Project

Peter Juritz

November 1, 2010

Contents

List of Figures

Abstract

Table top and surface computing systems have grown in popularity of recent years. However, many of these systems are prohibitively expensive and require a complex architecture often taking up an entire room.

In this report we detail the design and implementation of a vision based interaction which requires only a web cam and a computer monitor. A detailed overview of current literature pertaining to the subject is given. A system design is developed which takes the various goals and constraints of the project into account. A software implementation was developed following this design. Detailed analysis is given on the implementation of the system. This includes discussion on the methods used as well as the constraints of the problem.

Experiments are performed to measure the performance and accuracy of the system. Results show real time performance and satisfactory accuracy. Various limitations of the current technique are analysed which suggest possible improvements to the current method.

Chapter 1

Introduction

There has been a great deal of research into vision based computer interfaces. Surface and table top computing has increased in popularity greatly in recent years. While there are many benefits to such systems, the architecture involved is fairly complex and expensive. A typical surface computing system consists of a special surface for projection, a digital projector and optical and infrared cameras. Furthermore these systems often take up an entire room.
In this report we present the design and development of a vision based interaction system which requires only a computer monitor and a web cam. Our system uses a head mounted web cam pointed at the computer monitor to allow a user to interact with system by using only their hands. Specifically, the users interacts by occluding the monitor with their hand. The goal of this is that the hand should seem to become seamlessly integrated with on-screen elements. Actions taken by the hand are reflected on-screen allowing direct manipulation interactions.

1.1 Motivations

Our design and implementation is motivated by the following factors.

1.1.1 Vision based interaction

Vision based interaction systems provide a novel method for interaction with a computer. Users are able to control a system in a fashion which more closely matches the way we interact with the physical environment. In conventional mouse and keyboard system there is a boundary between input and output. Input happens with the physical mouse and keyboard and the output is seen on the computer monitor. No such boundary exists in a vision based system. This has the effect of breaking the barrier between the system and the physical environment.
Since we use a head mounted web cam, our system matches the users perceptions to a higher degree than conventional tabletop systems. The position where the user sees their hand in relation to the monitor will match the computed position very closely due to the small parallax between the eye and the camera.

1.1.2 Expense and simplicity

Another motivation for the design of this system is the expense. This system is relatively cheap to recreate. The only additional piece of hardware required is a web cam. When compared to the expense and complexity of a conventional table top system, the value of such a project

becomes clear.

1.1.3 Additional Information

In some regards vision based interaction systems can be thought of as a superset of touch based interfaces. While a touch interface only knows the positions of the fingertips, a vision system can have knowledge of the entire hand configuration in three dimensions. This allows for interaction beyond the level of a pointing device. Hand configuration can be used contextually for almost countless purposes.
A further motivation for the design of our system is that it can produce more information that conventional table top interfaces. The fact that the camera is head mounted allows the system to compute the users head orientation with regard to the monitor. This information can be used to add three dimensional spatial information to a program context. For example in a first person simulation a user can look around a scene with their head or even peak around a corner.

1.1.4 Research Value

Finally, there is currently no research on the design of such a system. For this reason it is possible to explore a new area. Valuable insights can be gained from exploring this area, no matter what the results are. Analysis can be performed to observe how well current techniques can be adapted to this purpose.New techniques may need to be developed in order to solve the problem. Such research is valuable in itself.

1.2 Outline

To develop this system a thorough review of current techniques was performed. This is detailed in the background chapter. The design process, design goals, constraints and conclusions are discussed in the design chapter.
A software package was implemented following this design. Specific details about the software implementation, problems and solutions are discussed in the implementation chapter.
Finally the system was tested and analysed. The findings of these experiments can be found in the results chapter as well as conclusions on the effectiveness of various design and implementation choices. These results suggest further work which can be performed in the area.

Chapter 2

Background

In this chapter we review the research and techniques developed in the following areas: object tracking, homography mapping, pose estimation, hand pose estimation, and direct manipulation and tangible interfaces.
An introduction to object tracking is given and the role that object representation and image features play in object tracking is explained. Edge detection schemes and the Hough transform is discussed as well as an overview of Moore's contour tracing algorithm. This is followed by a description of the popular techniques in background subtraction and silhouette tracking.
A brief overview and wide overview of popular techniques in hand pose estimation is given. This includes a brief introduction to homography mapping. More specifically we concentrate on model based pose estimation and single frame pose estimation. The differences and benefits of these topics are discussed.
Finally a brief description is given of tangible based interfaces, difficulties in their implementation and their benefits.
These topics are integral to the development of a vision based interaction system. It is necessary to know which results have been obtained and which techniques have been developed in order to build such a system.

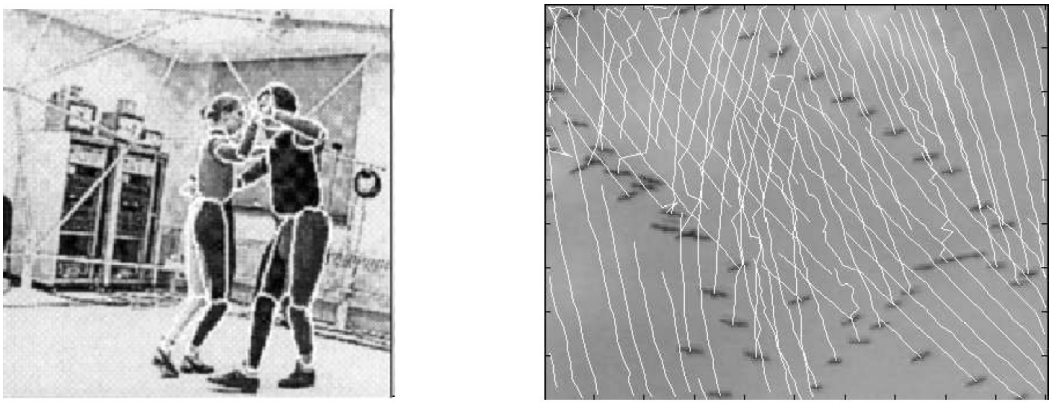
2.1 Object Tracking

Object tracking is the task of detecting the position of an object in a scene consisting of single image or sequence of frames[Yilmaz et al. 2006]. There are many techniques which can be applied to solve this problem, however, each techniques is often best applied to a specific domain. Yilmaz et. al. suggest that the techniques can be categorised based on the following factors: the object representation used, the image features used and the method of modelling motion, appearance and shape[Yilmaz et al. 2006].

2.1.1 Difficulties

There are many difficulties involved in object tracking. Yilmaz et al identify the following common obstacles to object tracking[Yilmaz et al. 2006].

- loss of information caused by projection of the 3D world on a 2D image
- noise in images
- complex shapes and motion



(a) (b)

Figure 2.1: (a) Human tracking segmentation from Gavrila and Davis [Gavrila and Davis 1996](b) Bird tracking from Shafique and Shah[Shafique and Shah 2005].

- nonrigid or articulated nature of objects
- partial and full object occlusions
- scene illumination changes
- real-time processing requirements

Many of these problems are addressed by constraining the search space and parameters during evaluation. For example, assumptions will be made about an objects position in frame T from its detected position in frame $T-1$. Constraints can also be applied to features of an object with articulated geometry, more on this will be discussed in later sections.

2.1.2 Object representation and Image Features

In order to track an object through a sequence of frames it is first necessary to detect the object. This process depends on two factors. Tracking algorithms must use the most applicable object representation and from this choose the best image features to identify the object by.

2.1.3 Object Representation

Object representation refers to how the target object is modelled in the scene. This representation can range from a single point to a complex composite of shapes as illustrated in figure 2.2. Object representation must be chosen to best match the target object and required information. Figure 2.1 illustrates how the requirement of the application affects the object representation. Shafique and Shah use a point representation to track the position of birds while Gavrilla and Davis use a composite of edge boundaries to track and analyse movements of people [Shafique and Shah 2005, Gavrila and Davis 1996].
Figure 2.2 shows different object representations derived for a single image.

2.1.4 Image features

Image features are components of an image or sequence of frames which gives rise to its structure. Common image features are colour, edges, optical flow, and texture.

Colour

Colour is the most widely used image feature used in object tracking although it is sensitive to illumination.[Yilmaz et al. 2006]. Colour data can be represented in a different number of ways. Most commonly the RGB colour space is used however it does not handle illumination changes well[Yilmaz et al. 2006]. Human perceptual differences in colour are known to differ from changes in RGB and hence RGB is not perceptually uniform[Paschos 2001].The $L*u*v*$, $L*a*b*$ and HSV (Hue, Saturation, Value) colour spaces have been shown to handle illumination more effectively although they are sensitive to noise[Song et al. 1996].

Edges

Edges are sections of images where there is a large transition in image intensity[Yilmaz et al. 2006]. An important feature of edges is that their detection is robust to changes in illumination. Edges are often used when the boundary of an object is being detected[Yilmaz et al. 2006]. The most commonly used edge detection algorithm is the Canny edge detector[Canny 1986].

