decomposable triangulated graph with elimination order A, B, C, D, E, F [4].

of point X_i, that maximizes the probability density function. We will define this optimal labeling as

$$\overline{L}^* = \arg\max_{\overline{L}\in\mathcal{L}} P(\overline{L}|\overline{X}) \tag{2.4}$$

where $P(\overline{L}|\overline{X})$ is the conditional probability of the observation $\overline{X}$ given the labeling $\overline{L}$, and $\mathcal{L}$ is the set of all permutations of the labeling.

Assuming N, the number of points, is equal to the number of body parts, a brute force search to solve Equation (2.4) would have runtime $\mathcal{O}(e^N)$. This poor efficiency is computationally prohibitive for a practical algorithm. The key breakthrough offered by the Song approach is to assume that certain parameters of the probability density function (2.3) are independent of each other. By determining such structure as the model is learned, the efficiency of the detector can be vastly improved using a dynamic programming algorithm. Before understanding how such an improvement is made, we must first understand how the model is represented.

As described in Section 2.1, the motion model is stored as a graph with each vertex representing a point feature, and each edge representing statistical dependence of one feature on another. A helpful assumption is to force this graph to have the special form of a decomposable triangulated graph[1]. Because of this structure, the graph will have one vertex

[1]A decomposable triangulated graph is a graph composed of triangles, such that there is always some single vertex that, when removed with its adjacent edges, the remaining edges and vertices constitute a decomposable triangulated graph. After a maximum number of decompositions, a single triangle will remain.

that is dependent on only two other vertices. When this vertex is eliminated, there will be another vertex that is dependent on only two others, and so on. The sequence of vertex removals that preserves this property is known as the *elimination order* of the graph. Figure 2.4 shows a simple example of such a graph and its elimination order. Thus, a probability density function stored in a decomposable triangulated graph can be approximated as a product of independent conditional density functions. For example, if the elimination order of the vertices of a graph are A, B, C, D, E, F, as they are in Figure 2.4, the probability density function can be represented as

$$P(A, B, C, D, E, F) = P(A|B, E)\, P(B|E, F)\, P(C|E, F)\, P(D, E, F).$$

Because the probability density function with many parameters can now be represented by a product of smaller conditional density functions, a dynamic programming algorithm is possible that can search the space of all possible labelings much more efficiently. Each conditional density function can be searched individually in $\mathcal{O}(N^3)$ to find the label that maximizes it for any possible pair of conditional labels. Thus, with M vertices in the model, the total runtime to maximize the probability density function is $\mathcal{O}(M\, N^3)$—a significant improvement over the $\mathcal{O}(e^N)$ brute force search. Specific details of the dynamic programming algorithm are presented in [4] along with how occlusion and clutter can be seamlessly handled by the same algorithm.

Now we know how to find the labeling $\overline{L}$ that maximizes the probability density function P. For a given $\overline{X}$, this procedure gives us two useful pieces of information: (1) the numerical value of the probability density function and (2) the optimal labeling of the point features. In order to complete the detection process, we can choose a threshold for the value of the probability density function. If the value is higher than the threshold, we consider the input points to match the model, and if the value is lower than the threshold, we say it does not match. The value of the threshold can be chosen empirically as a tradeoff between false

detections and missed detections, as will be discussed in Section 4.1.

2.5 Summary

We have explained our approach to each of the three stages of the human motion detection system:

1. Image segmentation achieved with a mixture of Gaussians approach to background subtraction.

2. Point feature tracking utilizing the Kanade-Lucas-Tomasi method.

3. Detection of a motion model by finding an optimal evaluation of a probability density function with the Song approach.

The next section will show how each of these stages was implemented in a real-time working system.

Chapter 3

Implementation

In the previous chapter, we discussed the choice of algorithms for each stage of our human motion detector. Although this discussion provides a good theoretical overview of how the detector works, it is not enough information to implement the system. In this chapter we discuss the practical details of how each algorithm is implemented and how they come together to form a complete system.

