Tracking Multiple Objects in Real Time

by

Robert Andrews

Department of Electrical and Computer Systems Engineering,

University of Queensland.

Submitted for the degree of

Bachelor of Engineering Honours

in the division of 15

October 1999.

Lot 1 Old North Road

Bray Park, Q. 4500

Tel. (07) 3205 4761

October 15, 1999

The Dean

School of Engineering

University of Queensland

St Lucia, Q 4072

Dear Professor Simmons,

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Computer Systems Engineering, I present the following thesis entitled "Tracking Multiple Objects in Real Time". This work was performed under the supervision of Dr Brian Lovell.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

Robert Andrews

Acknowledgements

I would like to thank Dr Brian Lovell for proposing my topic and supervising my development, also for his constant guidance and support throughout the year.

I would also like to acknowledge my peers for their motivation and aid during the design, debugging and implementation stages of this system.

Abstract

This thesis documents the design and implementation of a real-time object tracking system capable of operating on any modern generic desktop PC including a Video-for-Windows image capture device. A secondary achievement of this thesis is a point tracking algorithm which works seamlessly (though not in real time) with the rest of the system. Multiple points within an object can be tracked.

Before presenting details of the implementation, a general introduction and a theory section relative to this implementation is given. In particular, the concepts of the Hausdorff distance and RGB histogram matching are explained.

Details and results of the system are presented, illustrating the preprocessing, object isolation, object tracking and histogram matching algorithms. An indication of the accuracy achieved is given by the two case studies included.

Speeds of up to 2.5 frames per second were achieved with adequate object tracking accuracy. The area matching method documented herein are shown to be too slow to run in real-time.

It is concluded that the multiple object tracking system described performs robustly and efficiently with an adequate accuracy. The feasibility of implementing real-time image processing applications with generic hardware is asserted.

Contents

List of Figures

Chapter 1

Introduction

Human perception has intrigued scientists for centuries. How do we see? How do we recognize visually similar objects? How do we follow objects innately with our eyes? These are all questions that interest researchers in the field of computer vision. With computers that can analyze their environment with a good deal of confidence, we can achieve more efficient automation in a myriad of tasks; applications in the broad genres of security, medical, military, mass-production and robotics, to name a few.

With future promise of increased processing power (due to technological innovations in the computer industry; the inevitable decrease in size and increase in speed of microchips) we will find more and more industries asking 'How can computer vision make us more productive?'. Implementation of real-time image processing applications has long been the realm of specialized or tailor-made hardware interfacing with a typically mainframe computer, such as an SGI Workstation. The aforementioned increase in processing power has now provided a new candidate for these roles, the humble desktop PC.

The aim of this thesis is to implement and analyze an object tracking algorithm with generic hardware in real-time, illustrating the immediate possibility of using off-the-shelf components to implement a real-time Multiple Object Tracking algorithm. In addition to a real-time object tracking algorithm, the system developed is capable of tracking points within detected objects, though not at a speed which is feasible for real-time processing.

There are three general steps in the real-time analysis of scenes[1]. The first is to acquire the frame to be processed by means of an image or video capture

device connected to the PC. The next is to preprocess the frame, removing any noise and to prepare the data for the subsequent analytical algorithms. In the third stage the data is analyzed by applying appropriate analytical algorithms, information about the computer's environment is extracted and the results are displayed in an informative manner.

Many matching algorithms have been proposed to solve the dilemma of object recognition including relative projection histograms, active contour models (deformable and non-deformable), texture analysis and Hausdorff distances[2][4][7]. My approach to the problem of tracking multiple objects in real time is to apply Hausdorff methodologies to edge sets obtained to successfully track objects through a scene. This has been previously achieved[4][8], this thesis' intention is to show that this method is viable using generic PC hardware. Taking into account the deformation of the body we are attempting to follow, I can apply a shape-matching method to track the movement of a human (or other deformable or rigid body) through the field of vision of a fixed camera.

Tracking the outline of objects as they move through scenes is the first criterion of my thesis, for the second part I use the outline of the region of the object being tracked (from the tracking section of my system) to constrain the search space of a histogram block matching algorithm looking for specified points within the object. I am able to track an object and points therein.

For example: I have a frame-sequence of a person walking forwards (profile view). I select the person as the object to track, and his elbow, shoulder, hips and knee as the points to track. The resultant data, (i.e. position and size of the object, position and scales of points to track) is collated and could be used for movement analysis or be rendered to give us three cylinders moving synonymously with the image-sequence.

A general breakdown of the analysis of each frame in a sequence, with a view to tracking multiple points in an object, is given below (Figure 1.0).

Using Matlab as my primary development environment algorithms have been tested for accuracy and computational cost. After the algorithms and data structures were tailored to produce an acceptable degree of accuracy in segmenting the image and tracking the objects (and points) the code can be translated into C or C++. This is possible to do automatically, using optimized

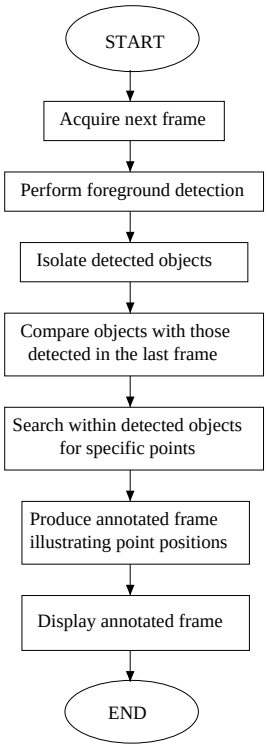


Figure 1.1: Flowchart of frame by frame processing

compilers for Matlab. An aesthetic, user-friendly interface and support for Video for Matlab devices has been added. After optimization of code to take into account the Intel Pentium series' instruction set, an increase in speed of the algorithms to above 1000% of their relative speed in Matlab is anticipated.

Many libraries of functions relating to image processing are available; If I'm employing any generic functions for which optimized code is available I can readily use this to reduce the CPU time of those sections of my implementation. Intel have released an Image Processing Library optimized for the Pentium family chipset.

The majority of real-time systems previously implemented have utilized such technologies as Digital Signal Processing boards and expensive high-quality cameras; it is my intention to show that for many image processing tasks it is possible to use off-the-shelf hardware on a standard desktop PC. My tracking system is written to support any Video-For-Windows device, running at an acceptable frame rate for real-time processing on a modern desktop computer. This expands the scope of implementation; as my software is not tailored to a specific hardware environment it illustrates the possibility of real-time

processing using any standard modern personal computer. The Matlab code supplied in Appendix A is easily ported to the Linux or Unix environment.

Object tracking systems have many obvious and many more obscure applications. In the field of physiotherapy, for example, it may be deemed fortuitous to use an object and point tracking system to analyze the movement of a patient's limbs through the range of an exercise. Security applications are virtually endless. It is possible to mark objects that have been left in the scene, any left for more than a certain time could call for authoritative attention automatically. If an object tracking system was implemented with a face recognition system it would be possible to detect known felons in public places (Scotland Yard is presently trialing such a system). Satellite images can be processed to reveal, say, the movement of large ocean-bound vessels, possibly automatically monitoring our largely undefended coastline automatically.

My thesis shows that these types of applications can be implemented on a standard desktop PC. The case studies I have included are all of image-sequences taken in a office-type room monitoring environment. The principal of this implementation extends to all fixed-background image sequences. The scope of the examples shown is chosen to succinctly illustrate the operation and results of my thesis. It can be directly implemented with few modifications to track any objects and points therein. The only constraints upon the system are the necessity of a fixed background sequence and, if point tracking is required, that the image be presented with three color planes (red, green and blue).

Chapter 2

Literature Review

2.1 Introduction to Real-Time Image Processing

This is a very basic book, introducing the topic of Real-Time Image Processing and illustrating some simple examples. Some of the included algorithms were insightful to guiding further study, and some parts of the optimization section (tips for optimizing software for real-time functionality) were interesting. As an initial stepping stone into the broad area of Real-Time Image Processing this tome was a useful resource. As an introduction to the topic should, it certainly whet my appetite, so to speak. It gave me motivation and direction for pursuit of information in this field with an increased general knowledge.

With pseudo-coded examples of optimization this reference was more relevant to me as I progressed into later stages of my implementation. When I was working to reduce the computational cost of my algorithms I applied some of the methods mentioned in this book.

As the pseudo-code dealt generally with a lower-level language than Matlab (i.e. C/C++ or Fortran), I mainly used the optimization procedures for C code I wrote which interfaces with Matlab. The optimizations I performed in this section of my thesis transformed it from a system which struggled to meet real-time frame rate criteria into a system which performed at an acceptable speed.

2.2 Variational Methods in Image Segmentation

The concept of image segmentation (or "edge detection") is as follows:

First, remove some of the noise in the frame. Then find the gradient at each point. (Abstractly, a picture may be seen as a snapshot of a vector field, the colors the intensities of the field at each point). Therefore, where the gradient is highest (when the intensity difference between a pixel and its neighbors is above a threshold), we have an set of points representing an edge. Next, smooth these edges and join any that look like they should be continuous.

Depending on the threshold value, spurious edges may be introduced, or too few edges may be present. We want to find the middle ground, the best fit giving us the edges we want without superfluous noise (if possible).

As the title of this tome implies, it is an analysis of algorithms, tracking their development and relating them all back to the Mumford-Shah algorithm. An algorithm for the energy of each segmentation is given, a qualitative representation of the correctness. A surprising conjecture is given and proven to be true, all edge-detection (segmentation) algorithms have basically the same form. Some choose to minimize different terms and add certain features in their implementation but this reference show their basis can be traced to the Mumford-Shah energy. This energy is given below (Equation 2.1):

$$E(u,K) = \int_{\Omega/K} (|\bigtriangledown u(x)|^2 + (u-g)^2)dx + \text{length}(K). \tag{2.1}$$

Many edge detecting algorithms are discussed at length and related back to this analytical basis. Concepts are discussed in great detail with a heavy reliance on variational formulae (i.e. differential equations).

This book details multiscale edge-detection algorithms which are worth mentioning if only for interests sake. The notion of a scale in this respect is best described as follows. We can consider each edge of an image to have a weighting. This weighting is directly proportional to the clarity of the edge (high contrast = obvious edge = large weighting). If we have an image A_0 we can obtain a series of images A_λ which are rougher versions of A_0 as λ increases. With multiscale analysis techniques we can find major edges at low scales (large λ's) and follow these back through higher scales. The justification for

this is a final segmentation which (hopefully) contains all the major edges and few spurious noisy edges.

Perona and Malik formulated an improvement and generalization of the edge detection algorithm as follows. Using conditional filtering in the initial noise-reduction step of segmentation, selectively smooth the image depending on $\bigtriangledown u(x)$ at each point. $\bigtriangledown u(x)$ is the gradient of the image around point x. Hence all edges are kept, noise is removed and the rest of the image is smoothed.

Elements of geometric measure theory are given, perhaps the most applicable of which is the Hausdorff measure and its properties. This is explained in my review of the next article (Section 2.3) in a more relevant light, but this tome has some interesting in-depth proofs of Hausdorff measures and lengths relating mainly to image segmentation.

As Cauchy and other edge-detection algorithms are supported in Matlab (they are pre-defined functions) my main motivation for inspecting this reference was for a more in-depth understanding of the topic of 'edge-detection' or segmentation. It is possible to use theses algorithms without the understanding of how they perform the translation from image to edge sets, but my quest for knowledge drew me into this book as it is a defining reference in the field.

2.3 Hausdorff Distance Image Comparison

This short resource, a two page web site, gives a concise, succinct and easily understood explanation of using Hausdorff Distances to assay the degree of similarity between two objects. This has immediately obvious applications relevant to my thesis.

We read that we can use the Hausdorff distance to identify instances of a "model" bitmap A (our object to be tracked) in a bitmap B (each frame of our sequence). The distance will identify multiple instances of the model. This is a very robust matching algorithm, it responds well to transformations and missing parts (perfect for the application of human movement, many parts of the object being tracked are occasionally obscured).

The Hausdorff fraction is introduced which measures the percentage of $A + t$ (the transformed "model") which lies within a certain distance of (within δ of) points of B (our frame).

To make the Hausdorff distance more robust to "outliers", single points which lie far away from the rest of the points of interest, this article mentions a *generalized* Hausdorff distance. This works by instead of taking the furtherest point we take the kth furtherest point. Instead of the largest distance we take, for example, the fifth largest, effectively filtering out the (possibly erroneous) four furtherest points of the object.

An important property of the generalized Hausdorff measure is that it separately accounts for (simple) transformations (by the distance δ) and for outlying points (by ranking them with k).

In direct relation to my thesis, I use the Hausdorff distance to recognize objects that were detected in the previous scene.

2.4 Tracking Human Walking in Dynamic Scenes

This article proposes modeling the human body as an articulated object connected by joints and rigid parts, describing human walking as periodic motion. A recognition scheme that determines the posture of the walker is given. Kinematic constraints due to the limited range of movement available to humans are used to facilitate tracking of the various body parts.