2.1.5 Edge detection

An integral part of most edge detection schemes is the Sobel filter. Images are convolved with this filter to produce another image which contains information about the rate of change within the image. The Sobel operator is usually implemented as two 3x3 filters which represent differentials in the x and y directions. Edges can be found by thresholding the values produced by the convolution.

The Canny Edge detector is a more advanced technique for edge detection. This step initially blurs the image with a Gaussian filter. During edge detection, only the strongest edges in each neighborhood are attained. This is accomplished through *non maximal suppression*

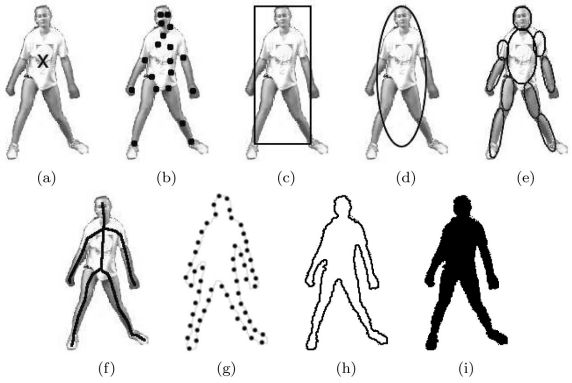


Figure 2.2: Object representations. (a) Centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object skeleton, (g) complete object contour, (h) control points on object contour, (i) object silhouette. Taken from Yilmaz et al.

wherein edges are classified by their orientation and required to have the maximum edge value in this direction. Edge locations are detected by means of hysteresis.

Texture

Texture describes the intensity variation of a surface and represents features such as smoothness or roughness[Yilmaz et al. 2006]. Image texture has also been shown to be less sensitive to illumination changes than colour[Yilmaz et al. 2006]. The most common texture descriptors are: Gray-Level Cooccurence Matrices [Dinstein et al. 1977], Law's texture meshes[Laws 1980], wavelets[Mallat 1989] and steerable pyramids[Greenspan et al. 1994].

2.1.6 The Standard Hough Transform

The standard Hough transform [Duda and Hart 1972] was first developed to detect line segments in an image . This technique relies on an image's edge features to determine the likely positions of line segments within an image.

In its initial formulation lines were modelled by their m and c parameters in the implicit $y = mx + c$ line form. An *accumulation buffer* in the m, c parameter space is used by the transform to record the likely parameters of lines within the image. One problem with the m, c parameter formulation is that this case is unable to deal with vertical lines as well as leading to a large accumulation buffer. To address this the r, θ formulation wherein a line is parameterised by r, the distance from the origin to the closest point on the line, and θ the angle of the vector to from the origin this point. Using this formulation a line can be written as

$$y = \left(-\frac{cos\theta}{sin\theta}\right) x + \left(\frac{r}{sin\theta}\right)$$

The algorithm works by *voting* for the parameters of all the possible which pass through each edge pixel using the accumulation buffer. Specifically the algorithm increments the accumulation value in the buffer at position (r_i, θ_i) for each line with parameters r_i and θ_i which pass through a given point.

Hence given an edge pixel at the point (x_0, y_0) the system *votes*, or increments accumulation values, along the sinusoidal curve $r(\theta) = x_0 cos\theta + y_0 sin\theta$ for $\theta \in [0, \pi]$.

When all votes have been performed for every edge pixel the system searches for local maxima in the accumulation buffer to find the probable line parameters.

One optimisation to this voting scheme was introduced. Instead of voting for every line, or every θ value in the range $[0, \pi]$, this optimisation makes use of gradient information produces in the edge detection step to estimate the orientation of the line. Votes are then cast in a range around this θ value, thus significantly reducing the number of votes cast. This leads to increased performance as well as a 'cleaner' accumulation buffer.

2.1.7 Real Time Hough Transform

There are various limitations to the standard Hough transform. Firstly, the voting process is relatively computationally expensive and hence the standard transform can not achieve real time processing. Secondly, the brute force nature of the voting scheme leads to a 'messy' accumulation buffer which can make finding local maxima difficult. This can also sometimes lead to the failure to detect weaker line due to the large number of votes cast.

To address these problems Fernandes and Oliveira introduced an improved kernel based voting scheme[Fernandes and Oliveira 2008]. Their technique operates on clusters of approximately colinear pixels. Votes are then cast in an oriented elliptical-Gaussian kernel which models the uncertainty associated with the best fitting line of the cluster.

By using this method they are able to achieve real time frame rates on relatively large images.

2.1.8 Contour Tracing

Contour tracing is the problem of tracing the pixels which lie on the boundary of a segmented region in an image. This problem has been studied in the context of graphs, as opposed to images, since the late 19th century. Here we present a brief mention of a few algorithms with a focus on Moore's tracing algorithm. The square tracing algorithm attempts to trace the contour of a region by using a simple movement method. The algorithm, in its simplest form, works by starting at an edge pixel, and moving in a given direction. When currently on a region pixel the direction of movement is turned right, when a pixel off the region is occurred the direction of movement is changed left. When the original pixel is encountered the algorithm terminates.

The square tracing algorithm has been shown to fail on regions which are not 4-connected. Hence this method can not be used for 8-connected regions which are occur commonly in computer vision and image processing.

A more advanced method is Moore's tracing algorithm. This algorithm uses backtracking and Moore neighborhoods to trace a region contour. The Moore neighborhood of a pixel p is the set of 8 pixels surrounding p. More generally the Moore distance of a vertex v in a graph is the set of vertices which have a distance of 1 from v.

The algorithm initially sets the first boundary pixel to be the current pixel. At each step the algorithm backtracks to the pixel from which it entered the current pixel. The Moore neighborhood of this new pixel is then walked in a clockwise fashion until a new boundary pixel is encountered upon which time this pixel is set to the current pixel. The process is then repeated until the original pixel is met.

However, in certain cases this stopping condition can fail and the entire contour will not be traced. Jacob Eliosoff suggested a stopping condition which is able to handle these cases. Jacob's stopping criterion requires that the original pixel must be entered from the same pixel from which it was originally entered in order for the algorithm terminate.

With Jacob's stopping criterion the Moore tracing algorithm is guaranteed to trace the outer contour of any 8-connected region. However some contours will contain holes inside them. In these cases these internal contours will not be traced. A search for holes can be made once the outer contour has been traced and the algorithm can be repeated. Other more algorithms exist which are able to trace internal contours without additional searches however these are out of the scope of this article.

2.1.9 Background Subtraction

An object can be tracked and detected by modelling the background of the scene and then finding deviations from this in each coming frame. This technique is called background subtraction. Many different methods have been used to achieve this. Difficulties include: changes in illumination, shadows, camera jitter and non-static backgrounds. A brief explanation of some of the most popular methods used for background subtraction is given.

To handle temporal changes in illumination Wren et. al. modelled the colour at each pixel $I(x, y)$ by a Gaussian function[Wren et al. 1997]. As new pixel data comes in over time the mean and variance of this Gaussian is updated. Incoming pixels are then tested against these Gaussian and marked as either background or foreground pixels. However, Gao et al showed that a single Gaussian does not perform well for outdoor scenes since it can not compensate for repetitive object motion, shadows and reflectance[Boult et al. 2000]. Stauffer and Grimson addressed this by using multiple Gaussian for each pixel[Stauffer and Grimson 2000]. If a match to one of the Gaussian is not found for an incoming pixel a new Gaussian is created with the mean set to the value of the new pixel.

More complex methods process more information than just colour values for each pixel. Elgammal and Davis have proposed a technique where the background pixels are modelled by a kernel density (Kernel refers to the shape and appearance of an object)[Elgammal et al. 2000]. Their method does not only look at a single pixel to classify an incoming pixel but its neighboring pixels as well. Their method can therefore handle camera jitter and small background movements.

Another approach to background subtraction is to model each pixel or block of pixels as being in a certain state - usually background or foreground. Ritcher et al use Hidden Markov Models to model the state and state transition of blocks of pixels to perform background subtraction[Rittscher et al. 2000].

Oliver et al. have suggested using eigenspace decomposition of a sequence of frames. Incoming images are then projected onto the eigenspace and foreground pixels are found by examining the differences.

Many of these techniques can not handle dynamic backgrounds well. Monnet et al, and Zhong and Sclaroff have proposed methods which handle time varying backgrounds by applying autoregressive moving average processes [Monnet et al. 2003, Zhong and Sclaroff 2003]. These processes can learn how an environment changes over time.

Modern methods use a combination of image features and attempt to compensate for dynamic backgrounds, changes in illumination and temporal effects. These methods are computationally efficient and can be run in real time[Yilmaz et al. 2006]. In practice these methods often lead to small errors which must compensated for in resulting algorithms. An important limitation is that many of these techniques require the camera to be static[Yilmaz et al. 2006]. This has been addressed by rebuilding the background every few frames or by attempting to model the changing projection of the camera. These techniques however are limited themselves as they make assumptions about camera movement and environment geometry[Yilmaz et al. 2006].