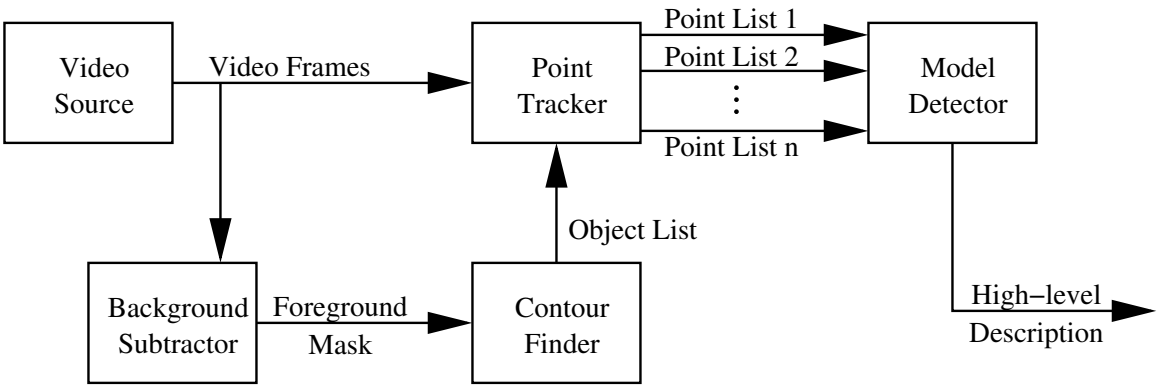


Figure 3.1: The software block diagram of the human motion detection system.

3.1 Software overview

The block diagram in Figure 3.1 gives a high-level overview of the software architecture of the human motion detector. Input to the system is provided by a video source which can be either a IEEE 1394 digital camera[1] or a sequence of still image files in JPEG format. The IEEE 1394 digital camera input allows for live video to be processed in real-time as it is captured by the camera. The still image input allows pre-recorded data to be processed. This pre-recorded data can be from any source, such as a Mini-DV camcorder or a video capture card, as long as it is first converted to JPEG format.

Video data from the video input is made available to the background subtraction algorithm, which is responsible for differentiating between foreground and background image regions. The implementation of the background subtracter is discussed in more detail in Section 3.2. The output of the background subtracter for each video frame is an 8-bit per pixel bitmap that serves as a *foreground mask*: Those pixels which are foreground have value 255 and those pixels which are background have value 0.

The foreground mask provided by the background subtracter is processed to build a data structure that enumerates the boundaries for each distinct foreground object. This enumeration is achieved by the *contour finder*, a simple algorithm that finds each "connected component" of the foreground mask. A connected component is defined as a region of an image whose pixels all have the same value and are adjacent to other pixels in the same connected component. For our contour finder, we used an implementation of the algorithm provided by the Intel Open Computer Vision Library [13]. Once these contours are located, those with small geometric area are ignored. Such small contours are likely to be noise or small image disturbances that are probably not human.

For each contour, a rectangular bounding box is computed which is used as the region

[1]IEEE 1394, also known as "FireWire" or "i.Link", is a standard for a high-speed serial bus. It supports a mode of transmission that guarantees fixed bandwidth, making it ideal for digital video cameras and any type of device that streams data at high speed. IEEE 1394 supports transfer rates up to 400 megabits per second.

of interest for the Kanade-Lucas-Tomasi point feature tracker, whose implementation is discussed in Section 3.3. The feature tracker outputs a list of point coordinates within this bounding box and a velocity for each. This list of point features is then input to the statistical model detector, which is discussed in Section 3.4.

The output of the model detector is a single number, the evaluation of the model's probability density function. Larger probabilities indicate that the set of input points match the model well, and smaller probabilities indicate a poor match. This probability can be thresholded to decide whether or not the object is human. The considerations in choosing this threshold are outlined in Chapter 4.

Finally, once each object is evaluated to be a human or not, the results are rendered on-screen, overlaid on top of the input video. A box is drawn around those objects which are detected to be human. Other visual indicators are also drawn to indicate the status of each stage of the algorithm. An example of the output of this rendering is shown in Figure 3.2.