This article is directly related to my thesis in both the problems it faces and the results it proposes to achieved. One major difference however is the constraint on the sequence being translated. This method requires that the subject is moving parallel to the camera, in a fluid motion. My implementation is intended to work with nearly all camera angles.

Their implementation is broken into four parts; pre-processing, modeling, recognition and tracking. The pre-processing stage embodies detection and positioning of the subject. The modeling stage describes the body and its walking (imposing kinematic constraints and estimating the period and phase of the simple harmonic motion of their gait). Posture is determined in the recognition phase. The tracking phase tracks the subject by referencing recognized body parts.

Using an affine model (small uniform change) to detect background movement they isolate the walker. This is achieved by isolating the motion of the background. Any remaining movement is attributed to the walker. Every second

frame the region corresponding to the walker is isolated. This segmentation also allows the height and position of the walker to be extracted.

Models of the human body and of the dynamics of walking provide them with criteria to evaluate each pose in a walkers stride. Using 11 angles, θ_i, ($i = 1, 2, \ldots 11$) they created a set $\Theta_M(p) \stackrel{df}{=} [\theta_{M1}(p) \ldots \theta_{M11}(p)]^T$ where $p \in [0, 1)$, containing angle combinations for orientations of the human body during walking (taken as the sequence of angles of a model walker). To be able to extend this into three dimensions I would have to use a two dimensional array to appropriately represent the angles inherent in 3D motion. (Even though I am using a two dimensional camera I will attempt to estimate three dimensional joint tracking with the aid of kinematic restraints).

The closest fit of the posture of the walker in each frame is then inspected to find its closest match in the set $\Theta_M(p)$, giving them the effective pose p.

To find these angles we want to find the orientations and positions of all five parts of the body; four limbs and a head and torso region, each with two or three rigid sections. The derivation of an algorithm to track an object with two segments is given in the article.

If the angles at each joint are θ_1 and θ_2, it is feasible to propose

$$\begin{matrix}\theta_1 \\ \theta_2\end{matrix} = -\tau \begin{bmatrix} \sum I_\theta^2 & \sum I_\theta (I_{2\theta_2} - dI_{2y}) \\ \sum I_\theta (I_{2\theta_2} - dI_{2y}) & \sum (I_{2\theta_2} - dI_{2y})^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum I_\theta I_t \\ \sum (I_{2\theta_2} - dI_{2y}) I_t \end{bmatrix} \tag{2.2}$$

where the terms that collectively comprise the matrices are found from the Taylor series expansion of the region of interest at time τ.

As this derivation is a candidate for inclusion into my thesis I shall give a more in depth description of the process.

$I(\mathbf{x}, t)$ is the articulated object at time t, consisting of two rigid section $I_1(\mathbf{x}, t)$ and $I_2(\mathbf{x}, t)$. $J_1(t)$ and $J_2(t)$ are the articulations, the distance between these joints is d. The assumption that the object only has two degrees of freedom is imposed. $I(\mathbf{x}, 0)$ is the reference image and $I(f(\mathbf{x}, cbq), \tau)$ is the image at time τ where $f(\mathbf{x}, \mathbf{q})$ is the motion of pixel $\mathbf{x}$, parametrized by $\mathbf{q} = [\theta_1, \theta_2]^T$. From this,

$$f_1(\mathbf{x}, \mathbf{q}) = \mathbf{R}[\theta_1]\mathbf{x} \tag{2.3}$$

$$f_2(\mathbf{x}, \mathbf{q}) = \mathbf{R}[\theta_1] \begin{bmatrix} d \\ 0 \end{bmatrix} + \mathbf{R}[\theta_1 + \theta_2] \left(\mathbf{x} - \begin{bmatrix} d \\ 0 \end{bmatrix} \right) \tag{2.4}$$

Where $f_1(\mathbf{x}, \mathbf{q})$ is the motion of the pixel in the upper segment and $\mathbf{R}[\theta]$ is a 2 x 2 rotation matrix.

They then propose estimating the motion vector $\mathbf{q}$ by minimizing the following 'cost' function.

$$C(\mathbf{q}) = \sum_{\mathbf{x} \in \Re} (I(f(\mathbf{x}, \mathbf{q}), \tau) - I(\mathbf{x}, 0))^2 \tag{2.5}$$

$I(f(\mathbf{x}, \mathbf{q}), \tau)$ is expanded in a Taylor series:

$$I(f(\mathbf{x}, \mathbf{q}), \tau) = I(\mathbf{x}, 0) + \theta_1 I_\theta + \theta_2 I_{2\theta_2} - d\theta_2 I_{2y} + \tau I_t + \text{h.o.t} \tag{2.6}$$

where h.o.t denotes higher order terms. The following variables are introduced; $I_\theta \stackrel{df}{=} I_{1\theta_1} + I_{2\theta_2}$, $I_{i\theta_i} \stackrel{df}{=} -yI_{ix} + xI_{iy}$, $I_{iz} \stackrel{df\}}{=} \frac{\partial I_i(\mathbf{x},0)}{\partial \mathbf{x}}$, $I_{iy} \stackrel{df}{=} \frac{\partial I_i(\mathbf{x},0)}{\partial \mathbf{y}}$ and $I_t \stackrel{df}{=} \frac{\partial I(\mathbf{x},0)}{\partial t}$. The cost equation then becomes (after discarding the higher order terms)

$$C(\mathbf{q}) \cong \sum_{\mathbf{x} \in \Re} (\theta_1 I_\theta + \theta_2 (I_{2\theta_2} - dI_{2y}) + \tau I_t)^2 \tag{2.7}$$

After differentiation with respect to $\mathbf{q}$ we arrive at the solution given in equation 2.4.1.

The results given in the article detailing this implementation endorse it as an accurate tracking method but mention nothing of the speed of computation. On inspection of their algorithms I built a suspicion that the computational cost in such a method may directly rule it out for inclusion in my thesis.

2.5 W^{4}S: A Real-Time System for Detecting and Tracking People in $2\frac{1}{2}D$

In this system the authors have claimed to achieve results similar to those I aim to achieve. Their method differs slightly from mine however.

W^4S is a visual surveillance system that operates in real time, tracking people and segments of people robustly despite occlusion and other error-inducing factors. A combination of stereo images, shape analysis and shape tracking is used to locate and track both people and their respective parts. Models are created to track humans through interactions such as occlusions. It is possible to track multiple people at a frame rate between five and twenty frames per second.

W^4S is the integration of a real-time stereo system (SVM) with a real-time person detection and tracking system (W^4). SVM is a real-time device for computing dense stereo range images, developed by SRI.

Computational models are employed to detect and track people moving against static outdoor backgrounds. Attributes of models are recorded which allows better recognition of them following an occlusion or other instances where tracking is inhibited. Employing a stereo approach overcame problems encountered with sudden illumination changes, shadows and occlusion.

Foreground and background regions are isolated and analytically examined in the following manner; background scenes are statistically modeled by the minimum and maximum disparity values. This is performed periodically on both the disparity (stereo difference) and intensity backgrounds. The foreground regions detected are combined to form a unified set. Each of these is matched to the current set of objects using a combination of shape analysis and tracking. Predictive models are used to both estimate positions and correlate correct region identifications.

Foreground regions are detected in both the intensity image and disparity image simultaneously, objects being identified in a four stage process:

1. Thresholding
2. Noise cleaning (filtering)

3. Morphological filtering (accounting for distortions inherent in deformable bodies)

4. Object detection

If an object is found it is subsequently tracked. The detection method can be summarized as follows. The background is subtracted from the image. This leaves us with our foreground regions. Noise accumulated from possible background changes (trees in the wind, illumination variations) is filtered. This filtering involves eroding the entire frame and removing small foreground regions. If two regions are close together and look as if they are actually one (stereo helps here) they can be joined. After this process they were left with the foreground objects.

This article includes a section on advantages and disadvantages of stereo vs intensity based analysis. This is interesting but not enlightening enough to include here.

After the frame is segmented into various foreground regions, each of these has to be classified as either a new object or an object we're currently tracking. In case of occlusion of a person (by an object in the scene or another person), stereo images provided this team with helpful data on the respective ranges of their regions. Matching objects to current foreground regions is achieved by inspecting the present and previously predicted positions of the respective regions' bounding boxes. Where there is sufficient overlap in the bounding boxes the object has successfully been tracked. It's not all as easy as that however; objects splitting, objects joining, objects vanishing and objects materializing compound the matter greatly. W^4S employs different methods for each of these cases.

Before a new region is added to the list of objects it is followed through a few frames (to ensure it isn't noise). W^4S uses a second order motion model to estimate future positions for each object. Finding the position of objects is made more onerous by the fact that it is unreliable to use centroids etc. to measure the middle of a deformable body (i.e. a human).

If it is detected that an object has split and two foreground regions are now present in consecutive frames where there used only to be one, W^4S registers

one of these as a new object, one of them as the original and tracks both through subsequent frames.

If two regions merge they are tracked together as if they were one region and then tracked separately once they split. The stereo implementation applied here makes it relatively academic to compute the distance between regions when one covers another. Object attributes are kept and updated for the life of the object in the scene. This means that after occlusion objects can be readily identified.

In light of my intended final product, the next section of the article, Tracking People's Parts, is of great interest. To achieve the goal of identifying body parts in a recognized object, this team have created two models, a Cardboard model illustrating the approximate sizes and positions of head, torso and legs, and a motion model to be helpful in identifying feet, arms and hands. We are presented with statistical information on the relative size of body segments; the head, torso and legs are $\frac{1}{5}$, $\frac{1}{2}$ and $\frac{1}{2}$ of the total height of the object. The moments of foreground pixels are found in order to estimate the principal axis, providing us with information about the pose of the person. The head is located first, then the torso, then the legs. Hands are located after the torso by finding extreme regions which are connected to the torso and are exterior to it. This system isn't at all effective at estimating the position of the occluded hand in a normal stride.

The team that developed W^4S are intending to extend their implementation by adding models which can recognize the actions of the people it tracks (e.g. people leaving objects or people picking up objects).

Due to the obvious correlation between my thesis and these results I found this a very helpful reference.

Chapter 3

Theory

3.1 Hausdorff Distance

As mentioned in section 2.3, the Hausdorff distance of two edge sets can be used as an indication of their similarity in shape. By calculating this distance it is therefore possible to robustly recognize an object through a series of frames. It performs acceptably when objects undergo slight deformations.

3.1.1 Correct Hausdorff Distance

The Hausdorff distance between two sets of points is defined as follows[2]:

Given two sets of points $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$, the Hausdorff distance is

$$H(A, B) = \max(h(A, B), h(B, A)) \tag{3.1}$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{3.2}$$

$h(A, B)$ is the *directed* Hausdorff distance from A to B, identifying the point $a \in A$ that is furtherest from any point of B and finding the distance from a

to its nearest neighbor in B. If the Hausdorff distance ($H(A, B)$) is d, we know that every point of A lies within the distance d of a point of B.

To make the Hausdorff distance more robust we can ignore some points whose positions deviate greatly from the other points of interest; this is the *generalized* Hausdorff distance. Instead of finding the furtherest point we find the kth furtherest point. Possibly erroneous points are filtered out in this fashion. The function is given below.

$$h(A, B) = \underset{a \in A}{k\text{th}} \min_{b \in B} \|a - b\| \tag{3.3}$$

In mathematical terms, $h_k(A, B) \leq \delta$ if and only if there exists a $A_k \subseteq A$ such that $A_k \subseteq B'$, where A_k contains k points of A. This means a subset of the block we're looking for (A_k), or more accurately a representation of the block we're looking for (the block without some points), is found in the set of points B'. $h_k(A, B)$ effectively partitions the block we're looking for into the sets A_k and $A - A_k$. A_k is the set which is similar to B (within δ of B). $A - A_k$ contains the outliers which cannot be reliably resolved as a part of B.

This method can be implemented directly or approximated, as described in the following section.

3.1.2 Approximation to Hausdorff Distance

Due to the nature of the edge sets the tracking algorithm is comparing, it is advantageous to make this comparison algorithm as fast as possible. Computing a straight implementation of the Hausdorff Length algorithm for two vector sets of average length 300 co-ordinates takes approximately 90, 000 comparisons. In the Matlab environment this is far too slow to be of use in a real-time system. I found that it is possible to approximate the Hausdorff Method by a recursive loop which operates as follows:

1. Divide $V1$ and $V2$ by DIV.
2. Add one to $ITERAT$.

3. Find the set difference between the rounded values of *V1*, and the rounded values of *V2*, and store it in *D*, the indices of elements of *V1* that are not found in *V2* being stored in *IND*.