2.1.10 Silhouette Tracking

Silhouette tracking aims to determine an object region based on a computed object region from previous frames[Yilmaz et al. 2006]. This is necessary when the target object can not be represented by simple shapes such as a point or ellipse. Silhouette tracking algorithms are beneficial in that they can handle a wide variety of shapes[Yilmaz et al. 2006].

Yilmaz et al suggest that silhouette tracking algorithms can be divided into two categories: shape matching and contour evolution.

2.1.11 Shape Matching

Shape matching tracks a silhouette by searching the image for the object in every frame. The search is performed by computing the similarity of a proposed object silhouette with that of the

previous frame. The object is assumed to have only translated from the previous frame. This assumption means that from frame to frame an object must exhibit rigid body motion only. To handle this problem the object silhouette representation is updated at each frame which allows for non rigid body motion over time. Shape matching algorithms differ mainly in how they attempt to match the hypothesised silhouette to a candidate silhouette in the new frame. Here we will detail some techniques which have been employed for this purpose.

In order to compute the score of a silhouette match Huttenlotcher et al. use the Hausdorff metric on an edge map representation of the proposed position to measure how well the predicted position of the silhouette matches the input frame[Huttenlocher et al. 1993]. The Hausdorff metric measures the magnitude of differences between two sets of points.
Another method attempts to match two silhouettes detected in consecutive frames. Each individual silhouette is usually detected through background subtraction. Once these silhouettes have been detected, matching is performed by calculating a distance between the old silhouette and candidate silhouette[Yilmaz et al. 2006].
Kang et al. suggest using colour and edge histograms [**?**] built from circles covering the silhouette. These histograms are then matched to the proposed silhouette using three different measures: cross-correlation, Bhattacharya distance and Kullback-Leibler divergence.
Finally, Sato and Agger wal compute flow vectors of pixels within a silhouette and perform Hough transforms on pixel/time blocks to obtain a Temporal Spatio Velocity image for each frame. Their method is less sensitive to appearance variations than other methods [Yilmaz et al. 2006].

2.1.12 Contour Evolution

As opposed to shape matching methods which attempt to find a silhouette in each frame, contour evolution methods evolve an object contour from frame to frame given an initial silhouette. The object's contour consists of the boundary between the silhouette region and everything else. Contours are often represented by a set of control points along the object boundary.
Yilmaz et al divide contour evolution methods into two categories: tracking using state space models and tracking by energy minimisation.

State Space Models

State space models work by assigning a state to the parameters of the contour (such as shape, motion and control point position).
Terzopoulos and Szeliski assign state according to the dynamics of the control points which are modelled by a spring model[Terzopoulos and Szeliski 1993]. Control point positions for the next frame are computed from this. The new position is then matched and corrected with a trained particle filter.
Chen et al. use Hidden Markov Models on elliptical control points to predict and evolve the contour state.[Chen et al. 2001b].

Energy Minimisation

Energy models utilise an energy function on the contour and attempt to minimise this according to a set of parameters. Minimisation is usually performed through greedy methods or gradient descent[Yilmaz et al. 2006]. Contour energy is often defined by either temporal gradient or appearance statistics[Yilmaz et al. 2006].

2.1.13 Silhouette Representation

Silhouette are represented in different manners. They can be represented by binary indicator functions, which take the value one on the silhouette and zero elsewhere. Alternate representations are explicit and implicit surfaces. Explicit surfaces represent the silhouette by a set of control points along the object contour. Implicit surfaces represent the silhouette by a function define on the image.

2.2 Pose Estimation

Pose estimation refers to the problem of detecting the three dimensional articulation of an object from a single image or sequence of frames. In this article we will briefly mention some of the algorithms developed in this field.
Pose estimation algorithms can generally be divided into three categories: Algebraic solutions, optimising algorithms and hybrid algorithms. Algebraic solutions attempt to determine pose through closed form solutions. Optimising algorithms make guesses at a solution an iteratively improve these solutions to better match the observed results. Hybrid solutions attempt to use the positive aspects of both of these techniques.
Both algebraic and optimising techniques have their own benefits and weaknesses. Grembowietz identifies the following properties:
Algebraic solutions
Strengths:

- Fast

Weaknesses:

- Poor noise filtering
- Numeric instabilities

Optimising solutions
Strengths:

- numerically stable

Weaknesses:

- Dependence on initial guess
- Divergence

Most techniques rely on two key pieces of information. Firstly the system must have a model of the object which is being viewed. Secondly the system must have a method of computing correspondence points which lie on the projected object in the camera and on the object itself. The aim of pose estimation is to estimate the articulation of an object with respect to the camera. More specifically the rotation and translation matrices R and T of the object with respect to the camera coordinate system.

The POSIT algorithm

The POSIT algorithm is an iterative hybrid algorithm[Dementhon and Davis 1995] which requires an object model as well as four or more correspondence points. This algorithm initially determines a weak pose by assuming an orthographic camera projection. This solution is iteratively improved by shifting estimated correspondence points until the solution converges. It generally takes four iterations for this algorithm to converge.

Homographies

A homography is an invertible projective transformation which maps points from one plane to another and maps straight lines to straight lines. Homographies are used extensively in computer vision. Any two images of a planar surface are related by a 2d homography. They are applied to many problems from image rectification to augmented reality.

More formally a 2d homography is represented by a 3x3 matrix. Given two planes P_1 and P_2 and four or more corresponding pairs of points it is possible to determine a homography $H : P_1 \rightarrow P_2$ which maps points on P_1 to P_2. Hence a point $m = (u, v)$ on P_1 can be mapped to a point $m' = (x, y)$ by matrix multiplication by H. In order to do this m must be expressed in homogeneous coordinates as $m_H = (u, v, 1)^T$. Hence we have m' in homogeneous coordinates:

$$Hm_H = m'_H$$

Hand Pose Estimation

2.3 Hand Pose Estimation

Pose estimation refers to the problem of detecting the three dimensional articulation of an object from a single image or sequence of frames. Hand pose estimation attempts to compute this articulation for the human hand.
There are two main approaches to this problem. The first, called model based tracking, attempt to model the kinematics and dynamics of a three dimensional model of the object. These techniques make use of temporal effects or time coherence and use data from previous frames to predict the new hand position.
The second type of method, called single frame pose estimation, attempts to compute hand pose independently in each frame. These techniques ignore temporal coherence. This can be partly justified due to how rapidly hand silhouettes can change during hand movement which makes temporal effects useless[Erol et al. 2007].

2.3.1 Common Difficulties

Hand pose estimation is, by the nature of the problem, a theoretically and computationally difficult task. There are many approaches to solving the problem with continued research into new techniques. There are alternative methods to pose estimation such as the use of sensor gloves, however here we will focus on vision based solutions.
Erol et al. have identified the following common difficulties encountered in the field.

- High Dimensional Problem: Due to hand articulation the problem has many degrees of freedom. Most studies use a hand model that has over 20 degrees of freedom[Erol et al. 2007].

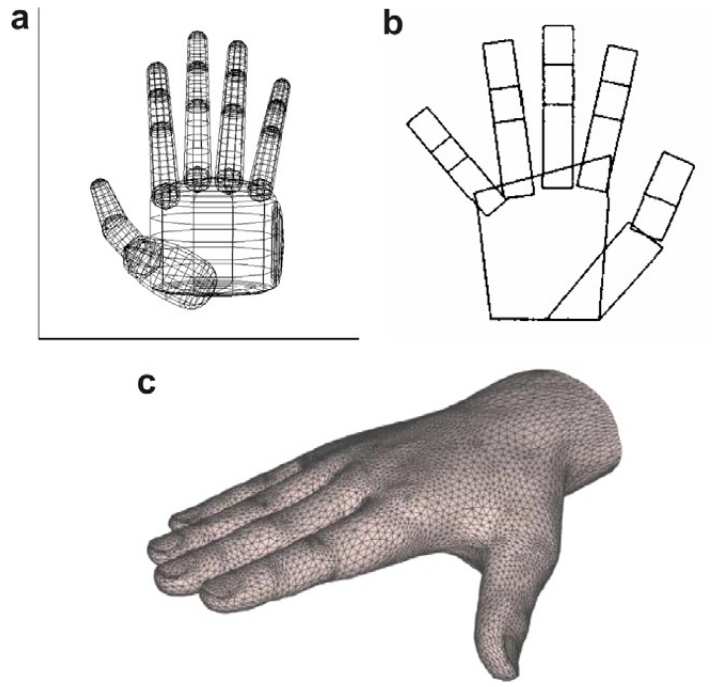


Figure 2.3: Hand models with varying complexity: (a) Quadrics based model from Stenger et al.(b) Cardboard model from Wu et al. (c) realistic model from Bray et al.