3.2 Background subtraction

Our implementation of background subtraction follows the general algorithm described in Section 2.2. However, there are a number of approximations and simplifications that have been made to increase speed. Most importantly, the timing of the algorithm has been changed so that a new background model is only generated once every 240 frames, or 8 seconds, rather than recomputing the model for each frame. During the time between model generation, statistics for the background are collected from each frame and processed incrementally. Once 240 frames have been reached, a new background model is computed for use on the next 240 frames. Furthermore, since successive frames tend to be very similar, only every fourth frame is used for statistics collection. Another simplification which increases speed is to only perform background subtraction on a sub-sampled version of each image. Thus, a 640 pixel by 480 pixel image can be re-scaled to a resolution of 160 by 120, a quarter of the

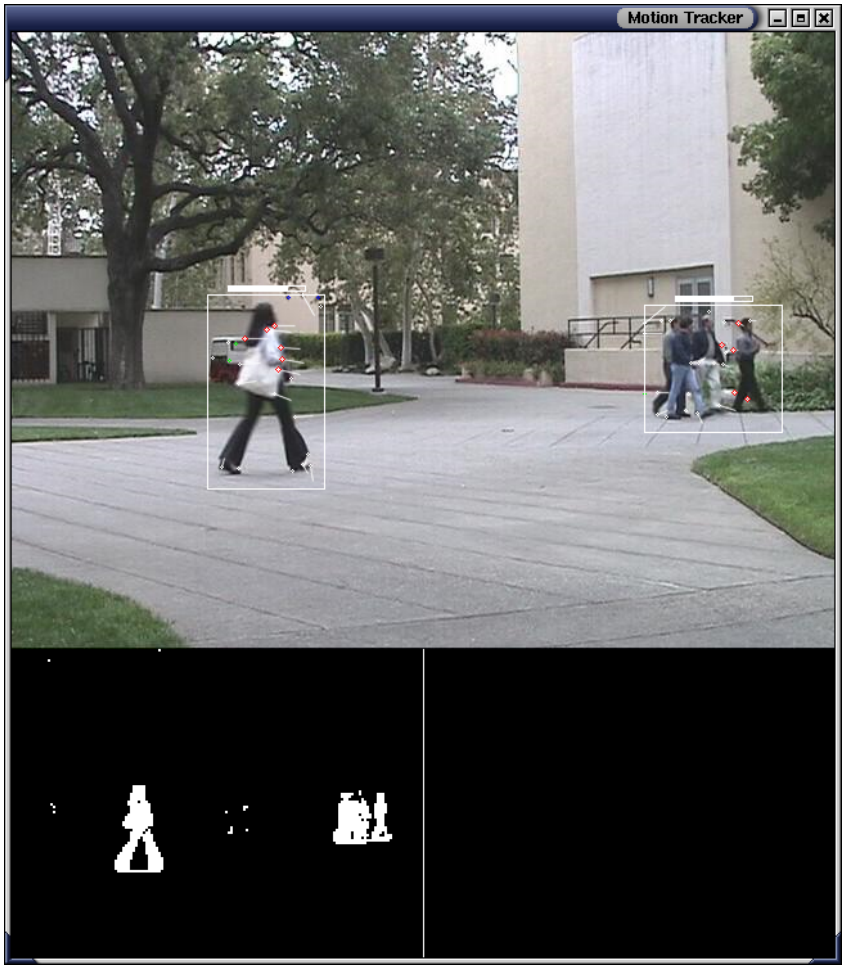


Figure 3.2: A screen capture of the motion detector's graphical user interface. A box is placed around any object that is detected to be human. A small "thermometer" bar above each box indicates the confidence of the model detector. A mostly-filled bar indicates high confidence that the object is human, while a mostly empty bar indicates low confidence.

linear dimensions. Since this image is one sixteenth the area of the original, the processing time for background subtraction is also one sixteenth of the original time.

We can express background subtraction in terms of three basic operations:

- *Background Update*, which is run once every 4 frames, gathers statistics for the background model.
- *Model Generation*, which is run once every 240 frames, computes the background image from the statistics which have been gathered during the Background Update phase.
- *Background Subtraction*, which generates a foreground mask for every frame. This step is simply performed by subtracting the background image from the current frame, taking the absolute value of the difference, and thresholding it with the value of three standard deviations of the average image noise.

As discussed in the approach, we maintain a set of Gaussians for each pixel of the background. Each Gaussian has a mean, variance, and weight. To simplify, the variance is assumed to be fixed, equal to the variance of the image noise. After each Model Generation phase, these Gaussians are reset to their uninitialized state so that they may regenerated from the Background Update phases that follow. In our algorithm, the weight of a Gaussian is simply equal to the number of frames for which the pixel has taken a value within three standard deviations of that Gaussian's mean. Futhermore, we keep track of the sum of these pixel values, so that the Gaussian's mean is simply this sum divided by the number of frames. Also, each pixel has exactly five Gaussians associated with it to simplify data structures.