4. Replace *V1* with only the elements indicated by *V1(IND)*.

5. If D is empty, exit with Hausdorff length DIV^{ITERAT}.

6. GOTO 1.

where $V1$ and $V2$ are the two edge sets to compare and DIV is a number slightly greater than one. $ITERAT$ starts with the value 0 and increases until the difference between the rounded edge sets is zero. Larger values of DIV, e.g. 1.8 -> 2.5 converge to an approximation of Hausdorff length very quickly, but this advantage is outweighed by the inaccuracy incurred. Smaller values, e.g. 1.3 -> 1.5 are the best trade-off between accuracy and speed.

3.2 Histogram Matching

An indication of the colors present in a given area is useful for finding points within an object, histogram matching uses histograms as this indication. A typical color image will have Red, Green and Blue color layers. By creating a histogram for each layer we can describe a basic color set with frequency bins for each color.

As this application of histogram matching is implemented with a view to track points within an object, the templates defined around each point are relatively small; anywhere from 15 to 30 pixels square.

Comparing two templates entails the following steps:

- Finding the histograms for the red, green and blue components of each template
- Calculating the mean absolute error between the respective histograms
- Calculating the average error across all three colors

The value that results from this operation can then be compared to a threshold value to determine whether the templates can be considered similar.

The Matlab function to perform histogram creation, $IMHIST$, is an optimized function; it has been pre-compiled to operate on greyscale intensity images with a small computational cost. After the histograms have been created, their similarity is calculated by

$$sim = \frac{\text{mean}(\text{abs}(R1 - R2)) + \text{mean}(\text{abs}(G1 - G2)) + \text{mean}(\text{abs}(B1 - B2))}{3} \tag{3.4}$$

where $R1$, $G1$ and $B1$ are the red green and blue histograms of the matching template and $R2$, $G2$ and $B2$ are the histograms of the area currently being evaluated for similarity. A low value for sim indicates that the two templates in question have similar color distributions.

This method doesn't take into account the position of colors. The images shown in Figure 3.1 would be classified equal despite their different orientations.

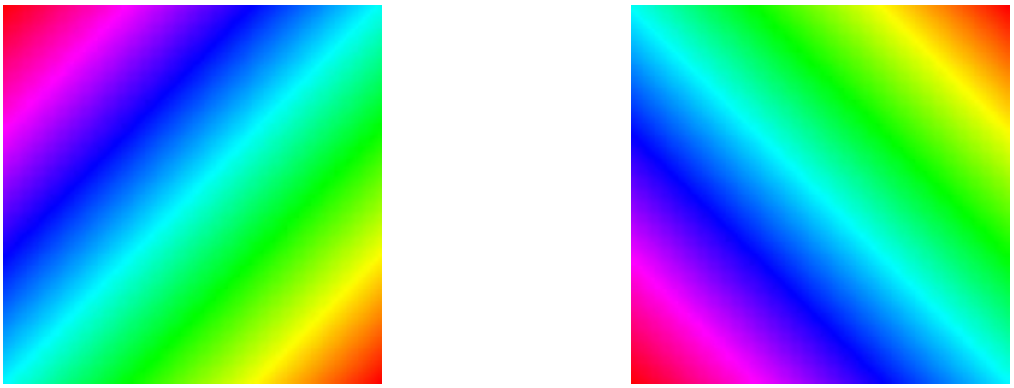


Figure 3.1: Two images with equal color distribution

Chapter 4

System Implementation

See Appendix A for Matlab code implementing this system.

4.1 System Structure

The flowchart shown in Figure 4.1 is a visual representation of the flow of operations of my object tracking system.

If points within objects are being tracked, these comparisons occur after storing the current object set and before frame annotation occurs.

As Figure 4.1 shows, after initialization of global variables and status checking, the first component of the system is to ask which points to track within an object. After the user has selected points, acquisition of the next frame starts the cycle of object tracking. The speed and quality of this step is limited by the image / video acquisition device connected to the computer; also the speed of any device drivers this step implicitly calls.

For testing purposes it is possible to direct this program to look for a sequence of frames (pre-recorded) on disk, using them for input data.

Following the acquisition stage of the tracking loop is the pre-processing stage.

4.2 Pre-Processing Methods

Before any scene analysis is possible, the image acquired must be 'cleaned up'. This involves removing the background from the image and removing any

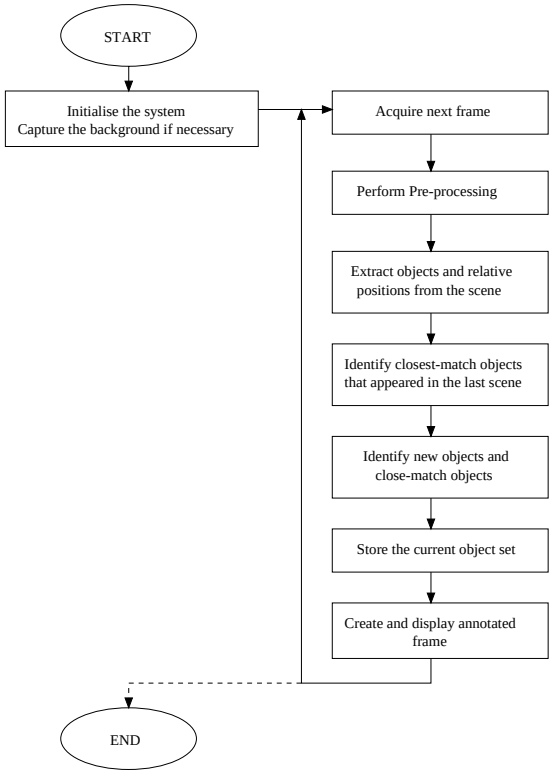


Figure 4.1: General structure of the object tracking system

noise generated either by the camera, variations in environmental lighting or the background subtraction process.

4.2.1 Background Subtraction

A simplistic, yet effective, method to remove the background from images in a video sequence is to simply subtract the background. This is achieved by a simple matrix subtraction operation.

$$F = I - B \tag{4.1}$$

where the foreground, F, is found by removing the background B from the acquired image I.

It is necessary to supply the algorithm with a background image, acquired before the tracking loop begins. This is the only part of the process which

is performed on color matrices. To conserve processing power in subsequent operations the matrix found from background subtraction is converted to an 8-bit greyscale intensity matrix.

To reduce noise from small intensity variations of the background, the intensity image, obtained by

$$f = \text{rgb2gray}(F) \tag{4.2}$$

is thresholded. The threshold value can be modified by the user. With the Logitech USB cameras a threshold value of approximately. 15 is recommended. With a Panasonic greyscale camera or a Sony Handycam, a threshold of about 6 is optimal. These translate into variances of 5.86% and 2.34% respectively. A mathematical representation of this process is

$$f^* = f - \text{thresh} \tag{4.3}$$

where *thresh* is the aforementioned cutoff. Figure 4.2 illustrates the background subtraction process. Note the discoloration of objects in (c).

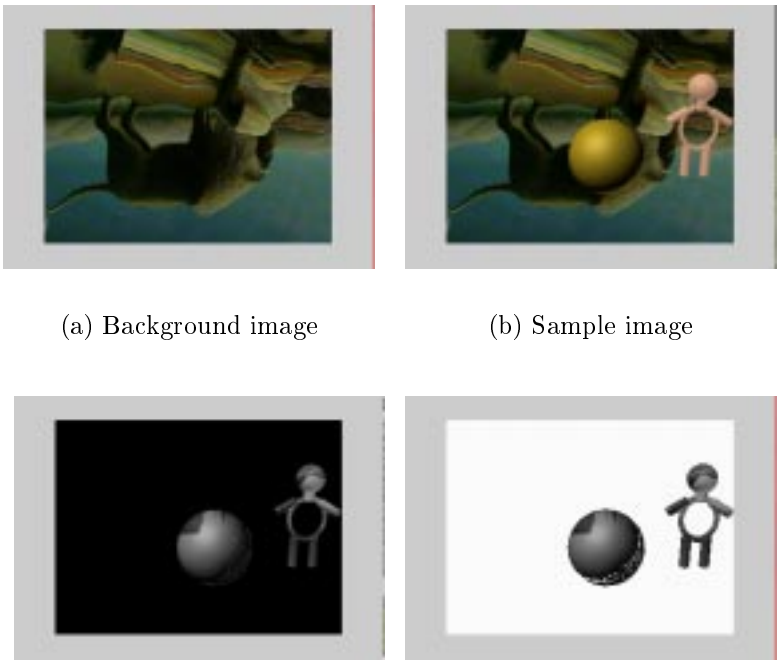
4.2.2 Noise Removal

After the foreground is found and a thresholding process is applied, a silhouette image is found by dividing the matrix by 255 and rounding towards infinity. This is represented by

$$SIL = \text{ceil}(f^*/255) \tag{4.4}$$

where ceil denotes rounding towards infinity. Any elements of the matrix f^* which are positive become 1, any zero or negative values become 0.

This is equivalent to



(a) Background image (b) Sample image

(c) After subtraction (d) After thresholding

Figure 4.2: Background subtraction

$$SIL = \begin{cases} 1, & f > \text{thresh} \\ 0, & f \leq \text{thresh} \end{cases} \tag{4.5}$$

for each element of f. The variable thresh is the cutoff described in section 4.2.1.

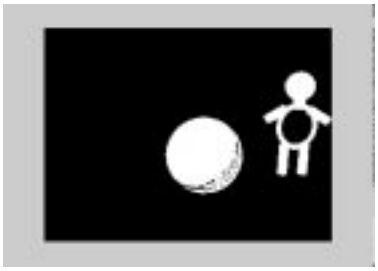


Figure 4.3: Conversion to binary image

Due to the background subtraction operation and variances in ambient lighting the resulting binary image is noisy. To clean this up we run it through some

binary image morphing algorithms. These are

- Erosion
- Cleaning
- Hole Filling
- Thickening
- Dilating

The erosion operation having syntax $SIL2 = \text{bwmorph}(SIL1, 'erode', 2)$ erodes each of cluster of 1s in the binary image. This removes a large amount of random noise in the system. Unfortunately it has the adverse effect of eroding the perimeter of objects whose features we'd prefer remain pristine.

The cleaning operation, with syntax $SIL1 = \text{bwmorph}(SIL, 'clean')$ removes all isolated 1s in the binary matrix SIL. If a pixel determined to be part of the foreground by the subtraction and thresholding operations has no connected pixels it is removed from the frame. This is a simple process for removing noise, yet effective when employed after the erosion operation.

The hole filling operation finds any small 'openings' in the foreground of the binary image. After locating the positions of these holes it performs a flood fill operation, effectively solidifying any areas that are detected as objects in the foreground.

The thickening operation thickens each area of 1s in the binary image without connecting any previously unconnected areas. This prepares the binary image for the dilation operation.

The dilation operation attempts to recreate the original silhouette, before the erosion operation, by dilating areas of 1s in the binary image.

Although all of these operations play an important part in readying the image for further processing, with certain camera configurations it is optimal to selectively choose which of these filters to apply. For example, in scenes where there is stark contrast between foreground and background intensities it may not be necessary to perform any preprocessing operations. This tailoring of preprocessing methods has the added advantage of increasing the frame rate;

they are comparable in computational cost with the Hausdorff shape matching algorithm. In the Graphical User Interface it is possible to choose which methods to use. It is also possible to view each frame after it is preprocessed. This functionality is included to make it possible to tailor the system for any configuration of processing power, lighting conditions and background / foreground contrasts.

Due to the intrinsic nature of these functions they are somewhat counterproductive. The noise is effectively removed from the system, but loss of edge information occurs. This means the edges of objects are less well defined, leading to a slightly less accurate Hausdorff measure in later stages. As these preprocessing functions are applied to all frames, however, the information loss is equal through all frames of a sequence, and this inaccuracy is relatively controlled.

4.3 Object Extraction Method

Isolated silhouettes are extracted from this binary image by the Matlab functions *BWLABEL* and *FIND*. *BWLABEL* labels each set of connected points in a binary image with increasing integers. It returns this matrix and the number of clusters of 1s it found. The function *FIND* searches this matrix for each area in turn, the number of areas supplied by the labelling function. Sparse matrices which indicate only the indices of located points are returned by *FIND*. Calculating the minimum x and y indices results in the objects' relative offset in the image. To effectively crop the image, resulting in smaller matrices, these offsets are subtracted from the set of x and y indices for this object. After converting these vectors back into normal binary matrices we achieve a silhouette of the object in question. Examples of such silhouettes are given in Figures 4.4.

This method results in a cell-array of matrices, each matrix representing a silhouette. Typically these matrices differ in size. Generally, if objects in the scene have relatively small silhouettes the system performs at a higher frame rate, hence the cropping operation. The speed of the Hausdorff algorithm, described in the proceeding section, varies geometrically with the size of the edge sets being compared.

Figure 4.4: Example object silhouettes

If an object is partially occluded, its corresponding matrix contains the silhouette of both objects as if they were one.