- Occlusions: Due to the concave nature of the hand, occlusions are regular. This presents a significant challenge to pose estimation algorithms as many parameters become unbounded during occlusions.
- Processing speed: Most of the uses for hand pose estimation require that the hand pose is computed in real time.
- Uncontrolled environments: To be a widespread and useful tool in HCI, hand pose estimation algorithms will need to cope with a wide variety of different environments.
- Rapid hand motion: During movement the hand can reach velocities of 5m/s and rotation speeds of 300 degrees/s [Erol et al. 2007]. This presents a problem due to hardware restrictions (refresh rates of sensors) and tracking algorithms. This can also make it impossible to use temporal coherence.

2.3.2 Model based tracking

Model based tracking algorithms attempt to calculate hand pose by use of a constantly evolving model of the current hand articulation. Various constraints and types of models are used in the field. Figure 2.3.2 shows the different complexities of models used for pose estimation.

Model based tracking can be further divided into two categories: single and multiple hypothesis tracking. Single hypothesis tracking maintains a single hypothesis about hand articulation throughout the tracking. While this method is more simple, it is susceptible to imperfections in the error function, background clutter, self-occlusion and complex motion[Erol et al. 2007].

Multiple Hypothesis methods keep multiple pose estimates at each frame. If the best estimate fails the system will continue tracking with another estimate[Erol et al. 2007].

Single Hypothesis

There are various approaches within single hypotheses pose estimation. Here we briefly discuss some of these methods.

- Optimisation methods: These methods make use of standard optimisation techniques. The optimisation criteria or fitness function is usually based on an error metric. Visual deviations from hypothesised pose and observed pose help guide the algorithm to a solution. Optimisation techniques which have been used include:
  the standard Gauss-Newton method[Rehg and Kanade 1993], Genetic Algorithms and Simulated Annealing [Nirei et al. 1996], Stochastic gradient descent [Bray et al. 2004] and divide and conquer algorithms[Wu and Huang 1999].

- Physical Force Models: These methods compute a pose and then use the match error to derive some physical forces. These physical forces are then applied to the hand model and dynamics simulations produce a new hand articulation. These techniques have been used in various different implementations [Lu et al. 2003, Delamarre and Faugeras 2001].

Multiple Hypothesis

Multiple hypothesis methods have been shown to have better resilience to error than single hypothesis methods[Erol et al. 2007]. Here we briefly discuss the different approaches to multiple hypothesis post estimation.
Techniques include particle filters, tree based filters (where a tree of hand pose images is stored and traversed), Bayesian networks, template database searching (multiple hypotheses are stored and their neighborhoods are searched at each frame).

2.3.3 Single frame pose estimation

Single frame pose estimation is a more difficult task than Model based tracking since no previous information can be used. However, this method is attractive since one does not need complex data structures and algorithms to store previous hand pose estimates[Erol et al. 2007]. We will briefly describe some of the single frame pose estimation methods.

- Object detection: Large datasets of artificially generated hand silhouettes are created and stored in a tree data structure which corresponds to "clusters of similar hand poses" [Erol et al. 2007].

- Image databases: These techniques aim to develop algorithms to rapidly retrieve similar hand pose images from a database given an input image. If a match is found the corresponding pose is known.

- Direct mapping: These techniques train machine learning systems to directly map a pose silhouette from 2D to 3D.

- Inverse kinematics: Inverse kinematics aims to calculate joint angles given finger tip, joint and palm positions. Closed form solutions for joint angles have been developed for this purpose, although they operate under a constrained model[C.S. Chua 2000].

One restriction of these methods is that they often require markers on the fingers and palm[Erol et al. 2007].

2.4 Direct Manipulation and Tangible Interfaces

Direct manipulation systems allow users to interact with an interface in a natural manner. There has been increasing interest in tangible interfaces over the past few years. Many of these interfaces are designed as table interfaces.

2.4.1 Table Interfaces

Table interfaces are tangible interfaces where users can interact with on screen elements through direct manipulation. These systems usually consist of projector camera systems. An image is projected onto a surface. This image is also in view of a camera. The users are usually given devices which they can use to interact with the system. Examples of these include fiducial markers [Kaltenbrunner and Bencina 2007] and light projecting pens[Piazza and Fjeld 2007].

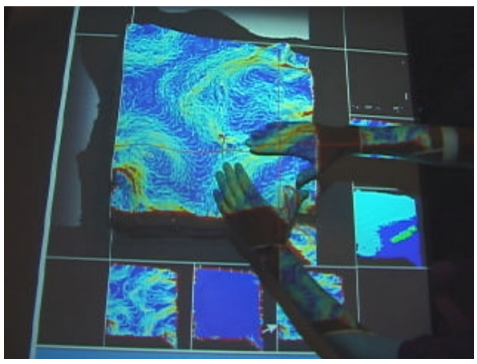


Figure 2.4: A landscape manipulation table top interface [Piper et al. 2002]

2.4.2 Difficulties

There are various difficulties involved in implementing a table based interface.

- Changes in illumination: Real environments often exhibit large changes in illumination. These include factors such as time of day and shadows created by users.
- Occlusion through interaction: While interacting with a table interface the users often occlude either the projector or camera system. To address this some system architectures use a translucent table top and place the camera below the surface.
- Projection distortion: The projector does not always face the table surface orthogonally. Furthermore some systems allow the users to tilt the projection surface[Song et al.].
- Dynamic or complex backgrounds: Changes in the background and background clutter can present significant problems to computer vision algorithms employed.
- Surface orientation: Some systems allow the table surface to be moved. The result of this is that the camera needs to correctly identify the position and orientation of this surface. Markers and fiducial are often used for surface detection and orientation.

2.4.3 Summary

Tangible interfaces allow for novel methods of human computer interaction. Piper et al. have implemented a tangible interface for landscape analysis[Piper et al. 2002]. Their system makes use of a projector and laser scanner which scans a table surface as can be seen in figure 4. The landscape is represented by a square of white Plasticine onto which a colour map is projected. Users are able to directly sculpt the landscape after which their changes are detected by the laser scanner.
Clearly tangible interfaces offer promising results for natural interaction. There are various approaches to implementing tangible interfaces, each having its own weaknesses and merits.

Chapter 3

Design

In this chapter we discuss the design considerations involved in the development of a vision based direct manipulation system. We analyse the various constraints involved in this implementation as well as what the goals of the system are. An overview of the basic system architecture is given followed by brief descriptions of individual components.

3.1 System description

The aim of this project is to design and develop a camera based interaction system. More specifically this system will consist of a head mounted web cam which faces the monitor. The user will then interact with the computer directly by using the hand. The fashion in which the hand occludes the monitor will control the system. Users will be able to use their fingers as a multi-point pointing device. The aim is to allow natural interaction with on-screen elements. This is to say that where the user perceives their hand to be should map onto correctly onto an on-screen position.
Further information can be computed from this configuration. The system will be able to compute the users head orientation in relation to the computer monitor hence creating 6 degree of freedom head tracking system. This data can be used to allow a user to look around an environment in a more natural fashion.

3.2 Design Considerations

Since the system and software package developed should be able to act as a useful input method to a program, there are some design considerations which must be taken into account. One of the key factors of an input system is that it responds quickly to commands. To achieve this the system must have a small latency in processing commands as well as a high rate of input. By its nature image processing is often a computationally expensive task. For this reason the design of this system is heavily influenced by the performance constraints imposed.

3.2.1 Sensor

Since the input technique being developed is spatially based, the rate of input is a key factor. The web cam used in the design of the system can capture 640x480 colour images at a rate of

30 frames per second. However, due to exposure times, this rate is often effectively lower.
The result of this is that in environments with sub optimal illumination the web cam takes longer exposure times to produce better images. However, this feature can be disabled leading to higher and more consistent frame rates. The lower illumination can then be dealt with through image processing.
It was decided that the minimum frame rate for the system should be 20 frames a second in order to provide useful and responsive input to a program. In order to achieve this many performance optimisations were made.

3.2.2 Tracking

The images produced from the web cam also often contain significant amounts of noise. Furthermore the environment can contain many noisy features which affect the detection of the desired features. For this reason, robustness is a key factor. The algorithms need to cope with many features which would otherwise be recognised as false positives and hence the system employs validation algorithms on computed solutions to decrease the possibility of this happening. A common technique which is used to deal with both performance requirements and noisy or partial solutions is tracking. A robust but slower algorithm can be used to initially detect the desired features and following this a faster algorithm can be used to track these features using spatial locality. However, during early testing it was discovered that due to the rapid movements involved in this system it is often ineffective to use tracking solutions. Furthermore, the tracking based solutions could easily latch on to an incorrect solution. For this reason no tracking was used and hence the algorithms developed have to be robust and perform efficiently. Hence the same process is used for each new video frame.