The procedure to update each pixel during the Background Update phase is as follows:

- Compute the mean of each of the five Gaussians by dividing each sum by each frame count (weight).
- If the current pixel value is within three standard deviations of any of the five means, increase that Gaussian's weight by 1 and add the current value to its sum.

- Otherwise, replace the Gaussian of lowest weight by a Gaussian with weight equal to 1 and sum equal to the value of the current pixel.

This procedure will tend to collect a pixel's past values into the five highest weighted Gaussians that represent them. Although it is only an approximation of the exact mathematical specification in Section 2.2, it balances accuracy and efficiency. In the Model Generation phase, the mean of the highest weighted Gaussian for each pixel is chosen to be the background value.

This background subtraction algorithm is easily extended to color images by applying the procedure separately to each of the red, blue, and green channels. If any one of the three channels is determined to be foreground for a given pixel, the entire pixel is marked as foreground.

After background subtraction is complete, the morphological dilate operation is applied twice to the foreground mask. This operation has the effect of enlarging the area of each connected region in the mask and will close any small gaps. Such gaps are closed to maximize the likelihood that each moving object is contained within a single region, rather than two smaller ones. Once this step is complete, the foreground mask is run through a contour finder as described in Section 3.1.

3.3 Kanade-Lucas-Tomasi feature tracking

The Kanade-Lucas-Tomasi algorithm was implemented almost exactly as described in Section 2.3. In fact, the code was based on a reference implementation of the KLT algorithm written by Stan Birchfield [14], although heavily modified to be optimized for speed and the details of this particular application. In order to be robust against large displacements of features, the algorithm is first run on a sub-sampled version of the image. The sub-sampled image is computed by first feeding the original through a Gaussian filter and then removing the odd-numbered rows and columns. Tracking is first performed on an image sub-sampled

twice to get an approximate displacement.

The tracking formula, Equation (2.2), is used to compute the displacement for each iteration of the algorithm. The origin of the starting image is then shifted by this displacement so that the tracking equation can be reapplied. Once the displacement converges near zero, tracking is complete. If it does not converge in a few iterations, the algorithm fails.

The KLT algorithm has several numerical parameters that were chosen for this implementation. The window size was selected to be 7 pixels by 7 pixels. This parameter is the size of the region over which the summations in Equation (2.2) are evaluated. Tracking iterations are carried out until a single iteration has a displacement less than 0.1 pixels. If 10 iterations are completed without a displacement less than 0.1 pixels, the feature is discarded. After a feature is tracked, its residue, $\iint_W |J(\mathbf{x}) - I(\mathbf{x} - \mathbf{d})|\, d\mathbf{x}$, is computed to determine if the image patch roughly matches the original feature. If the result divided by the area is greater than 20.0, the feature is discarded.[2]

The x and y gradients of the images are computed by convolving them with the Sobel 3×3 operators:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Since the computed displacement is often a fractional value, the image origin has to be shifted by a non-integer value. In this case, bilinear interpolation is used to compute the approximate value between the pixels so that the sum, $I(\mathbf{x}) + J(\mathbf{x})$, and difference, $I(\mathbf{x}) - J(\mathbf{x})$, can be computed with as much accuracy as possible.

Throughout the implementation of this algorithm, code in the Intel Performance Primitives Image Processing Library was used. The IPP IPL provides assembly-optimized versions of standard image processing primitives for the x86 processor architecture. In many cases, these functions provide a four-fold or more improvement in efficiency. One function that the

[2]The value of each pixel ranges from 0 to 255.

IPP IPL provides computes the value of the minimum eigenvalue for a set of pixels. This function gives us a high-speed method of finding which pixels to use for initial features in the KLT algorithm. Of these initial features, the N pixels with the largest minimum eigenvalues are chosen for tracking. This implementation was tested with N set to 15, 20, 25, or 30. Results for each of these values are given in Chapter 4.

3.4 Human motion detection

Much of the code used for the model evaluator for human motion detection was borrowed from the implementation of Y. Song [4]. A few speed increases were achieved by use of the Intel Performance Primitives. Since our implementation does not include the ability to train a new motion model, it imports this model from a Matlab workspace. The workspace can be saved directly from the Matlab session used the train the model, and our code will read this workspace at runtime and use the enclosed model for human detection. The structure and contents of this model were described in Section 2.4. The results presented in Section 4 use a specific model which was generated by Y. Song using unsupervised learning on a different video sequence. A graphical representation of this model is shown in Figure 3.3.