The corresponding x and y offsets of each object are computed and stored in two arrays. These are needed to correctly annotate the display to return correct visual data to the user.

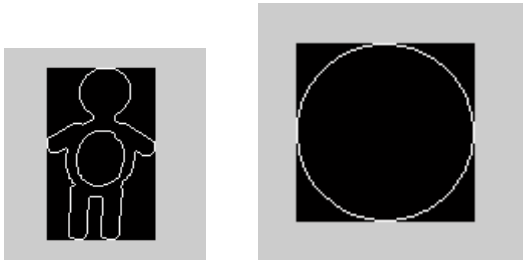


Figure 4.5: Example edge-detected objects

From the silhouettes we need to extract only information about shape. This is achieved by edge-detecting the silhouettes. Examples of the results of edge-detection by Matlab function *BWPERIM* are given in Figure 4.5. These outlines are used to create a two-column vector with as many rows as there are points on the perimeter of the object. Each row of the vector corresponds to a coordinate on the outline. Vectors from all of the objects in the present and previous scene are then used by the Hausdorff measure to indicate the degrees of similarity.

4.4 Hausdorff Methods

The Hausdorff Method, described in sections 2.3 and 3.1, is applied to calculate the Hausdorff Distance, a measure of similarity between objects (or vectors).

Given two two-dimensional column vectors, it calculates the distance of the most dissimilar point to its closest point in the other vector. As stated in section 2.3 it is sometimes advantageous to take the kth furtherest point, instead of the most outlying point. Generally this would add robustness to the algorithm but in the case of this system, due to the smoothing effects of the preprocessing steps, the functionality is similar when either method is applied. To ease computation I take the furtherest point.

When running in the Matlab environment this approximation works well to increase the speed of processing, however, by analyzing example procedures provided by Matlab I created a Dynamic Linked Library, 'haus.dll', which implements the correct Hausdorff measure in C code. The speed-up obtained by this compilation was enough to allow the correct Hausdorff measure to be used, achieving up to 3.5 frames per second as opposed to a maximum of a single frame per second with code running entirely in the Matlab environment.

4.5 Matching Algorithms

As the Hausdorff measure gives only an indication of how similar two objects are, it is necessary that these values be process further to determine which objects in the current scene are most closely related to objects in the previous scene. To achieve this end the Hausdorff values of all possible comparisons are found; all objects in the last scene are compared with all objects in the present scene. A small matrix of the results of these comparisons is formed. Detected objects in the present scene are marked as respective objects in the last scene after sorting the matrix of comparisons. Objects are matched if their Hausdorff distance is the minimum of all other comparisons with that object.

A cutoff for similarity is introduced; objects which have a value less than this threshold but aren't the closest match to any objects in the previous scene are given a matching value of 500, indicating they are questionable. Objects that don't achieve the cutoff for similarity are marked as unrecognized, or new objects.

In this fashion it is possible to recognize objects from scene to scene and to give a confidence of the match (a function of the Hausdorff values). The next step is to find the requested points within the object.

4.6 Histogram Matching

To match two areas in a RGB image, it is favorable to use all color information we can obtain. To obtain color information for the foreground regions, lost during the background subtraction step, the silhouette found in step 4.2.2, $SIL2$, is used as a mask against the original image. This translates to multiplying each element in the original image with the corresponding element in $SIL2$. Areas where no foreground objects are detected are marked with 0s in $SIL2$, and are kept black through this operation. Parts of the image that have been detected as foreground objects are kept intact.

After the foreground colormap is found, a search space is chosen. Areas similar in color composition to those requested are found. If an object is recognized as the object in which the requested points to be tracked were initially indicated, the search space only includes this object's silhouette. In any other case, the search space is all detected regions.

After the search space is defined, it's area is calculated to indicate an appropriate scaling factor for the histogram match. If the detected search space is not within 20% of the original size, an alternatively scaled histogram template is used. The production of these multiscale templates occurs when the user selected the points to track, Each template is formed by scaling histogram information for smaller or greater areas, all centered around the point chosen. For example, if the user were to select their nose to track the unscaled template may encompass their nose and the side of their eyes. The 50% template may then encompass their forehead, both eyes and their mouth.

Every nth point within the silhouette is then inspected for histogram similarity; if the returned measure of difference is less than a selected cutoff difference, the point and its respective 'closeness' is added to a list of possible points. If the point is more similar than all previously found points it is recorded as the best match. After all points to undergo comparison have done so, a weighted average is calculated for all similar points.

Both the weighted average and best match are displayed in the annotated frame. The weighted average is represented by a cyan circle, the best match by a white diamond. This annotation occurs during the histogram matching, but the frame is only displayed in the proceeding section.

The next step is to collate the data in a visual form, informing the user of objects detected and their recognition status. This annotation is described below.

4.7 Output Algorithms

The output stage of the tracking process gives the user a visual cue to what objects have been detected, what objects have been recognized and what objects are new to the scene. A recognized object has a green bounding box. A object that is dissimilar to all objects in the last scene is given a red box. Objects that meet the matching value cutoff but aren't recognized are given a blue box.

At the completion of processing the current frame, it is annotated and displayed as a figure using the Matlab functions *IMSHOW, RECT*, *TEXT* and *PLOT*. Each recognized object's bounding box is annotated with a letter which denotes the match between objects in the present and last scenes. A object will retain its letter if it is recognized as the same object through subsequent frames. If the object has been given a blue box it is marked with three question marks.

As the relative offsets of the objects in the image are stored in vectors in the object extraction method, it is possible to draw these rectangles around objects correctly.

Matlab makes it possible to edit the properties of figures currently displayed by use of handles. In a composite figure, i.e. a figure with many distinct areas of input and output events, handles can be requested to the children of the figure, i.e. each sub-section of the GUI. Each object has its own set of parameters, modifiable at any time by the use of Matlab function *SET*. In the parameters for the set of axes where the annotated image is to be displayed there are two parameters of direct interest, *DrawMode* and *AspectRatioRenderingType*. By ascribing these the respective values of *'fast'* and *'auto'* a slight increase in drawing speed was discovered. The cost of providing the user with data accounts for approximately 20 % of the entire processing time of this tracking system, so any increase is more than welcome.

User interaction is possible through the use of a Graphical User Interface, described fully in the next section.

4.8 User Interface

The User Interface, shown in Figure 3.7.1, provides for the user a set of buttons to control the operation of the tracking system.

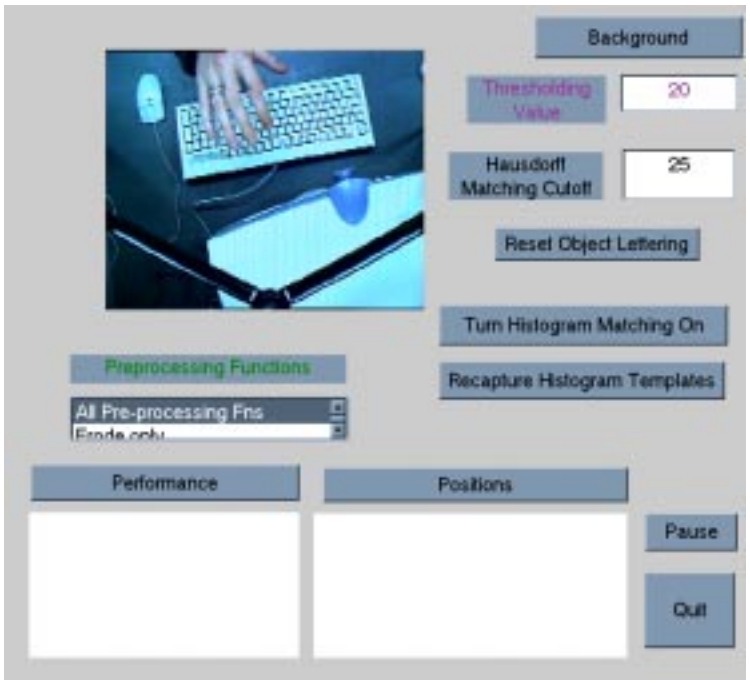


Figure 4.6: Graphical User Interface

Various performance and data analyses can be requested by use of the 'Performance' and 'Positions' pushbuttons, the continuously updated results of these calculations (if requested) are displayed in the text boxes beneath the image. More specifically, the frame rate is calculated and displayed, along with the number of frames processed and the total time spent running. Objects can be listed with either their center (displayed in x-y coordinates) or lowest Hausdorff matching values given.

The image is the last processed frame, annotated with visual information, bounding boxes for all detected objects. The bounding boxes are colored as described in section 4.6. The center of each object's bounding box is described by a blue pentagram. Histogram matching output is also displayed, cyan circles showing the weighted average of all comparisons performed and white diamonds indicating the best match for each template.

To recalibrate the background frame, necessary if ambient lighting conditions

have drastically changed, the user need only press the 'Background' button, then step away from the camera for three seconds.

Both the background threshold and Hausdorff matching cutoff can be manually specified by use of the edit boxes to the left of the image.

Histogram matching can be turned on or off at any time with the use of the so labelled pushbutton. It is possible to recalibrate the histogram matching templates at any time. By activating the 'Recapture Histogram Templates' button the user can choose new templates in the present frame.

It is possible to Pause and Resume processing at any time. If any of these buttons happens to be pressed, processing of the current frame will finish and the user's request will be processed before acquisition of the next frame.

Chapter 5

Results

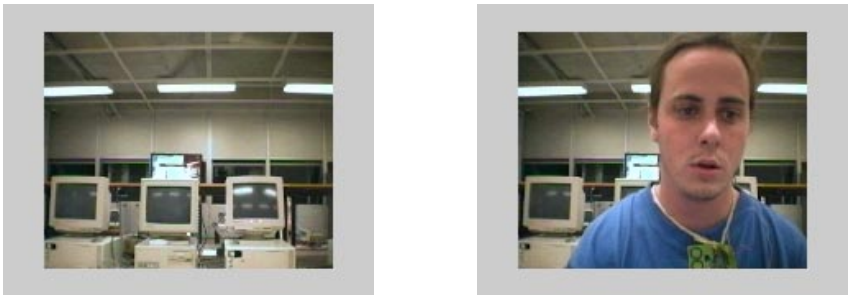
This section details results obtained from various sections of this system. The first section is an analysis of the foreground detection and pre-processing functions. The second section is a case study of the performance of the tracking algorithm (without histogram matching) across 35 frames. The third section is a case study illustrating both tracking and histogram aspects of the system operating through 15 frames. The fourth section is a profiling analysis detailing the relative computational cost of each algorithms. The final section details results achieved across a broader range of data sets.

Appendix B shows 12 frames of the output from the system when tracking three objects and performing point recognition in two separate objects. These images were not included in this section as they are not examples of real-world processing. They do provide an illustration, however, of my system's ability to track multiple objects. They were rendered in a 3D animation package as a test set for my system.

5.1 Foreground detection and pre-processing

Figure 5.1 shows the background image for this scene and a sample image. These images were acquired from a Logitech Quickcam USB camera at a resolution of 160x120 pixels. The background threshold for the system was set to 15, corresponding to a 5.86% loss of information across each color channel. Such a (relatively) high threshold is necessary in such frames due to the complexity of the background. The wide range of colors and intensities present,

as shown in Figure 5.1 (a), complicate the task of isolating foreground regions considerably.



(a) Background image (b) Sample frame

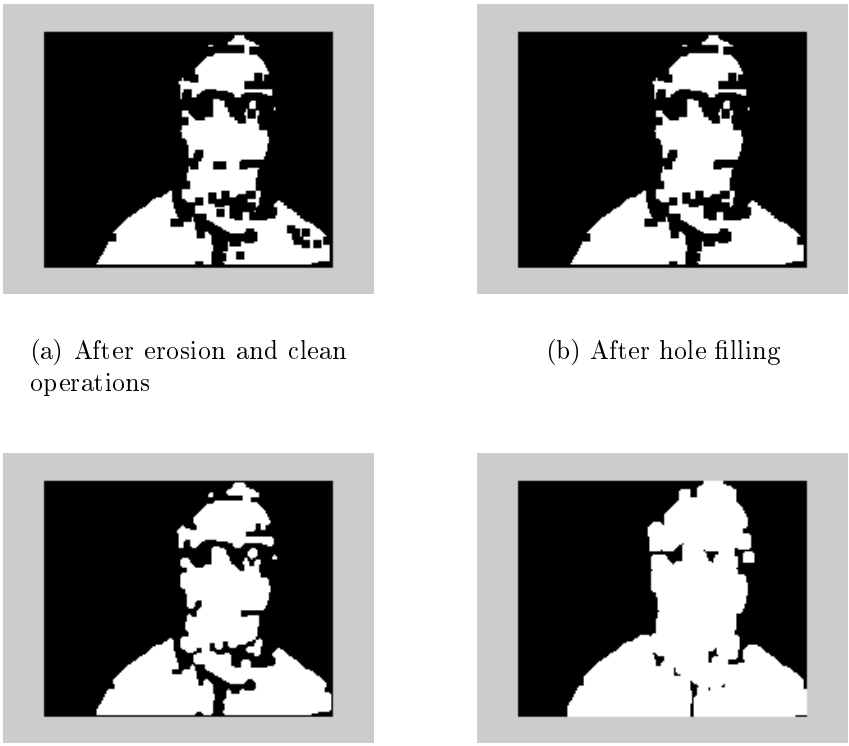
Figure 5.1: Background image and sample frame

Figure 5.1 shows a background image for a scene and a sample image from a set of frames taken in that scene.