3.2.3 Limited information

The most important step in this system is the accurate detection of the monitor. This monitor can be positioned in many different orientations within a frame and can undergo motion due to camera movement. This can lead to partial occlusions from the monitor being either partially out of frame or occluded by a hand. Most tracking and detection algorithms attempt to use as many interest points on an object as possible for use in detection and tracking. However, since the object being detected is a computer monitor, naturally the majority of this object is completely dynamic. It is impossible to know anything about the appearance of the display section of the monitor other than a general lumminence threshold. No interest points can be detected on the monitor display and used in tracking. These facts have a large impact on the design of such an algorithm. Two specific features were observed which can be used to detect the monitor.

Another effect of this is that the hand will be positioned to occlude the monitor. In this case the system must must detect the hand against a completely unknown background. Furthermore this background is dynamic and can change rapidly making background subtraction nearly impossible.

3.2.4 Monitor Boundary Features

Firstly, regardless of what images are displaying on the monitor, the border between the active display area of the monitor and the monitor border is clearly visible. This creates strong edge

values along the four internal borders. We call these lines the internal border lines. Borders lines on the exterior of the monitor are also usually visible. We call these lines the exterior border lines. We call the region between these sets of lines the monitor boundary. We call the display part of the region inside the monitor boundary the monitor interior.

Secondly, the configuration of these lines is consistent. They form a quadrangle under any orientation. Hence the geometry of the appearance can be used to aid detection.

Lumminence Properties

Another property is that, even though we can not know the images being displayed on the monitor interior, this area s bright and has high lumminence values.

The monitor being used for the design of this system had a black border. This is commonly the case and hence we believe it is justifiable to use this information to aid detection. Therefore it is possible to use the dark border and its low lumminence values to aid in the detection step.

3.2.5 Selected Features

From this analysis it was decided that the internal boundary lines and the dark border would be used for detection since these were the only relatively invariant features of the monitor. Hence it is necessary to make use of image features such as edges and lines as well as thresholding on the lumminence values.

Goals:

- Accurate detection of monitor and pointers
- Real time performance
- Detection of commands

Constraints:

- Sensor quality and noise
- Sensor frame rate
- Environment illumination

Problems:

- Noise
- On-screen elements

3.3 Overview:

Here we present a brief overview of the design of the system. A detailed analysis of the design of each component is included in the following sections.
The overall goal of this system is to allow a user to interact with a piece of software by using only their hand and and a head mounted camera. This interaction is in the form of occlusion where the actions are made by using the hand to occlude the monitor. This occlusion can then be interpreted as an action such as a pointing device or a command. A further action which can be used is the position of the monitor in relation to the user.

To achieve this the system must first detect the monitor. This is accomplished by detecting the four line which compose the borders of the display. Once the monitor has been detected the system must then determine if there is a hand occluding the monitor in any place. If so the system must then determine contextual information from this set of occluding pixels. Pointer positions are calculated from detected finger tips. Commands can be interpreted from this data. Two command types exist. The first command is activated when a pointer stays in a certain position for a time period and is known as the hold action. The second command is activated when two finger points are brought together and is known as the pinch action. Lastly the monitor pose is computed.
This process forms the following set of data:
Monitor position (2d image space, 3d environment space) Occluding pixel set Finger tip positions Events (pinch, hold)
This set of information can then be used as a input to a program and thus allows a user to control it through this mechanism.

3.3.1 Feature Extraction:

The first step in the process of detecting a monitor is the detection of edge pixels. This is done through a standard process. To detect edges the system performs a convolution with a Sobel filter. This was chosen, above other more complex techniques, for its speed.
To detect the boundaries of the monitor a modified hough transform is used.
Tests showed that a standard hough transform could not maintain real time frame rates and hence modifications and optimisations were made which significantly increased the performance. These shall be discussed in detail in the implementation chapter.

3.3.2 Monitor detection:

The system detects that a monitor is present in a frame by examining detected lines as well as contextual information. A correct solution is a set of four lines which are in a quadrangular configuration which contain boundary pixels below a lumminence threshold. This threshold is set to match that of the monitor border.
The intersection points of these four lines give the four corners of the monitor.

3.3.3 Homography/Pose estimation:

The four detected corners of the monitor allow the system to compute the correct homography from the image space image corners to the display corners of the monitor. This allows the image space fingertip coordinates to map to the correct display coordinates.
Furthermore with an internal model representation of the monitor as well as these four image corners the system computes the rotation and translation of the monitor with respect to the camera.

3.3.4 Hand/Fingertip detection:

The hand is detected by using combined information from feature detection stage and the monitor detection stage. First the system detects a region of pixels which occlude the dark border of the monitor. A connected component fill is performed on this set in conjunction with edge

pixel data to obtain the full segmented hand region.
To obtain the fingertip locations the contour of this region is searched for peaks and valleys. Thresholding and suppression is performed on these peaks to obtain unique fingertip locations.

3.4 Test applications

One application will be developed for the purpose of development and testing of the system. This application is a simple drawing program wherein the user draws lines with the hand fingertips. The program will draw a dot at any fingertip position which is currently on screen. One of the benefits of a drawing applications is that the fingertip trajectory can be seen easily. This will allow us to evaluate the accuracy and efficiency of the system.
The second test application which will be used is a real time deformable terrain system. This package allows a user to navigate and deform a terrain environment in real time. One of the benefits of this system is that the on screen images change rapidly during movement and contain a large range in illumination differences. These aspects will be useful when compared to the simple drawing program which has static background with little illumination differences.

3.5 Summary

In this chapter the problem being solved was analysed. This analysis lead to design criteria for the solution of the problem. A brief overview of the design and purpose of all system components was given.
Finally, the applications used for testing were mention as well as their respective benefits.

Chapter 4

Implementation

In this chapter the details of implementation are discussed for each of the components of the system. For each stage of the process, a description of how the task is accomplished is given as well as reasons for these choices. Some brief discussion on the complexity of various algorithms is included with a more detailed discussion following in the results chapter.

4.1 Feature extraction

To detect the monitor and hand the system makes use of three basic features: edges, lumminence and lines. In later stages connected component analysis is used but this will be discussed in the monitor detection section.
For each frame the system determines the edge and line locations. This process must therefore be performed efficiently and accurately.

4.2 Edge detection

Edge detection is performed using by a convolution of the image with a 3x3 Sobel filter in the x and y directions. Only the lumminence channel is used for intensity, disregarding the colour components. At this stage no thresholding is used and the sum of the x and y gradients is stored for later use.

4.3 Lumminence thresholding

An optimisation was made to reduce the number of edges used in the hough transform. Since the border of the monitor used is black we can easily disregard many edge pixels which do not lie on the monitor boundary. This has the effect of improving performance and reducing the number of votes cast for lines which are not desired.

To achieve this we set a lumminence threshold which segments the black monitor border pixels from other pixels. From this we can classify a pixel as being either on (below the lumminence threshold) the monitor border or off (above the lumminence threshold) the monitor border. This terminology is used to describe pixels in later sections. During the convolution

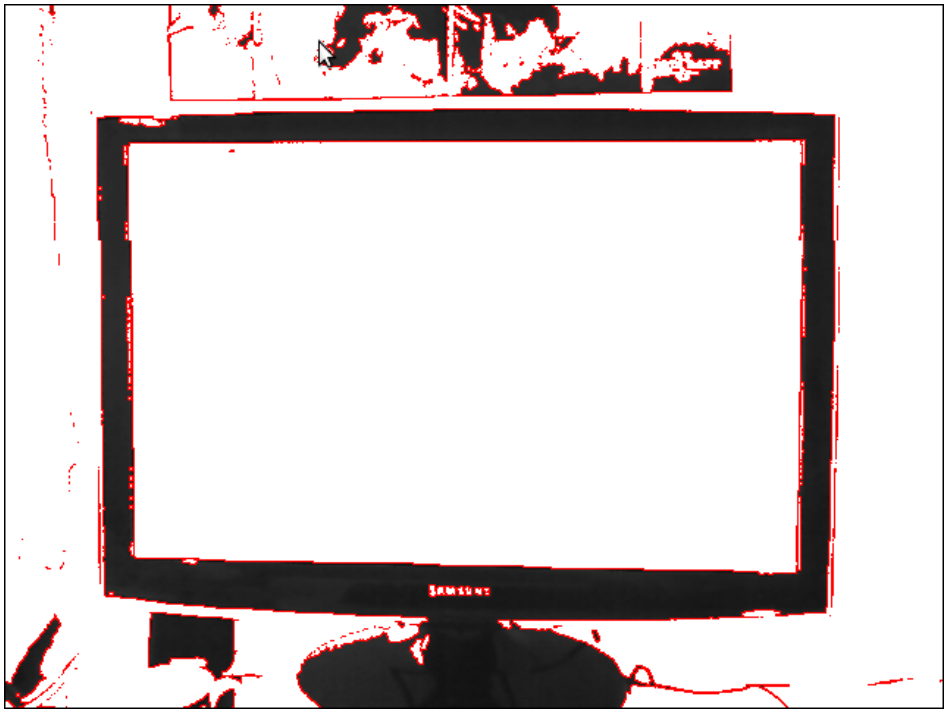


Figure 4.1: Graphic showing effects of lumminence thresholding. Dark pixels are below the lumminence threshold (off pixels). White pixels are pixels which fell above the lumminence threshold(on pixels). Red pixels are pixels which have neighbors which are both on and off.

with the 3x3 Sobel filter we check that the pixels used to compute each edge value have pixels which are both 'on' and 'off' the monitor boundary. Only edge pixels satisfying this condition are used in the hough transform. .
This lumminence threshold is also used in later steps to determine if a given pixel is on the dark monitor boundary as described in .