The format of the model is such that it is invariant to translation but not scaling. For example, the model can detect two humans equally well regardless of the absolute value of the x and y coordinates. However, the spacing of the point features matter a great deal. The model will only match human figures that are about the same size in pixels as it was trained on. Since this causes a problem for general purpose detection, all x and y values are scaled before being input into the model detector. Since the output of the contour finder tells us the bounding box for the foreground object, we can use this height as the approximate height of the human figure. The coordinates can then be scaled to match the model.

A final consideration is that a single model can only detect a person walking from right-to-left or left-to-right, but not both. This restriction exists because the model contains a

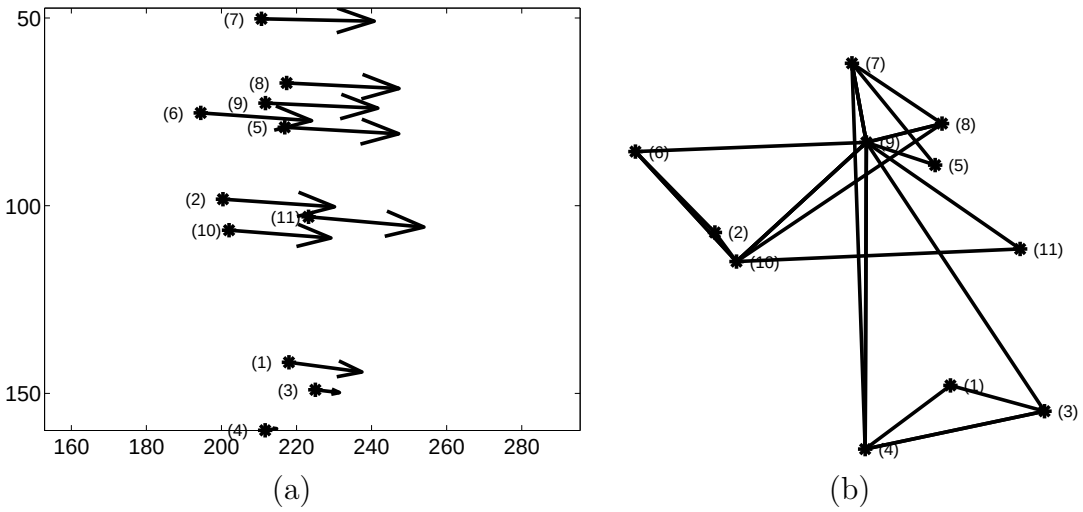


Figure 3.3: A graphical depiction of the learned motion model used for data collection in this project [4]. (a) The mean position and mean velocity for each point in the model. (b) The structure of the decomposable triangulated graph for the model.

mean value for the x velocity of each point feature. If this mean is positive the model matches left-to-right walkers, and if negative it matches right-to-left walkers. To overcome this limitation, we take the absolute value of the x velocity of point features before feeding them to the detector. This allows a model with positive mean x velocity to match both right and left walkers, making the assumption that the walks are symmetric.

Chapter 4

Results

In this chapter, we present two video sequences, each approximately 100 seconds long, which were used to analyze the performance and accuracy of the human motion detection system. For each video sequence, the "correct" location of human motion is known ahead of time. This knowledge allows us to measure the accuracy rate of the system and present tangible evidence in support of the algorithm.

4.1 Method of Performance Analysis

In order to objectively measure accuracy, we developed a procedure to annotate the *ground truth* of a video sequence. Ground truth refers to the actual presence of human motion as a human viewer would interpret it. Once this ground truth is known for a sequence, the performance of our system in detecting human motion can be evaluated. In our interface, ground truth is annotated by running the detector on a pre-recorded video sequence and manually labeling each frame. The background subtracter will find connected regions of each frame that show motion. Each of these regions is manually annotated as either human or non-human. In order to streamline the process, the operator simply corrects any mistakes the detector makes rather than labeling each object individually. If the system fails to detect

a human, the operator clicks the mouse on that region to indicate the mistake. Likewise, if the system falsely detects a human where there is none, the operator clicks at that location as well. Frames that contain ambiguous objects can be specially marked so that they are not included in the accuracy statistics. For example, if a human is only half-visible at the edge of a frame, the decision between human and non-human would not be meaningful. Once this process is complete for a sequence, the software will output a file that lists the ground truth for each frame. This file can then be used by the software to evaluate its own accuracy.