(a) After background subtraction (b) After thresholding

Figure 5.2: Data during background subtraction

Figure 5.2 shows the images resulting from background subtraction and the proceeding thresholding operation, respectively. In the second image of Figure 5.2, the white areas are detected as background regions; conversely, the grey areas are detected foreground regions. Figure 5.2 illustrates the drawbacks of background subtraction. The ideal output of a foreground detection operation is a mask matrix, that, when applied to the frame in question, indicates all of the foreground regions and none of the background. With a simple background, e.g. a bluescreen, background subtraction more closely resembles this ideal.



(a) After erosion and clean operations

(b) After hole filling

(c) After thickening

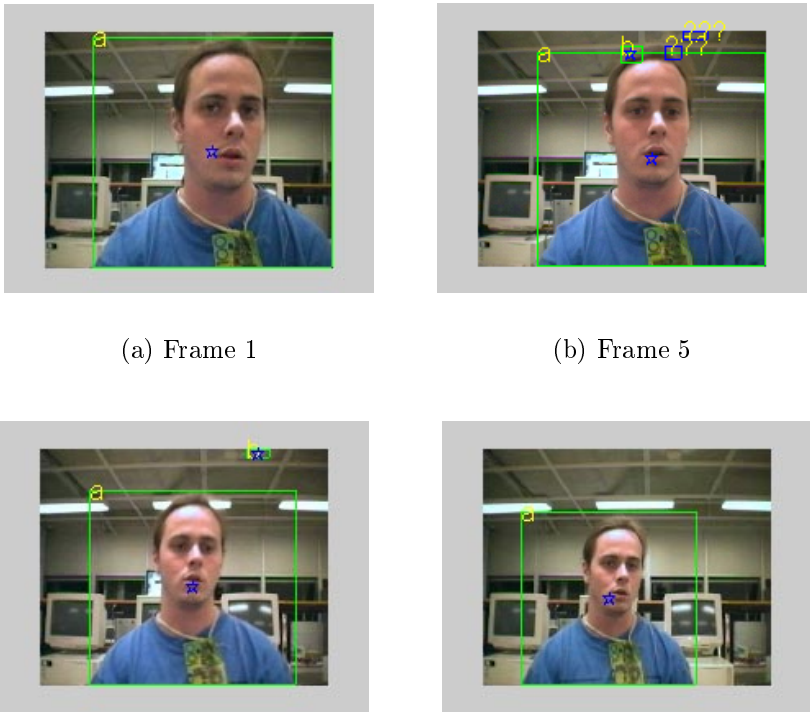
(d) After dilation

Figure 5.3: The noise minimizing procedure

Cleaning and noise removal is the next operation to be employed; Figure 5.3 illustrates each step of this procedure. The noise incurred from similarity in color and intensity of foreground and background regions becomes obvious upon inspection of Figure 5.3 (a). The erosion and cleaning operations, operating on the binary equivalent image of Figure 5.2 (b), remove any small areas of, usually erroneous, detected foreground. The hole filling operation acts as its name implies, filling any regions of 0s bounded by 1s with 1s. The thicken operation grows the edges of the silhouette without connecting any previously connected pixels. It can be seen that this improves the result of the dilation operation (Figure 5.3 (d)).

5.2 Case Study of Object Tracking

Herein is a selection of frames illustrating the operation of the aforementioned tracking system. It can be seen that the system performs robustly despite deformations of the object being tracked. The background threshold of the system was 15 and the matching criteria cutoff was 30. The following frames were captured using a Logitech Quickcam USB camera. A Pentium III processing these frames by running the proposed tracking algorithm achieved a frame rate of 2.6 frames per second (fps). Each image shown is a scaled representation of a 160x120 pixel RGB image presented to the user as visual feedback.



(a) Frame 1 (b) Frame 5

(c) Frame 10 (d) Frame 14

Figure 5.4: Case Study 1: Frames 1, 5, 10 and 14

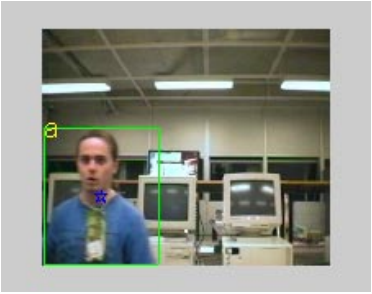
Figure 5.4 shows four frames selected from a series of processed images. Frame 1 consists of the objects initially recognized and segregated in the scene, in this case the sole human was given the nomenclature 'a'. In frames 5 and 10 the system is recognizing the ceiling fans as new objects as they differ from the background image significantly. Frame 5 shows that part of the object being

tracked is not recognized as belonging to that object. The shape deformations perceived by the camera of the object throughout these frames is relatively small. Hausdorff distances in this set are range between three (3) and 19.

(a) Frame 18

(b) Frame 20

(c) Frame 21



(d) Frame 22

Figure 5.5: Case Study 1: Frames 18, 20, 21 and 22

Through frames 18 to 22, illustrated in Figure 5.5, the object quickly reduces in size. This reduction results in Hausdorff values of 15 through 22. The object is still within the matching cutoff criteria of 30 and therefore keeps it's lettering.

As the object leaves the scene, shown by Figure 5.6 (a) - (c), it's area, as seen by the camera, is reducing slowly. It doesn't leave the scene completely however. Part of the object, the person's head, is detected as the original object (Hausdorff distance 26, 4 below the matching cutoff). The rest of the person is a questionable object. It looks like an object seen in the previous scene (Hausdorff of 29), but another object is more similar. As the person moves back into the scene, in Figure 5.6 (d) and (e), he is recognized as a new object, and subsequently tracked walking towards the right.

5.3 Case Study of Tracking and Histogram Matching

This section gives a case study of the system performing object tracking and histogram matching through 15 frames. A background threshold of 6 was used and a matching criteria of 25 was applied. The images were acquired through use of a Sony Handycam connected to a Pentium III desktop computer. On this system the frame rate was 0.8 fps. Each image is of resolution 160x120 pixels, with red, green and blue intensity planes. Each frame's caption includes a Hausdorff distance for the main object in the scene, the person being tracked.

The white dots in the frames of this case study were hand added as small white diamonds are difficult to see at this resolution on the printed page. They were placed directly above the diamonds, representing the histogram matching algorithm's best matches in each fame.

The points being tracked by histogram matching are marked by white diamonds in Frame 1, illustrated in Figure 5.7 (a). Through these, and subsequent frames it is seen that this method is sporadically erroneous, more so for matching the point on the body. The template over the middle of the body travels around the blue shirt and even to the arms of the object due to the similarity of color compositions present in these regions. The point on the face is tracked reasonably well throughout this set of frames, always being found on the face, often right on the nose.

The weighted mean positions, marked by cyan circles, have a much lower variance in frames 1-4, remaining in the middle of the body. It is seen that when the points on the object are initially chosen, shown in Frame 1, these averages are located on the neck. Through the next frames, as more of the object comes into view, the averages move down relative to the object. This is explained by the increase in visible skin, and seeing more of the blue shirt, thereby affecting a mean that occurs closer to the center of the body. At similar scales, when the object appears to be approximately the same size between frames, these averages can be seen to remain relatively constant.

Figure 5.8 illustrates the object breaking into two parts due to similarity of foreground and background regions across part of the object.

In Frame six, shown in Figure 5.8 (b), the best match for the body template is

seen to be located on the object's right arm. This is explained by the similarity in hues between the greens of the tag around the object's neck and the shorts, and the similarity in hue of the arm and neck regions.

Despite deformations and scale differences this object has undergone it still retains its labelling as object 1.

In Frame 9, Figure 5.9 (a), the area marked as object 'c' results from shadows cast by object 'a'. In the two following frames object a's left hand is segregated as another object. As it is similar in shape to the erroneous shadow, it is (wrongly) recognized as object 'c'.

Frame 10, Figure 5.9 (b), shows the body deforming greatly. The outstretched arm causes the object to be treated as a new shape introduced to the scene. Hence the bounding box is red. It is labelled with a 'd', a letter not previously used.

The relatively large amount of similar blue hues causes the histogram matching algorithm to erroneously move the best match of the body template around the body.

In frame 13 through to frame 15, Figure 5.10, the object splits into two, rejoins, and is finally recognized. The lettering of the object is now 'g'.

5.4 Profiling Results

It is possible to break down the computational load with respect to time spent processing each function run in the operation of my system. This gives an indication of the efficiency of each aspect of the system.

A breakdown detailing the cost of each large function run when performing only object tracking is given in Figure 5.11.

From Figure 5.11 we see that the hole filling operation is disproportionally expensive computationally. If we choose not to include this preprocessing function the breakdown shows a much more even distributing of processing power between functions. A table of performance without the hole filling operation is given in Figure 6.2.

Without the hole filling operation the system performs at an average speed of 2.8 frames per second. This speed is acceptable for real-time processing.

The accuracy lost by not applying this operation is minimal in comparison to the benefit in speed of the system. The relative breakdown when using this configuration of preprocessing functions is given in Figure 5.12.

Performing histogram matching as well as object tracking decreases the average frame rate of the system to approximately 0.9 fps. Figure 5.13 shows the breakdown of computational cost for functions when histogram matching is being performed. With each added point being tracked, the Histogram Comparison section increases in speed by a third of its speed when performing single comparisons.

From these figures it is apparent that my object tracking system is capable of functioning well in real-time. If point tracking operations need to be performed, they can only run at approximately 0.9 fps.

5.5 General Results

A good indication of one aspect of the performance of a real-time system is the speed it runs at, measured in frames per second. Frame rates of up to 3 fps were achieved when reading the frame sequences directly from the hard disk and performing only the tracking aspects of the system. This frame rate drops to between 1.8 and 2.5 fps when a Logitech Quickcam USB is used to capture images. These frame rates are for the processing of 160x120 pixel RGB frames. Results achieved when using a Sony Handycam were comparable to reading each image off non-volatile storage. Frame rates of up to 2.8 fps were achieved with this configuration. Frame rates when using the Panasonic security cameras were approximately 2.6 frames per second, on average, but these cameras only supply intensity information and result in poorer performance of both the tracking and histogram aspects of this system.

Running the system on a Pentium II 350 MHz processor, reading each frame from hard-disk, speeds of up to 2.2 fps were achieved for object tracking. A frame rate of approximately 0.6 fps was achieved when tracking objects and two user-selected points.

The speed of the system alone, however, is not a suitable measure of performance. Quality of object recognition and point recognition within the object must also be analyzed to give a more objective assessment of systems such

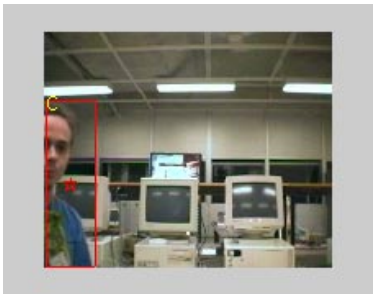
as mine. The case studies displayed in sections 5.2 and 5.3 are indicative of the general accuracy of the system. When more objects enter the scene, for example another person is visible, the speed of the tracking system is slightly impeded and the accuracy of the comparison and matching is slightly reduced, in general. This is due to occasional discrepancies as to which object is recognized as which. These errors only occur when the objects are of comparable size and shape with respect to the camera.

As is shown in the preceding sections of this chapter, occasional problems arise when the scene consists of a complex background with many color and intensity variations. With more hues and intensities, it is increasingly probable that an object in the foreground will share similar attributes with the background, and be split into multiple objects.