4.4 Hough transform

To detect the lines a standard hough transform is used with some modifications. The r resolution used is 1 and the angle resolution used is PI/180 radians or 1 degree. Alternative methods of line detection were investigated, such as the kernel based real time method[Fernandes and Oliveira 2008]. However, it was decided that the standard hough transform would be used with optimisations to attain real time frame rates.
To reduce the number of votes for each edge pixel the system uses the gradient information computed from the Sobel filter. Instead of voting for every theta value in the range 0,pi the system only votes in the range around the gradient direction computed from the Sobel filter.
To increase the performance further the system makes use of precomputed sin and cos tables. This optimisation was implemented to reduce the number of expensive sin and cos function calls. Another effect is that there is no need to convert integer array indices into floating point angles through floating point multiplication and division.

Once all edge pixels have been processed the accumulation buffer is searched for local maxima. Each of these maxima and their parameters are stored in an array. This array is then sorted by accumulation buffer value in descending order for reason which will be discussed in section 4.5.3.

4.5 Monitor detection

To detect the monitor the system must find a set of four lines which satisfy an angle constraint as well as a border constraint.

4.5.1 Quadrangle Detection:

The first constraint on a monitor solution is that it consists of four lines in a quadrangle. This quadrangle should consist of two pairs opposing of lines. The lines in each pair should be near parallel and the two pairs should be near perpendicular.

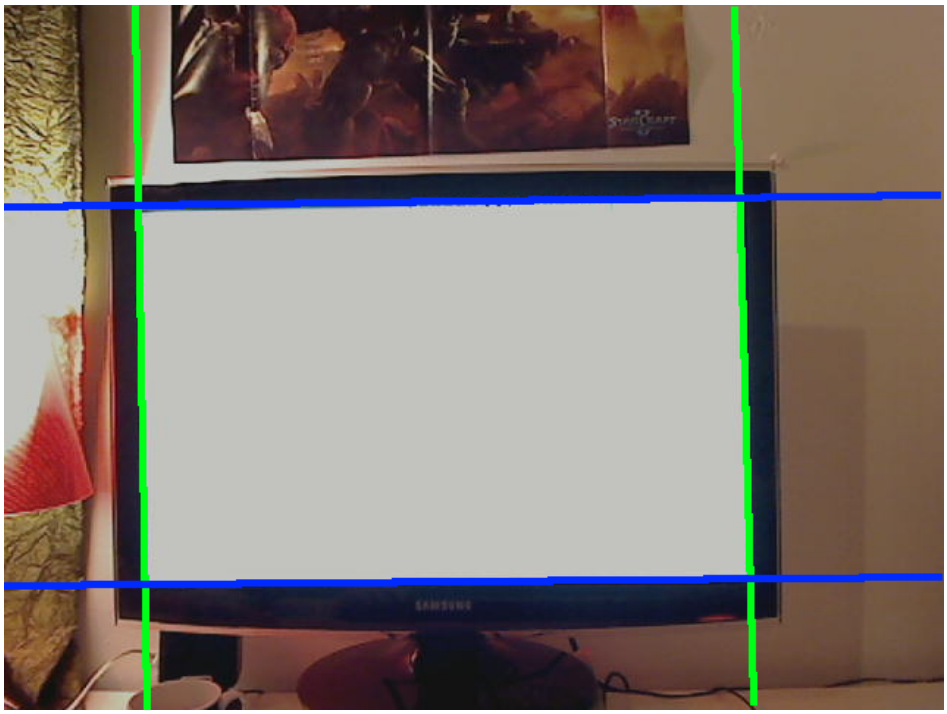


Figure 4.2: Examples of two pairs of lines. The blue lines form the parallel pair $P_0 = (l_i, l_j)$. The green lines form the parallel pair $P_1 = (l_n, l_m)$. These two pairs form a valid candidate quadrangle.

The first step is to find parallel pairs of lines. For this process the sorted list of lines from the feature extraction step is used. The system builds a second array of lines which consist of near parallel lines. This is done by computing the angle difference between each pair of lines. Two lines are considered to be parallel if their theta parameter difference is small . Further checks ensure that these parallel lines have a distance between them above a certain threshold. This reduces the number of parallel pairs of lines greatly. The justification for this optimisation

is that two lines which are very close can not form the opposite sides of a monitor.
The parameters of each the two lines are stored in an array.

Once this set of parallel pairs of lines is the system now finds perpendicular pairs. To do this the system now examines each pair of two of parallel lines. The theta parameters of each of these pairs are compared. If the angle difference between the two pairs is near $\pi/2$ then the two pairs are considered to be perpendicular. In summary, a set of lines which is accepted as a candidate quadrangle is structured as:

$$P_i = \{(\theta_{i_1}, r_{i_1}), ((\theta_{i_2}, r_{i_2})\} P_j = \{(\theta_{j_1}, r_{j_1}), ((\theta_{j_2}, r_{j_2})\}$$

$$\text{Quad} = (P_i, P_j)$$

where P_i and P_j, are pairs of parallel lines and P_i's first line is perpendicular to P_j's first line.

Once a candidate quadrangle is found it undergoes a validation step described in section 4.6.

4.5.2 Out of Frame detection

If no candidate quadrangles are found the system makes a final search based on an altered assumption. It is possible that the monitor is not entirely within the camera's view volume. In this case one or more of the monitor boundaries will not appear in the frame. This can lead to failure to track the monitor even when a small portion is the monitor is out of the frame. To handle this case the system makes a search for a partially occluded monitor assuming that at least three of the four monitor borders are on the screen.
Instead of searching for two pairs of parallel line pairs the system attempts to match each parallel line pair $P_f = (l_i, l_j)$ with an existing orthogonal line l_o and an estimated off-screen line l_e which is parallel to l_o. The estimated line l_e is produced from l_o by assuming l_e is parallel to l_o , that is l_e is the same theta parameter as l_o, but that l_e is translated from l_o by a distance , specifically l_e's r parameter is equal to l_o's r parameter plus or minus δ or $r_{l_e} = r_{l_o} \pm \delta$.
The distance delta is estimated from previous frame where a full detection was possible. Due to the uncertainty of the system of which line is being occluded the system uses the distance between the lines in the existing parallel pair P_f to determine the distance as follows.
When a full detection takes place the r parameter differences (δ_1, δ_2) between the lines within each of the two parallel pairs is recorded. In more simple terms this is equivalent to storing the respective width and height of the monitor as measured in the projected camera frame.
Subsequently when an out of frame detection takes place the system estimates the distance δ between the line l_o and the estimated line l_e by comparing the r difference of the existing parallel pair $P_f = (l_i, l_j)$ with the previously recorded δ_1 and δ_2. The estimated delta distance is set as follows:

$$\delta = \begin{cases} \delta_1 & : \text{if } ||dr(l_i, l_j)\delta_2|| < ||dr(l_i, l_j) - \delta_1|| \\ \delta_2 & : \text{if } ||dr(l_i, l_j)\delta_1|| < ||dr(l_i, l_j) - \delta_2|| \end{cases}$$

where $dr(l_i, l_j)$ is the r parameter difference between l_i and l_j

Hence for each pair $P_f = (l_i, l_j)$ the system attempts to find a line l_o and an estimated line l_e such that l_o is perpendicular to l_i and l_j, and l_e has an estimated distance from l_o so that the quadrangle (l_i, l_j, l_o, l_e) best matches the previously detected quadrangles geometry.

It is worth mentioning some weaknesses of this scheme. Firstly, the assumption that the estimated out of frame line is parallel to the detected line is not a valid one. When the monitor is viewed from directly in front of it this assumption holds however as the viewing angle varies more from this position the accuracy of this assumption decreases. While better approximations exist, they are out of the scope of this work.
Secondly, Since one of the lines is out of the video frame, the validation steps described in section 4.6 can not be performed on this line and hence the possibility of an erroneous solution is increased.