A useful tool for the visualization of these results is the Receiver Operating Characteristics (ROC) curve. Since the system produces a numerical probability for each moving object in the scene, there is no clear cut boundary between what should be interpreted as human and what should be interpreted as non-human. In order to make such a distinction, a cut-off threshold must be chosen for the probability values. A ROC curve shows the rate of correct detection versus the rate of false detection for any choice of threshold. The definitions for each rate are as follows:

Rate of correct detection, P_D**,** the number of correctly identified humans divided by the total number of true humans in a video sequence for a given threshold.

Rate of false detection, P_{FA}**,** the number of non-human objects identified as human divided by the total number of true non-humans in a video sequence for a given threshold.

Examples of a missed detection and a false detection are shown in Figure 4.1.

If the threshold is too high, no objects will be detected, giving a rate of 0.0 for both. If the threshold is too low, all objects will be interpreted as human, even non-human objects. This case corresponds to a rate of 1.0, or 100%, for both. Thus, the receiver line on any ROC curve will extend from $(0.0, 0.0)$ to $(1.0, 1.0)$. A "good" curve is one that exhibits a large P_D and a small P_{FA} at some point along the curve. This point is considered to be the optimal choice of threshold, and is usually defined as the point where $P_D = 1 - P_{FA}$.

(a) (b)

Figure 4.1: (a) An example of a missed detection: The man on the right has not been detected by the algorithm. Such a case negatively impacts P_D. (b) An example of a false detection: The lawnmower has been incorrectly labeled as a walking human. Such a case negatively impacts P_{FA}.

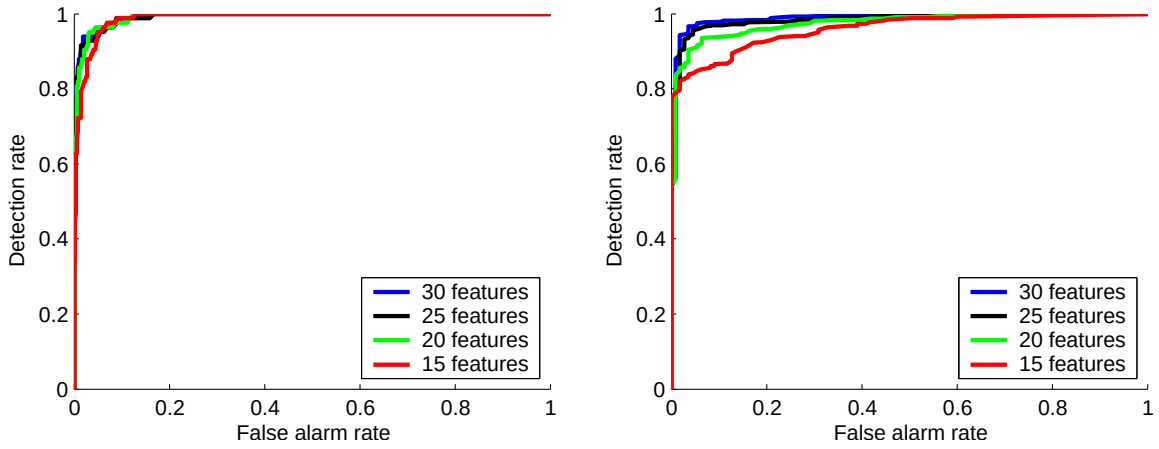
4.2 Results of Performance Analysis

The two scenes used for performance measurement are shown in Figure 4.2. The first scene was shot outside the Caltech bookstore using a IEEE 1394 webcam operating with a resolution of 640 by 480 pixels and a rate of 30 frames per second. The second scene was shot outside the Jorgenson building using a Sony Mini-DV camcorder with a resolution of 720 by 480 pixels and a rate of 29.97 frames per second. Each video sequence is approximately 100 seconds long.