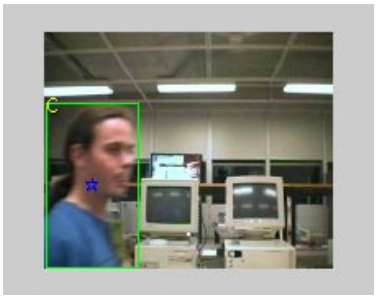
(a) Frame 23

(b) Frame 25

(c) Frame 26

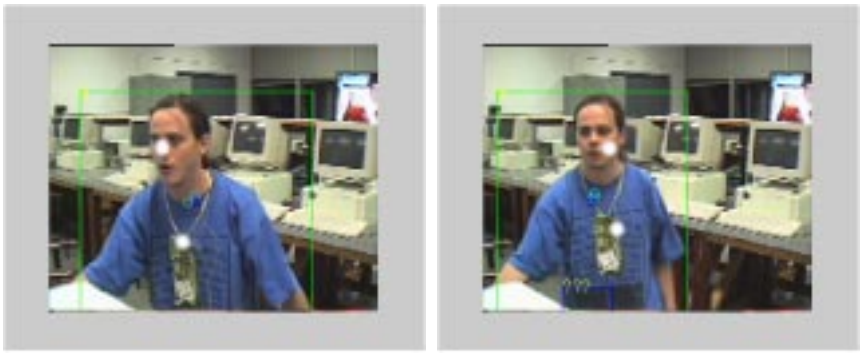


(d) Frame 28



(e) Frame 30

Figure 5.6: Case Study 1: Frames 23, 25, 26, 28 and 30



(a) Frame 1, H: 0

(b) Frame 2, H: 13

(c) Frame 3, H: 18

(d) Frame 4, H: 6

Figure 5.7: Case Study 2: Frames 1 through 4

(a) Frame 5, H: 9

(b) Frame 6, H: 10

(c) Frame 7, H: 5

(d) Frame 8, H: 6

Figure 5.8: Case Study 2: Frames 5 through 8

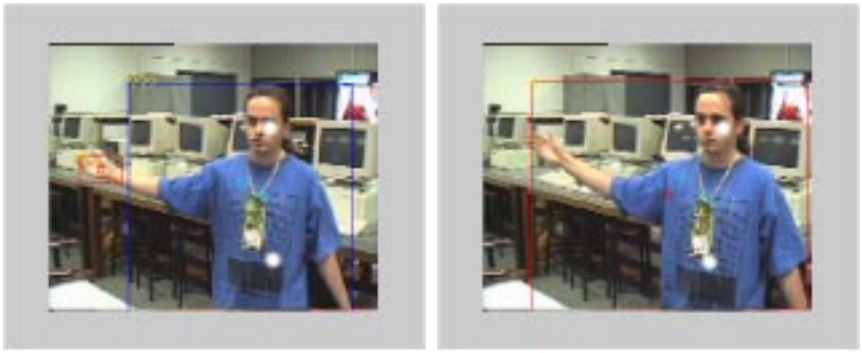
(a) Frame 9, H: 17

(b) Frame 10, H: 32

(c) Frame 11, H: 23

(d) Frame 12, H: 11

Figure 5.9: Case Study 2: Frames 9 through 12



(a) Frame 13, H: 26 (b) Frame 14, H: 28

(c) Frame 15, H: 7

Figure 5.10: Case Study 2: Frames 13 through 15

Function	% of processing time	Time for one 160x120 frame (sec)
Background Subtraction	4	0.05
Conversion to Binary	2	0.02
Erosion	4	0.05
Cleaning	1	0.01
Hole Filling	65	0.74
Thickening	10	0.14
Dilation	2	0.02
Object Isolation	4	0.04
Object Comparisons	4	0.05

Figure 5.11: Relative processing time spent when tracking objects

Function	% of processing time	Time for one 160x120 frame (msec)
Background Subtraction	12	57.6
Conversion to Binary	4	19.2
Erosion	10	48
Cleaning	4	19.2
Thickening	28	134.4
Dilation	7	33.6
Object Isolation	13	62.4
Object Comparisons	14	67.2

Figure 5.12: Relative processing time when tracking objects without Hole Filling operation

Function	% processing time	Time for one 160x120 frame
Background Subtraction	4	58
Erosion	3	19
Thickening	9	135
Dilation	2	29.7
Object Isolation	4	61
Creating Prospective Templates to Match	12	183
Template Comparisons	40	600
Object Comparisons	4	63

Figure 5.13: Relative processing time when tracking objects and performing Histogram matching

Chapter 6

Discussion

The system performs at its best when there is little fluctuation in the environmental ambient lighting conditions. If there are any windows in the proposed tracking environment, the output can be slightly effected by passing clouds and shadow movements. If operating indoors in an environment with many windows, the system will function to an acceptable quality for approximately three hours during the day (due to movements of shadows cast by objects in direct or reflected sunlight). In a closed office environment or during the night the system will work at a fine quality until the lighting in the room is changed severely (a light is extinguished or the sun rises, respectively). Any fluctuations in ambient light introduce small superfluous objects to the scene, small areas of distinct variances, e.g. a reflective surface that was previously in shadow is suddenly illuminated.

Due to the speed of the cameras and their respective interfacing functions, the user is forced to choose a resolution of 160x120 pixels if they wish to track objects in real-time. This is the higher limit of camera resolutions acceptable if the real-time implementation is run on a PC with processing power comparable to that of a Pentium III, 450MHz. We can achieve, on average, between 2 and 2.6 frames per second. If we lower the resolution we gain speed quickly, but are faced with loss of accuracy. Analyzing image sequences shows that it is possible to attain speeds of up to 1.4 frames per second at resolution of 320x240 pixels.

The background subtraction method implemented works suitably with a relatively complex background. If the background is simple, low-quality cameras

can be implemented with success. If the background is more complex higher image quality is needed to appropriately detect the foreground regions.

As seen in Figures 5.11 through 5.13, the preprocessing methods are computationally expensive, but they are necessary if one or more of the following is true

- The lighting conditions of the room vary,
- The camera adjusts its gain, exposure or white-balance automatically,
- The camera introduces a significant noise into the picture,
- The background is very complex and shares colors with the objects moving through the scene.

Most usually at least one of these four criteria is true. By tailoring the preprocessing functions the system is using, with the GUI, the user can optimize the performance of the system while it is running.

As can be seen in Figure 5.3 of the results section, these noise-removal operations recreate a semblance of the shape the human eye would perceive. Unfortunately, however, the binary image produced has lost definition in its edges. If the preprocessing functions resulted in a perfect silhouette of the object, lower Hausdorff matching criteria could be chosen. This would result in a more robust system. Despite the information lost, however, the system performs with acceptable accuracy.

The object detection section of the system works well to isolate each silhouette in the image it is presented with.

The Hausdorff distance calculation algorithm when implemented in Matlab is prohibitively slow. At most, a frame rate of 0.3 frames per second can be achieved. After this Matlab code was converted into C and compiled into a '.DLL' file, a speed-up of more than 1100% was experienced. The approximation to the correct Hausdorff distance ran at a maximum speed of 1.5 fps. The compiled correct Hausdorff algorithm was chosen for use for both computational and accuracy reasons.

The nature of my implementation of histogram matching, more specifically, my choosing to use red, green and blue histogram comparisons, rules it out for inclusion in a real time system. If the Matlab code were compiled, however, into

a stand-alone process, it is possible that the speed increase would be enough to feasibly include histogram matching in real time operation. Alternatively, if only greyscales intensities were used, it would be immediately possible to implement these algorithms to run at an acceptable rate for real-time processing. This, however, would reduce the accuracy of the matches greatly. This also applies for reduction in size of the templates being matched.

Due to the nature of the human body, deformations of the representative silhouette are at times severe. Small deformations (e.g. a person moving his legs and arms slightly when walking) pose no real problems to my tracking algorithm. Larger deformations, however, such as a person bending over to pick an object up, or a person suddenly pointing into the sky (illustrated in Case Study 2 in the results section of this thesis) when their arms were previously situated by their side, result in the person not being recognized and subsequently being labelled as a new object to the scene. One way to overcome this problem is to integrate histogram matching and object matching closely. If an object is not recognized, but multiple points are detected within that object as being points chosen by the user to track, it could be provisionally accepted as being the original image.

The Graphical User Interface presented to the user makes the system easy to both use and maintain. It also gives the user options to optimize the system for the current working environment and environmental conditions.

Chapter 7

Conclusion

Overall, this thesis successfully illustrates the feasibility of running a real-time object tracking system on a conventional desktop computer with simple Video-for-Windows capture devices. Hopefully, this thesis will improve awareness of potential image processing applications in the general engineering field.

Any sequence of images, whether they be captured and stored on a medium or acquired from a video device in real-time, can be analyzed by this system given that the following are true

- A background frame for the scene is available
- The camera is not moving with respect to the scene being viewed
- Any variations in lighting conditions of the background are minimal

The tracking system described performs effectively, it is able to track objects through a scene with a relatively high degree of accuracy. The silhouette of a person moving through the scene was successfully recognized from frame to frame, despite the deformations this non-rigid body undergoes and apparent deformations due to rotations of the object. New objects entering the scene are recognized and given an identity to be tracked through subsequent frames. The system runs at an acceptable rate for real-time processing on desktop PCs with off-the shelf video hardware. Frame rates of up to 2.6 fps (frames per second), an adequate rate for a real-time system, were achieved.

Point location within an object was also implemented with a relative degree of success. Though not operating at a frame rate necessary for real-time processing, this aspect of my thesis expanded its potential scope to include point-tracking analyses through sequences of pre-recorded frames. Analysis of data that isn't time-critical is the direct focus of this portion of my thesis.

Chapter 8

Future Work

8.1 Different Pre-Processing Methods

My choice of background subtraction as my foreground extraction process was mainly guided by the ease of implementation, the relatively small amount of processing involved, and therefore the high speed of the algorithm. Other algorithms to achieve this end could be employed if processing power permits. With more precise foreground detection algorithms the 'Hausdorffian' matching would become more accurate, making the entire system more stable.

If other foreground detection methods were employed it would be possible to track multiple foreground objects with a non-stationary background. This would involve mapping the frames from the camera into a moving vector space. Any parts of the image that don't conform closely to the mean movement of the frame would be classified as being part of the foreground. This has been successfully achieved by a number of researchers, [7][8].

Temporal filtering, filtering the frame's moving objects by mapping parts of the image that have changed from frame to frame, would be a useful extension if we were to assume that all objects in the scene were moving at all times. If we took a large time slice, e.g. the last 6 frames, we could avoid this problem. However, with larger time slices more 'shadowing' occurs. This is when an object moves and the background is revealed. The background is a part of the image that has changed, and so is the new position of the object. These 'copies' make temporal filtering difficult to implement with the aforementioned tracking algorithm.

A colour segmentation algorithm would be an interesting replacement, or addition to background subtraction. The image could be broken into regions depending on the continuity of colors from pixel to pixel. If a person were wearing a blue shirt and green shorts, the color segmenter would recognize each of these as distinct objects. This could be applied to validify an object recognition, constraint a block matching algorithm or perform object recognition.

8.2 Tracking Points within an Object

The block-matching (Histogram Matching) algorithm I've employed is a simple, yet effective solution to the problem of locating points within a scene. More comprehensive block matching algorithms could be employed to perform this task if they could either be implemented more efficiently, precompiled and optimized or if processing power were more abundant.

Relative Projection Histograms are an extension to the form of Histogram matching that I've employed. Instead of building just one histogram for the templates, relative projection histograms consist of a histogram for each individual row and column of the template. Section 3.2 illustrates the orientation independence of simple Histogram matching; with relative project histograms blocks will only be matched if their orientations are similar.

These additions would most likely have the effect of an increase in accuracy over the rudimentary system I've employed. If they could be implemented without a noticeable decrease in overall system speed it would, obviously, be advantageous.

8.3 Memories of Objects

The present implementation of this tracking algorithm has only a one-frame memory. Objects detected in the present scene are compared only with objects detected in the last scene. If we wanted to always recognize objects with a particular shape, suitcases or people for example, a simple extension to the matching algorithm to include these comparisons would be possible. This

leads us to an object (shape) classification system as well as an object tracking system.

Upon first inspection, the overhead involved in such a project would be too great to enable it to run in real time. If a measure with a simpler data set than the Hausdorff measure were used (i.e. a single, relatively small one-dimensional vector as opposed to two vectors of lengths up to 300 rows) the possibility of implementing such a system would be more distinct.

8.4 Measures of Deformation

As the human body is not of a rigid form, quite apparent when it is viewed walking through a scene, it undergoes a large amount of deformation. These deformations could foreseeably be mapped to a three-dimensional model of the general human form. Pose, probable position of limbs, could be estimated and used to extract information from the scene. It is potentially possible to model-fit data from the histogram matching algorithm to this human model and record the poses throughout a video sequence. These poses could then be used in any number of applications, including analysis of a golf swing, an athlete's running style or the range of movement recovered in post-operative physiotherapy sessions. Alternatively, the data could be used to control multimedia applications such as 3D animation packages or 3D sound applications.

8.5 Stereo Imaging

By either using two cameras, or two stroboscopic light sources, it is possible to infer much more accurate positional data from a scene. By measuring either the positions of points or the length and orientation of shadows in both images and calculating the parallex error it would be possible to obtain range information. A 3D model of an object in the scene could be created automatically. A model of the static environment could be used to assist the system making accurate predictions. A simple application of this could be a camera mounted in a room warning a person away from a hot oven if they were to step within the predefined boundary.