4.5.3 Importance of Sorting

Since performance is a key factor of the design of this system, an analysis of the complexity of this process is necessary.
Given N lines we first find the set of parallel pairs. Since each pair of lines is checked we perform $O(N^2)$ comparisons on this set and we expect to produce a list of parallel lines of size $O(kN^2)$ where $0 \leq k \leq 1$.
To then find sets of perpendicular pairs we must search through the list of parallel lines. Since the list of parallel lines is of size $O(kN^2)$ we expect a maximum of $O(k^2N^4)$ checks.
During implementation it was discovered that this section of the monitor detection algorithm could have significant effects on performance. Each of these candidate quadrangles undergoes a further expensive validation step and hence it was necessary to minimise the number of these checks.
For this reason the array of lines from the feature extraction step was sorted by accumulation buffer value. This has two benefits.
Firstly, knowing that the strongest lines came first in the array allowed the system to drop off lines near the end of the array. Hence during parallel line checking the system only processes the first 25 lines in the line array. This assumption relies on the fact that the monitor border lines are generally stronger. Processing is not performed on weaker lines which are unlikely to form part of a possible solution.
Secondly, the algorithm terminates on the first quadrangle which passes the validation step described in section 4.6. Hence it is favorable that the N^4 algorithm terminate as soon as possible. Due to the sorting of the line array, the solution is often found in a small number of checks. This is to say that the effective number of checks is m where $m << N^4$.

4.6 Validation step: Border Check

Testing showed that in a given frame there are often many quadrangle shaped boundaries. Hence without further validation, simply finding quadrangles in a frame was not sufficient. To address this a further check was added. The task of this check was to determine if this quadrangle matched the monitor by determining if the borders matched the dark monitor borders. Once a candidate quadrangle was found a check was performed on the four exterior border of the lines as illustrated in figure 4.3. The check examines the lumminence values along the exterior line of each border and counts how many values are above and below (on or off) the lumminence threshold defined in section 4.3. If this line consisted of mainly values below the monitor lumminence (on pixels) threshold then the solution is accepted and the border passes the validation check.

If all four monitor borders pass the validation check the the candidate quadrangle is accepted as a valid solution.
The algorithm does not require all points on a line to be under the lumminence threshold for two reasons:
Firstly if the lumminence threshold is set tightly then noise variations lead to some pixels on the monitor boundary being classified as not on the monitor boundary. These pixels could cause a potentially correct solution to fail. Secondly, if a user's hand is performing an action on screen their arm or hand will partially occlude the monitor boundary and cause many pixels on the monitor border to be above the lumminence threshold.

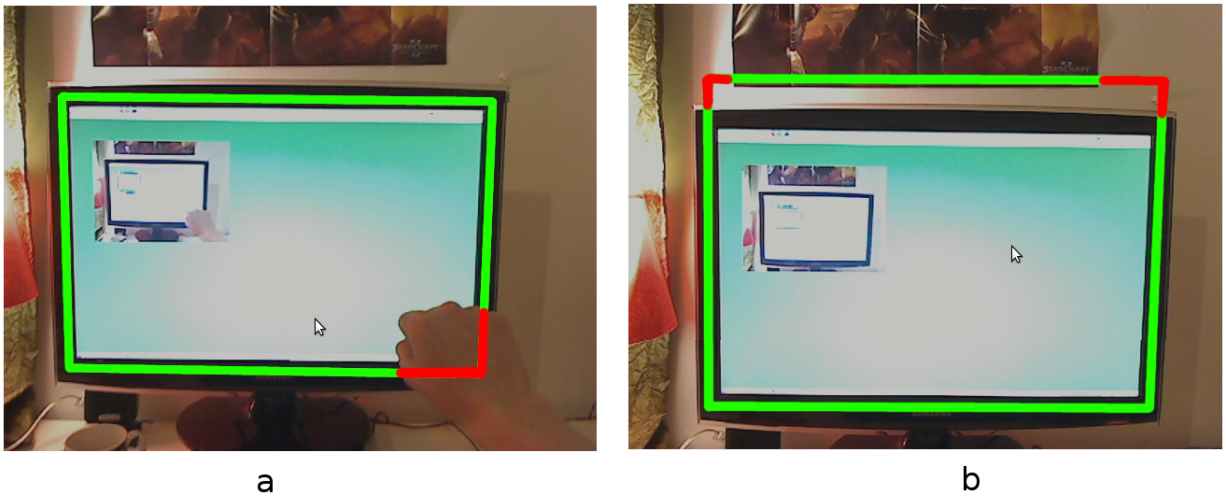


Figure 4.3: Graphic showing which kinds of occlusions are acceptable during the border check. Green pixels show where the border check passed (low lumminence values) and red pixels show where the border check failed (high lumminence values). (a) This quadrangle is acceptable, there are no occlusions on two parallel lines (opposite sides of the monitor). (b) This quadrangle is unacceptable since there exist occlusions on two parallel lines.

Instead the system allows a certain ratio of pixels on the candidate quadrangle's border to be above this threshold. The positions of the pixels which fall above the lumminence value are stored for each of the four borders in order to be used in the hand detection step as described in section 4.7.
However, the system does require that if two of the four border checks contain significant regions of pixels above the lumminence threshold they can not be in the same parallel line pair (opposite sides of the monitor). This requirement is designed to stop a false detection where one spurious the lines is near a correct line but is not connected by any dark pixels. This is illustrated in figure 4.3. This does, however, all two of border checks to contain bright pixels if they are perpendicular as is the case in the occlusion of a monitor corner.
One problem with this scheme is that this assertion can not be made for a partial detection since one of the four lines is out of frame and does not undergo a border check. This will be expanded upon in the results section.

4.7 Hand/Fingertip detection

Initial designs for the hand detection modules aimed to detect the hand silhouette against the monitor by skin colour segmentation or background subtraction. Skin colour segmentation was found to be difficult due to the large amounts of variation in illumination. Furthermore since the hand is silhouetted against a usually bright screen the resulting frames contain a dark colour for the skin pigment which can not be uniquely identified within a frame.
Background subtraction was found to be impossible due to the unknown and highly dynamic background presented by the computer monitor. Finally, it was decided to use a two stage hybrid algorithm which is bootstrapped by the occlusion of the dark monitor border by the hand. Hence in some sense this can be thought of background subtraction against the known dark monitor border. However its implementation is accomplished through lumminence thresholding as opposed to subtraction.
The hand detection algorithm can be split up into two logical steps. First the hand pixels must be segmented after which the system must parameterise the hand by detecting fingertip positions.

4.7.1 Step 1: Hand Silhouette Segmentation

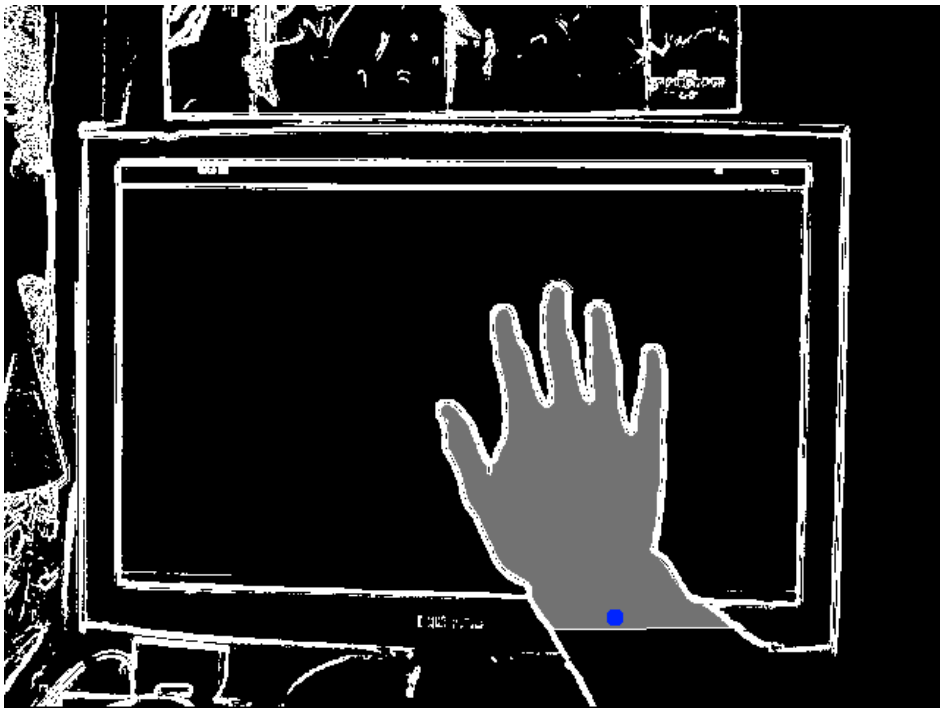


Figure 4.4: Results of connected component analysis in the edge buffer to achieve hand segmentation. Grey pixels represent pixels segmented as hand pixels. White pixels are strong edge pixels and black pixels are weak edge pixels. The blue dot shows the origin of the connected component analysis.