Figure 4.3 shows the ROC curves for these two scenes, demonstrating the relatively high accuracy of the system. For the bookstore sequence with 25 features per object, at the point where $P_D = 1 - P_{FA}$, P_D is 95.2%. For the Jorgenson sequence, P_D is 95.5% where $P_D = 1 - P_{FA}$. Additional statistics are shown in Table 4.1. There are some differences between the sequences that should be noted. The bookstore sequence contains a very large number of objects that are not human—usually moving shadows or trees that the detector does not easily confuse for humans. Thus, the bookstore sequence shows a very low false alarm rate. In contrast, the Jorgenson sequence contains a large number of human objects,

(a) (b)

Figure 4.2: The two scenes used for performance measurement of the human motion detector. The first scene (a) was captured outside the Caltech bookstore and the second scene (b) was captured near the Jorgenson building.



(a) (b)

Figure 4.3: Receiver Operating Characteristic (ROC) curves for the (a) bookstore and (b) Jorgenson video sequences. Each curve was collected with varying numbers of features tracked per object, ranging from 15 to 30 points. This number of points was input to the model detector. More points increases the accuracy of the detector by giving it more input to work with, but also increases computational cost.

Bookstore sequence (83 humans, 473 non-humans)			**Jorgenson sequence** (528 humans, 110 non-humans)		
	$P_D(\%)$	$P_{FA}(\%)$		$P_D(\%)$	$P_{FA}(\%)$
15 features	95.0	5.0	15 features	87.5	12.5
20 features	95.9	4.1	20 features	93.6	6.4
25 features	95.2	4.8	25 features	95.5	4.5
30 features	95.0	5.0	30 features	96.4	3.6

Table 4.1: Detection rates and false alarm rates for each video sequence given varying numbers of features input to the model detector. These rates are found on the ROC curves in Figure 4.3 at the point where $P_D = 1 - P_{FA}$.

but relatively few non-human objects. Thus, the false alarm rate in this sequence is much more sensitive to the threshold since it is shaped by only a few data points.

4.3 Computational Cost

The algorithm presented in this thesis was designed to be efficient enough for real-time applications. For video running at 30 frames per second with a resolution of 640 pixel by 480 pixel, real-time performance would require each frame to be processed in 33.3 milliseconds or less. Although our results show that this algorithm is slightly slower than that goal, additional optimizations or lowering the video quality would provide fully real-time performance.

The results for runtime speed of each stage are shown in Table 4.2. In gathering the timing data, the software was run on a single-processor machine equipped with a 2.0 GHz Pentium 4 Processor. These results demonstrate that for a frame with one object and using 25 features for the model detector, the total processing time is 108 ms. If only 15 features are used for the model detector, the total is reduced to 38 ms. Although both figures are larger than the goal of 33.3 ms or less, the algorithm is running fast enough to allow practical applications of the software. Futhermore, by parallelizing the algorithm to run on a multi-processor machine, real-time performance could be achieved.

1.	Background subtraction	1.0 ms per frame
2.	Feature tracking	
	Computing gradients	12.5 ms per frame
	Finding features	1.8 ms per object
	Tracking features	2.5 ms per object
3.	Model detection	
	given 25 features	90.0 ms per object
	given 20 features	45.0 ms per object
	given 15 features	20.0 ms per object

Table 4.2: Runtime for each stage of the human motion tracking algorithm on a 2.0 GHz Pentium 4 Processor. "Per frame" operations are executed once for each frame of video. "Per object" operations are executed once for each connected region that the background subtracter finds per frame.

Chapter 5

Conclusions

As shown in Chapter 4, the system presented herein is successful as a robust detector of human motion. However, there is still much progress to be made, especially to handle shortcomings that are not exercised by the two video sequences used for performance measurement. One such shortcoming is that the model of human motion contains a *static* representation of what a human figure should look like. Although this static representation was built using statistics over a certain period of time, the model is a time-average of these statistics and inherently cannot contain information about the periodic nature of the human gait. Thus, any moving object that has roughly the shape of a human and moves with the speed expected of a human will be detected as a human. The false detection shown in Figure 4.1(b) is an example of a detection that could be avoided with a model that takes into account the periodicity of the human gait. In the captured scenes for this thesis, this limitation did not cause large numbers of false detections, but it would be a significant problem in more complicated environments.