Bibliography

[1] E Dougherty, P Laplante: Introduction to Real Time Imaging, IEEE Press; New Jersey, 1995.

[2] Anon: Hausdorff Image Comparisons, '*http://simon.cs.cornell.edu/Info/People/dph_Hausdorff/hausdorff.html*', Cornell University, 1994.

[3] J Morel, S Solimini: Variational Methods in Image Segmentation, Basel: Birkhause, 1994.

[4] D Huttenlacher, G Klanderman, W Rucklidge: Comparing Images using the Hausdorff distance, IEEE Trans on Pattern Analysis and Machine Intelligene, Vol 15, No 9, pp 850-863, 1993.

[5] I Haritaoglu, D Harwood, L Davis: W^4S: A Real-Time System for Detecting and Tracking People in $2\frac{1}{2}D$, Proc, ICPR, IEEE, Los Alamitos, 1998.

[6] J Cheng, M Jose: Tracking Human Walking in Dynamic Scenes, Proc, ICPR, IEEE, Los Alamitos, 1997.

[7] D Huttenlocher, W Rucklidge: A multi-resolution technique for comparing images using the Hausdorff distance. Proc, ICPR, IEEE, Los Alamitos, 1993.

[8] D Huttenlocher, J Noh, Rucklidge: Tracking non-rigid objects in complex scenes. Proceedings of the fourth international conference on Computer Vision. pp93-101, 1993.

[9] D Huttenlocher, L Lorigo: Recognising 3-Dimensional Objects by comparing 2-Dimensional images, Proc ICPR, pp878-884 IEEE; Los Alamitos, 1996.

Appendix A

Code listings

The software I've written to implement my object tracking and point tracking system can be separated into modules for ease of understanding. These are

- The control module, taking care of initialization, capturing each frame and controlling the flow of operations.
- The preprocessing module, performing background subtraction and noise removal.
- The object detection and isolation module
- The tracking module, performing Hausdorffian inquisitions and interpreting the results. This module also performs some frame annotation.
- The area location module. Here the point within the object is found, if possible, and the frame is annotated accordingly

A.1 Control Module

The following is Matlab code. Comments are demarked by a percent sign (%) at the beginning of a line.

```
% R-T. O. T. S.
%
% Real Time Object Tracking System
```

```
%
%  Written by Robert Andrews
%
%  Last Modified: Tuesday 12th October
%
function SetupAndControlLoop
global chile;     % Handles for GUI Children
global guifunc;   % Selected GUI function
global paused;    % Are we paused?
global preproc;   % Which pre-processing?
global perftype;  % Which performace monitoring?
global perfcount; % How long have we been running?
global thresh;    % Background Threshold
global scalar;    % For later implementation, downscaling
global hcut;      % Hausdorff Matching Cutoff
global hist;      % Histogram matching On or Off
Setup;
guifunc = 0;
%
% Read the background image from disk.
%
B = imread('background.tif');
hcut = 25;
histsize = 15;
%
% Yellow is a counter which indicates the current frame
%
yellow = 1;
firstrun = 1;
figure(1);
tic;            % Start the stopwatch
while(guifunc ~= 99)
  %
  % We can choose to acquire frames from the hard disk.
  %
%  s = strcat('.\monday11th\anim',letter, num2str(yellow), '.tif');
%  a = imread(s);
```

```
I = vfm('grab', 1);
for count = 1:1;
   a = I(:, :, :, count);
   argb = a;
   %
   % Perform background subtraction and thresholding
   %
   a3 = backnoise(a, B);
   if hist == 1
     a5(:, :, 1) = uint8(double(argb(:,:,1)) .* double(a3));
     a5(:, :, 2) = uint8(double(argb(:,:,2)) .* double(a3));
     a5(:, :, 3) = uint8(double(argb(:,:,3)) .* double(a3));
   end
   figure(1);
   imshow(goodness);
   [objs, num, xoff, yoff] = ObjectIsol(a3);
   if yellow == 1 & hist == 1
      figure(1);
      [xinp yinp] = getpts;
                    xinp = round(xinp);
      yinp = round(yinp);
      %
      % Create histograms of templates around
      % User input
      PointLook = imcrop(a5, [xinp(1)-(histsize/2)
                        yinp(1)-(histsize/2) histsize histsize]);
      RPL = imhist(PointLook(:, :, 1));
      GPL = imhist(PointLook(:, :, 2));
      BPL = imhist(PointLook(:, :, 3));
      count = 1;
      %
      % Next the scaled histograms are made
      %
      for coun = .8:-.1:.3
         count = count+1;
        PointLook = imcrop(a5, [xinp(1)-(histsize/coun/2)
          yinp(1)-(histsize/coun/2) histsize/coun histsize/coun]);
         RPL = [RPL , imhist(imresize(PointLook(:, :, 1),
```

```
                [histsize histsize]))];
    GPL = [GPL , imhist(imresize(PointLook(:, :, 2),
                [histsize histsize]))];
    BPL = [BPL , imhist(imresize(PointLook(:, :, 3),
                [histsize histsize]))];
  end
  %
  % Find the occluded area of the first frame.
  % This is used as a basis in future frames
  % to calculate the scaling factors
  %
  [xxx yyy] = find(a3);
  origsize = size(xxx);
end
        if hist == 1
[histxvect histyvect] = find(a3);
bestyet = 100000;
xlist = 0;
ylist = 0;
histvalues = 0;
goodpoints = 0;
scaling = size(histxvect) / origsize;
ccc = 0;
cccd = 0;
        factorage = 1;
        if scaling < 0.8
          factorage = round((1 - scaling)*10);
          if factorage > 3
        factorage = 3;
        end
        end
for xcount = 1:30: size(histxvect)
  if histxvect(xcount) < (size(a5, 1) - histsize)
    if histyvect(xcount) < (size(a5, 2) - histsize)
      cccd = cccd + 1;
      check =  imcrop(a5, [histyvect(xcount)-histsize/2
        histxvect(xcount)-histsize/2 histsize histsize]);
      hval = rph(RPL(:,factorage), GPL(:,factorage),
```

```
                BPL(:,factorage), check);
                    end
                      ccc = ccc + hval;
            if hval < 4.5
              goodpoints = goodpoints + 1;
              xlist(goodpoints) = histyvect(xcount);
              ylist(goodpoints) = histxvect(xcount);
              histvalues(goodpoints) = hval;
              if hval < bestyet
                ggx = histyvect(xcount);
                ggy = histxvect(xcount);
                bestyet = hval;
              end
                  end
                   end
      end
     end
    ccc = ccc / cccd       % Display the average hval.
    end
    %
    % Annotate the frame with the best point found
    % and the weighted average.
    %
   if goodpoints > 0
    gx = mean(xlist) + (xlist - mean(xlist)) / (histvalues/(4*mean(histvalues)));
    gy = mean(ylist) + (ylist - mean(ylist)) / (histvalues/(4*mean(histvalues)));
     hold on
     plot(gx, gy, 'co');
     plot(ggx, ggy, 'wd');
    end
  end
  if firstrun == 1
    % Setup all the object matrices and vector lists for
    % the first run.
    last = objs;
    firstrun = 0;
    chars = ['A'];
    for a = 1:size(last, 2)
```

```
        [c b] = find(bwperim(last{a}));
        list2x(a) = {c};
    list2y(a) = {b};
    chars(a) = char(96+a);
  end
end
%
% If there are any objects in the scene, enter the tracking
% algorithm.
%
if size(xoff,1) > 0
  [list2x list2y, chars] = tracks(objs, list2x, list2y, xoff, yoff, chars);
end
yellow = yellow +1;
drawnow;
if paused
  waitforbuttonpress;
  paused = 0;
end
if perftype == 1
  s = sprintf('%d Frames processed\n\n%d elapsed seconds\n\n
                %d frames per second', yellow, toc, yellow / toc);
  set(chile(12),'String', s);
end
end
close all;
end

%       Notes:
%               With the Panasonic videos, it seems that it is best
%  to use a background threshold value of around 4.
%  It seems that the image is sharper and doesn't vary
%  of its own accord very much.
%  This means we can use less preprocessing algorithms.
%  If the resolution is increased past 160x120, the speed
%  is impeded greatly.  To increase the speed I can shrink the
%  matrices to be compared.
%
```

```
%       Larger hausdorff matching criterion are needed to correctly
%       track objects in higher resolutions.
%
%  Logitech cameras need a threshold of around 15.
%
%  The Sony Handycam works well at any threshold value from 8-20.
%
```

A.2 Background Subtraction and Preprocessing

This function subtracts the background and thresholds an image. Input variables A and B represent the current frame and the background, respectively. The output variable *retu* is a binary image of the same x and y dimension of A.

```
function [retu] = backnoise(A, B)

a = rgb2gray(abs(double(A) - double(B)));
a = a - thresh;
a = ceil(a / 255);
a = bwmorph(a, 'erode', 2);
a1 = bwmorph(a, 'clean');
a2 = bwfill(a1, 'holes', 8);
a3 = bwmorph(a2, 'thicken',1);
a3 = bwmorph(a3, 'dilate', 2);
retu = a3
```

A.3 Object Detection and Isolation

This procedure takes as input the binary image resulting from background subtraction and preprocessing. The return variables are:

- **objects:** This is a cell array of matrices. It contains as many matrices as there are objects in the scene
- **act:** This is the number of objects detected in the scene.

- **x** and **y**: These are the offsets into the image of each object. They are one-dimensional vectors of length **act**.

```
function [objects, act, x, y] = ObjectIsol(A)
[S N] = bwlabel(A, 4);
objs = {0};
actualn = 0;
x = 0;
y = 0;
%
% Find each object and create a matrix
% containing it alone.
% Also store its offset into the image
for count = 1:N
  [a, b, c] = find(S == count);
  xx(count) = min(a);
  yy(count) = min(b);
  a=a-xx(count)+1;
  b=b-yy(count)+1;
  c=ones(size(c));
  SIL1 = sparse(a,b,c);
  SIL1 = full(SIL1);
  if ((size(SIL1, 1) * size(SIL1, 2)) > 64)
    actualn = actualn + 1;
    objs(actualn) = {SIL1};
    xof(actualn) = xx(count);
    yof(actualn) = yy(count);
  end
end
objects = objs;
act = actualn;
x = xof;
y = yof;
end
```

A.4 Tracking Algorithm

This consists of three sections of code. The first is the shell of the algorithm, maintaining an interface with the control loop; receiving variables and returning data from the present scene. The second section sends data to the Hausdorff algorithm. The third section performs the Hausdorff distance calculation. The third section is C code, interfacing with Matlab by being compiled into a DLL file.

A.4.1 Tracking Shell

This procedure converts the cell array of objects, input variable A, into edge sets and compares each object with each object of the previous scene. The edge sets of the objects from the previous scene are the variables $list2x$ and $list2y$. Input variables $offx$ and $offy$ are vectors containing each object's offset in the image. These are need to correctly annotate the frame. The variable $chars$ contains the labelling of objects in the previous scene.

After the comparison matrix has been made, object matching occurs. Proceeding this is the section of code which annotates the frame.

```
function [lx, ly, c] = tracks(A, list2x, list2y, offx, offy, chars)

global hcut;
global chile;
global posntype;
list1x = {};
list1y = {};
rect(1, :) = [0 0 0 0];
for a = 1:size(A, 2)
   [c b] = find(bwperim(A{a}));
   rect(a, :) = [offy(a) offx(a) max(b)+offy(a) max(c)+offx(a)];
   % This normalizes the edgesets to the same size
   % I'm not implementing this as it (obviously)
   % removes size information
   %
   %   list1x(a) = {c*100 / rect(a, 4)};
```

```
   %   list1y(a) = {b*100 / rect(a, 3)};
   list1x(a) = {c};
   list1y(a) = {b};
   end
end
for a = 1:size(list1x, 2)
      matched(a) = 0;
end
clmat(1,1) = 0;
%
% Create the matching matrix
% Haus2 is the Hausdorff Comparison algorithm
%
for a = 1:size(A, 2)
   for b = 1:size(list2x, 2)
      if (size(list1x{a}, 1) > 0) & (size(list2x{b},1) > 0 )
         l = haus2(list1x{a}, list1y{a}, list2x{b}, list2y{b));
      else
         l = 500000;
      end
      clmat(a,b) = l;
   end
end
coun = 0;
%
% Find the best matches for each object
%
for a=1:size(A,2)
   [mm, i] = sort(clmat(a, :));
   b = 1;
   h(a) = 0;
   while ismember(i(b), matched) == 1
      b = b + 1;
      if b > size(i,2)
         h(a) = 500;
         ch(a) = '?';
         break;
      end
```

```
    end
    if h(a)<500
      matched(a) = i(b);
      if chars(matched(a)) == '?'
        ch(a) = char(max(chars)+1);
      else
        ch(a) = chars(matched(a));
      end
      h(a) = mm(b);
    end
  end
outstring = ' ';
outhaus = ' ';
%
% Annotate the frame displayed to the user
%
for a=1:size(A,2)
  figure(1)
  hold on;
  if h(a) < (hcut)
    if (size(list1x{a}, 1) > 0)
      myrect(rect(a, :), 'g');
      text(offy(a), offx(a), ch(a), 'Color', 'y');
      plot(rect(a,1) + ((rect(a,3)-rect(a,1))/2),rect(a, 2)
            + ((rect(a,4)-rect(a,2))/2), 'bp');
     st = sprintf('Object %c is recognised. Center: %d, %d\n',ch(a),
        rect(a,1) + ((rect(a,3)-rect(a,1))/2),rect(a, 2)
        + ((rect(a,4)-rect(a,2))/2));
      outstring = strcat(outstring, st);
     st = sprintf('Object %c is recognised.  Haus: %d\n',ch(a),h(a));
      outhaus = strcat(outhaus, st);
    end
  else
    if (size(list1x{a}, 1) > 0)
      if h(a) == 500
        myrect(rect(a, :), 'b');
        text(offy(a), offx(a), '???', 'Color', 'y');
        st = sprintf('Object %d unknown\n',a);
```

```
        outstring = strcat(outstring, st);
      else
        ch(a) = max(ch)+1;
        myrect(rect(a, :), 'r');
        text(offy(a), offx(a), ch(a), 'Color', 'y');
        plot(rect(a,1) + ((rect(a,3)-rect(a,1))/2), rect(a, 2) +
                  ((rect(a,4)-rect(a,2))/2), 'rp');
        st = sprintf('Object %c is not recognised.
                  Center: %d, %d\n',ch(a), rect(a,1) +
                  ((rect(a,3)-rect(a,1))/2), rect(a, 2) +
                   ((rect(a,4)-rect(a,2))/2));
        outstring = strcat(outstring, st);
        st = sprintf('Object %c is not recognised.
                   Haus: %d\n',ch(a),h(a));
        outhaus = strcat(outhaus, st);
      end
    end
  end
  hold off;
end
drawnow;
lx = list1x;
ly = list1y;
c = ch;
%
% Report on the positions of objects
% Or their lowest Hausdorff distances
% If required
%
if posntype == 1
   set(chile(11),'String', outstring);
elseif posntype == 2
   set(chile(11),'String', outhaus);
end
```

A.4.2 Hausdorff Shell

This section sets up data and sends it to the DLL file. Included (but commented out) is my original approximation to the Hausdorff distance. The input to this procedure is two pairs of edgesets. The output variable is the calculated Hausdorff distance.

```
function maxd = haus2(edgesetx, edgesety, edgesetx2, edgesety2)
size2 = size(edgesetx2, 1);
size3 = size(edgesetx, 1);
if size3 > size2
   ex = edgesetx;
   ey = edgesety;
   edgesetx = edgesetx2;
   edgesety = edgesety2;
   edgesetx2 = ex;
   edgesety2 = ey;
end
%
% Call the Hausdorff dll I created
%
maxd = haus(edgesetx, edgesety, edgesetx2, edgesety2);
end
%
% The next section is the code to approximate the
% Hausdorff distance
%
%if type == 2
%   count = 0;
%        num = 1.4;
%   t1 = [edgesetx, edgesety];
%   t2 = [edgesetx2, edgesety2];
%   true1 = t1;
%   true2 = t2;
%   [a] = t1;
%   sizlim = size(t1, 1) * 0.01;
%        iterat = 30;
%   count = 0;
```

```
%   while (size(a,1) > sizlim) & (count<iterat)
%     count = count+1;
%     t1 = round(t1 / num);
%     t2 = round(t2 / num);
%     if mod(count, 2) == 0
%     [a, i] = setdiff(t1, t2, 'rows');
%     t1 = t1(i,:);
%     else
%     [a, i] = setdiff(t2, t1, 'rows');
%     t2 = t2(i, :);
%     end
%   end
%   maxd = (num*(count-1));
%end
```

A.4.3 Hausdorff C Code

This is an implementation of equations 3.1 and 3.2. The input variables are four one-dimensional vectors; or two pairs of x and y dimensions of edge-sets.

```
/* $Revision: 1.2 $ */
/*
 * HAUS.C      .MEX file implementing the
 * Hausdorff distance algortihm
 *
 * The calling syntax is:
 *
 *              [dist] = haus(Ax, Ay, Bx, By)
 *
 *
 */

#include <math.h>
#include "mex.h"

/* Input Arguments */
```

```
#define Ax_IN    prhs[0]
#define Ay_IN    prhs[1]
#define Bx_IN    prhs[2]
#define By_IN    prhs[3]


/* Output Argument */

#define H_OUT    plhs[0]

static void hau(double hp[], double ax[], double ay[],
                    double bx[], double by[],
                    unsigned int m1, unsigned int m2)
{
  unsigned int c1, c2;
  unsigned int pdis, maxd, tempd;
  signed int tx, ty;

  maxd = 0;
  for (c1=0; c1<m1; c1++) {
    tempd = 9999;
    for (c2=0; c2<m2; c2++) {
          tx=(ax[c1]-bx[c2]);
          ty=(ay[c1] - by[c2]);
      pdis = tx*tx + ty*ty;
      if (pdis <= maxd) {
        tempd = 0;
        c2 = m2;
      }
      if (pdis < tempd)
        tempd = pdis;
    }
    if (maxd < tempd)
      maxd = tempd;
  }
  hp[0] = sqrt(maxd);
  return;
}
```

```
void mexFunction(
                int nlhs,       mxArray *plhs[],
                int nrhs, const mxArray *prhs[]
                )
{
  double        *hp;
  double        *axp,*bxp;
  double        *ayp,*byp;

  unsigned int m1, m2;

 /* Get the size of the edge-sets given */

  m1 = mxGetM(Ax_IN);
  m2 = mxGetM(Bx_IN);

  /* Create a matrix for the return argument */

  H_OUT = mxCreateDoubleMatrix(1, 1, mxREAL);

  /* Assign pointers to the various parameters */

  hp = mxGetPr(H_OUT);
  axp = mxGetPr(Ax_IN);
  bxp = mxGetPr(Bx_IN);
  ayp = mxGetPr(Ay_IN);
  byp = mxGetPr(By_IN);

  /* Do the actual computation */

  hau(hp,axp,ayp, bxp, byp, m1, m2);
  return;
}
```

A.5 Histogram Matching

Most of the preparation for the Histogram matching procedure is performed in the control loop. It is yet to be modularised. This code performs a single template assessment. The return value *val* is the average mean absolute difference over red, green and blue comparisons. The input variables are:

- **R, G, B:** These are the respective histograms of the template we are searching for.
- **BB**: This is a matrix of the same size as the original template, with 3 color dimensions.

```
function val = rph(R, G, B, BB)
r=imhist(BB(:, :, 1),64);
g=imhist(BB(:, :, 2),64);
b=imhist(BB(:, :, 3),64);

val = (mean(abs(r - R)) + mean(abs(g - G))
    + mean(abs(b - B)))/3;
```

Appendix B

Test-Set Output Results

The following twelve frames were selected from a series of 50 to illustrate the ability of this system to track multiple objects. They were originally rendered as a test set for my system. Three objects are tracked, two points are located in each frame. One point was chosen as the middle of the sphere, The other as the middle of the bipedal's head.

We can see from Figures B.1 and B.2 that the system tracks these objects. This data set is free of environmental lighting noise, the preprocessing functions need only account for errors introduced by the background subtraction process.

Point tracking in this data set obviously enjoyed mixed success. The point located on the head of the bipedal figure was located through all frames. The point on the sphere, however, moved quite sporadically over it's lit surface. Obviously this is due to the similarity in color of the large portion of the visible sphere. The weighted mean, indicated by the cyan dot, of the trialed positions for this point remained relatively constant, however.

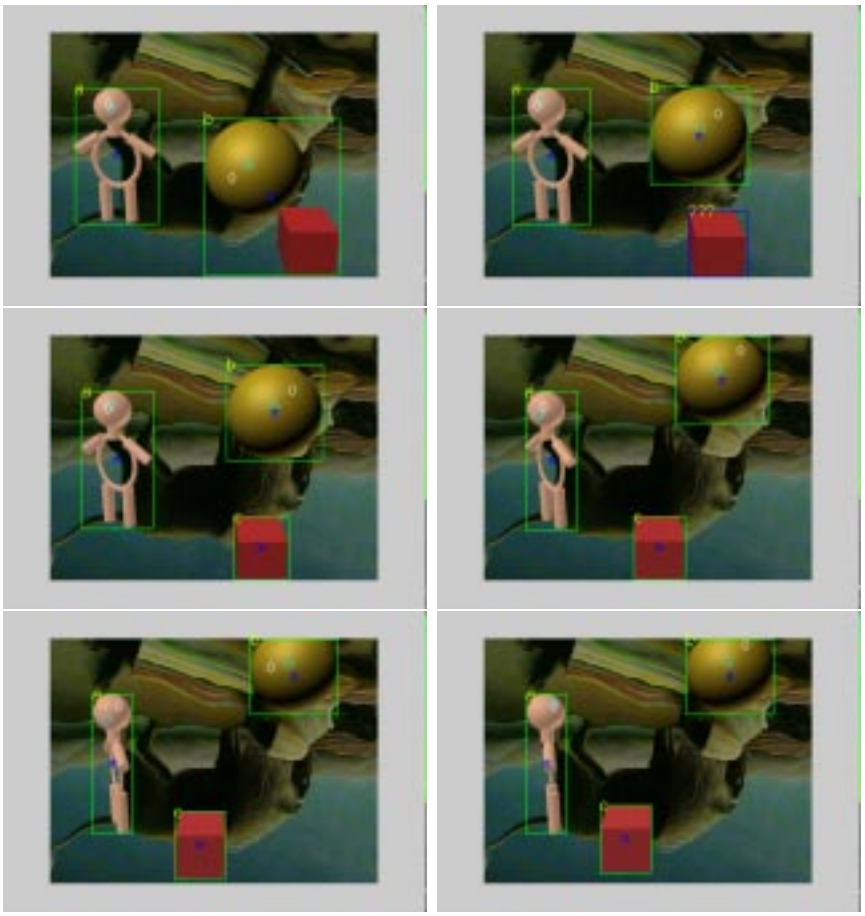


Figure B.1: The first six selected frames

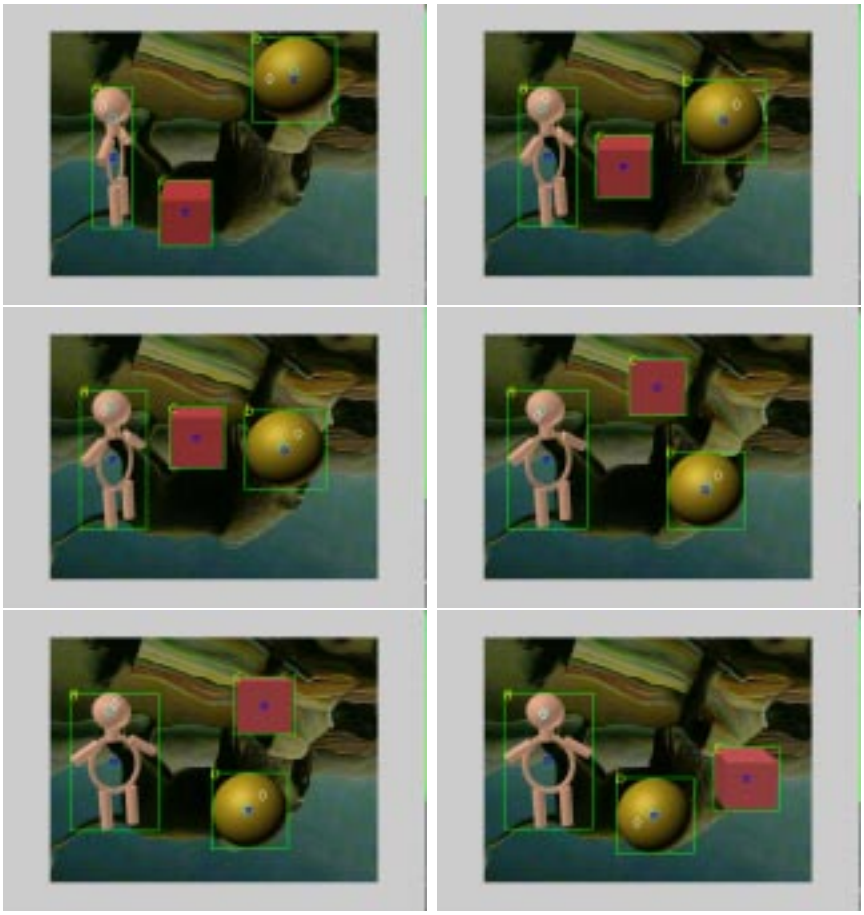


Figure B.2: The last six selected frames