A second limitation is that of the background subtracter. We make the fundamental assumption that the background will dominate most scenes and will be stationary. This is unacceptable for more crowded environments where surveillance is desired. In addition, the requirement for a stationary background rules out situations where the camera is in motion,

such as on vehicles or robotic cameras. Futhermore, any foreground objects that overlap or are close to each other will be treated as a single object. This undesired grouping is the nature of the connected component algorithm currently used by the contour finder. In order to successfully handle these cases, a better algorithm for image segmentation is needed. One that relies on optical flow rather background subtraction would be more robust in these situations. However, such an algorithm would be far more expensive in terms of runtime performance.

After analyzing the detections missed by the algorithm, such as the one shown in Figure 4.1(a), it appears that limitations in the feature tracker are responsible. Nearly all missed detections were caused by a human figure that did not have enough texture for accurate feature tracking. For example, if a human wearing dark clothes walks in front of a dark wall, there are few features for the KLT tracker to use. In this case, the model detector will have inadequate information to make a decision, and will most likely reject the object as a human. Eliminating this shortcoming would require an approach to feature tracking that does not fail on minimally textured surfaces.

Aside from these limitations, the system presented in this thesis serves as a successful proof of concept for a robust human motion detector in the field. In addition, it serves the validate the theoretical work of Y. Song in developing a useful model of human motion and an efficient detection algorithm.

5.1 Future Work

There are a variety of enhancements that could be made to this system to achieve greater detection accuracy and increased robustness:

- Objects could be tracked between frames rather than simply performing human motion detection on single frames. For example, a Kalman filter could be used to predict the future position and human likelihood of a given object. Such a filter would smooth out

particular frames in which detection fails, and would eliminate many false detections. The net effect would be an improvement in both P_D and P_{FA} along with a smooth tracking capability useful for higher-level applications.

- As described above, the current model of motion does not take into account the time-dependent nature of a walking human. Much greater accuracy would be possible with a detector and model that takes advantage of this periodicity in time.

- The current background subtraction algorithm can be confused by fast lighting changes or moving shadows. A better algorithm would use a technique based on optical flow for the image segmentation. This approach would also allow the camera to be in motion relative to the background.

- Modeling different types of human motion should be explored, such as walking seen from different viewpoints. The current system fails to detect humans walking directly towards or away from the camera. In addition, other forms of motion such as running should be modeled so that the detector can reliably detect and classify these cases.

Bibliography

[1] G. Johansson, "Visual perception of biological motion and a model for its analysis," *Perception and Psychophysics*, vol. 14, pp. 201–211, 1973.

[2] J. E. Cutting and L. T. Kozlowski, "Recognizing friends by their walk: Gait perception without familiarity cues," *Bulletin Psychonometric Society*, vol. 9, pp. 353–356, 1977.

[3] G. Mather and L. Murdoch, "Gender discrimination in biological motion displays based on dynamic cues," *Proc. R. Soc. Lond. B*, vol. 259, pp. 273–279, 1994.

[4] Y. Song, *A perceptual approach to human motion detection and labeling*. PhD thesis, California Institute of Technology, 2003.

[5] N. Howe, M. Leventon, and W. Freeman, "Bayesian reconstruction of 3D human motion from single-camera video," Tech. Rep. TR-99-37, Mitsubishi Electric Research Lab, 1999.

[6] L. Goncalves, E. D. Bernardo, E. Ursella, and P. Perona, "Monocular tracking of the human arm in 3D," in *Proc. 5th Int. Conf. Computer Vision*, (Cambridge, Mass), pp. 764–770, 1995.

[7] S. Wachter and H.-H. Nagel, "Tracking persons in monocular image sequences," *Computer Vision and Image Understanding*, vol. 74, pp. 174–192, 1999.

[8] D. Gavrila, "The visual analysis of human movement: A survey," *Computer Vision and Image Understanding*, vol. 73, pp. 82–98, 1999.

[9] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, pp. 245–252, 1999.

[10] C. Tomasi and T. Kanade, "Detection and tracking of point features," Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, 1991.

[11] S. Birchfield, "Derivation of Kanade-Lucas-Tomasi tracking equation." http://robotics.stanford.edu/~birch/klt/derivation.ps, 1997.

[12] J. Shi and C. Tomasi, "Good features to track," *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, pp. 593–600, 1994.

[13] "Intel open source computer vision library." URL: http://www.intel.com/research/mrl/research/opencv/.

[14] S. Birchfield, "KLT: An implementation of the Kanade-Lucas-Tomasi feature tracker." URL: http://robotics.stanford.edu/~birch/klt/, 1998.