The first stage of the hand and fingertip detection algorithm is to segment the hand pixels within the frame. This task must be completed first before any further analysis can proceed.

If a hand is positioned to occlude the screen then either the hand or arm will occlude the monitor boundary. As mentioned in section 4.6, the monitor detection step records pixels on the monitor boundary which fall above the black lumminence threshold (off pixels). These off pixels on the monitor boundary will coincide with a brighter object which occludes the dark monitor boundary, and hence correspond to hand pixels when a hand is in the frame.
The first step is to detect if a hand is occluding the monitor. To achieve this the system determines if any of the four monitor boundaries had an off pixel count above a certain threshold. In this case we use the threshold N=15. If any borders satisfy this condition the system then selects the border with the maximum off pixel count. In this context we now refer to off pixels as hand pixels.
The mean position of the hand pixels of this border is then computed. Testing showed that this position did not always coincide with another hand pixel due to noise variations and more complex occlusions such as the double occlusion of two fingers where the mean position falls between them. For this reason the system searches locally for a hand pixel.
The search is performed on the line of the border check line and progressively searches the range around the mean position starting at [mean,mean] and ending at [mean -80,mean + 80]. The search terminates when a hand pixel is found. The location of this original pixel is used later and we shall call it the principal hand pixel.
Further hand pixel segmentation is accomplished through connected component analysis on the edge image produced from the Sobel filter. The process is started at the detected hand pixel. Pixels are accepted as neighbors by the following binary function:

$$\text{accept}(p) = \begin{cases} 1 & : \text{if } NE(p) < E_T \text{ and } LUM(p) < L_T \text{ and } p \in B \\ 0 & : \text{otherwise} \end{cases}$$

where $NE(p)$ is the normalised edge value at p,

$LUM(P)$ is the lumminence at p and B is the *extended border area*.

E_T and L_T are the edge and lumminence threshold respectively

Hence the connected component analysis produces a set of pixels representing the estimated hand pixels. It should be noted that the accept function operates on edge values. This method is more robust to illumination change than methods which rely on a skin colour. However this technique relies on the assumption that skin colour changes continuously and gradually throughout the hand and that the background to the hand is a sufficiently different colour so as to produce a strong edge on the hand boundaries.
In this expression the *extended border are*a refers to the quadrangle and interior set of pixels created by slightly enlarging the monitor quadrangle. The expression $p \in B$ ensures that the connected component analysis does not accept pixels outside of the monitor boundary. This has two benefits.
Firstly the connected component analysis will not mark unnecessary pixels all the way down the arm until the end of the frame, thus wastefully increasing processing time.
Secondly, areas outside the monitor boundary are often badly illuminated. In these areas the edges and boundaries between segments, say the arm and the background, become weaker which can cause the connected component analysis to 'leak out' of desired areas. More will be discussed on this topic in the results section.
The segmented hand pixels are stored as a binary mask in order to be used further.

4.7.2 Step 2 (Fingertip detection)

The system detects fingertips by analysing the hand contour. However, alternative fingertip detection schemes were analysed. One such scheme is the use of convex hull algorithms. In this technique the convex hull of the hand silhouette is computed. The resulting vertices in the convex hull generally are the fingertip points, excluding some points which can be automatically removed. However, it was observed that the fingertip points do not always form a convex polygon and in these cases the algorithm would fail. For this reason we decided to use contour analysis to detect fingertips.

Hence it is necessary to chain and parameterise this contour for further analysis, to this end the Moore Contour tracing algorithm is employed. Jacob's stopping criteria was not used since the cases where this was required are not encountered in this environment. The limitation that Moore's tracing algorithm only traces the exterior contour does not present a problem since the system needs only the outer contour point for fingertip detection. Moreover, the lack of interior contour points negated the possibility of spurious fingertip detections at interior holes in the hand silhouette.
The result of the contour tracing step is a list of adjacent contour points in clockwise order. To detect the fingertip a search is performed on this list.
Let P be the list of contour points. The pseudo curvature at each point P_i is examined as follows.

Let v_1 be the vector (P_i, P_{i+k}) and v_2 be the vector (P_i, P_{i-k}) where $k > 0$ is an integer constant called the scale. The angle θ between $v1$ and $v2$ is computed as

$$\theta = cos^{-1}\left(\frac{v_1 v_2}{|v_1||v_2|}\right)$$

. This angle is computed at a number of scales. If this angle is less than 0.5 radians for any of the scales then we mark the pixel as a fingertip. This process leads to many pixels on each the fingertip being marked as fingertips.
Once all fingertips have been added the system searches for local maxima in contour space.

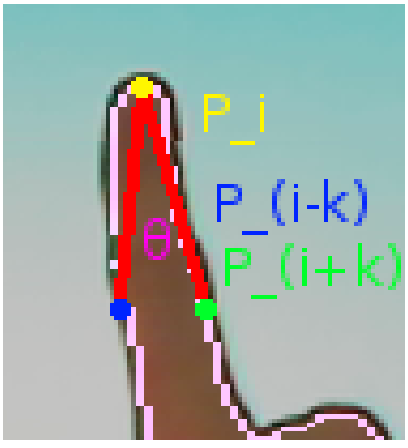


Figure 4.5: Graphic showing how pseudo curvature is computed for fingertip detection.

For each cluster of fingertip pixels the system selects the pixel with the maximum distance from the principal hand pixel. The result of this is that the pixels on the tip of the finger are chosen

to represent the fingertip position.
This technique was chosen for its simplicity and speed. More complex methods exist, such as fitting conics to the fingertip. However testing showed that this technique was sufficient. One result of this method is that not only the fingertips were detected but the valleys between fingers as well. The system corrects this by computing the average contour position. Once this is computed a check is made on each fingertip to ensure that they are above a certain distance from the average contour position.

4.8 Homography estimation

Homography If monitor detection is successful the system knows the four corner points of the monitor from the intersections of the four monitor border lines. Furthermore if a hand and fingertips are detected, the positions of these fingertips are also known. All of these measurements will naturally exist as camera coordinates. In order for these measurements to be used as a control system it is necessary to transform them into the native space of the monitor display.
This kind of mapping is known as a planar homography. In particular we need to map the coordinates from camera space to monitor space.
Let p = $(XY1)$ be the camera space position of an interest point in homogeneous coordinates. Let $M = \{c_{tl}, c_{tr}, c_{bl}, c_{br}\}$ be the collection of the top left, top right, bottom left and bottom right corner position of the monitor in camera space. Similarly let $D = \{d_{tl}, d_{tr}, d_{bl}, d_{br}\}$ be the collection of corners in display space. If the monitor resolution in pixels has a width of w and height of h then $d_tl = (0, 0) d_tr = (w, 0), d_bl = (0, h)$ and $d_{bl} = (w, h)$.
We require the homography matrix $H : M \rightarrow D$ such that $Hc_{tl} = d_{tl}$, $Hc_{tr} = d_{tr}$, $Hc_{bl} = d_{bl}$, $Hc_{br} = d_{br}$.
Since the sets M and D are known we can solve this system for H by means of a least squares solution. For this purpose the OpenCV library is employed.
Given H it is now possible to transform any point $p = (uv1)$ in homogeneous camera coordinates to a point $p' = (xyw)$ in homogeneous display coordinates by matrix multiplication:

$$Hp = p'$$

In this case we have the display x coordinate $x' = x/w$ and y coordinate $y' = y/w$.

4.9 Pose Estimation

Given the four corner points in camera space it is also possible to determine the rotation and translation of the camera with respect to the camera. This process relies on accurate measurements of the monitor corner positions as well as an internal representation of the monitor geometry.
For pose estimation the system makes use of the POSIT algorithm, a 2 part iterative algorithm to compute object pose given four or more corresponding pairs of points in the camera projection and on the model.
The result of this process is a rotation matrix R and a translation matrix T describing the rotational and translational components of a reference point on the model with respect to the camera.

4.10 Data produced

Once all stages of the monitor and hand detection algorithms have been completed the system has knowledge of the positions and regions of the monitor, hand and fingertips. Each of pieces of information has their own specific object representation as shown in table 4.10.

Data type	Description
Monitor lines	Four (r, θ) pairs specifying line parameters
Monitor Corners	Four (x, y) positions of monitor corners
Monitor Pose	3x3 rotation and translation matrices R, T
Hand Silhouette	Binary mask identifying hand pixels
Fingertips	List of (x, y) pairs of detected fingertip positions

Table 4.1: Table showing types of data produced by the system.

All this data can be used as functional input to a program.

4.11 Summary

In this chapter the implementation details of each of the components was discussed. The constraints inherent in solving each stage of the problem were discussed and from this the chosen methods were explained as well as the reasons for these choices.

Chapter 5

Results

In this chapter we analyse present and analyse the results obtained through various experiments. This analysis focuses on two areas: Performance and Accuracy.
Performance analysis focusses on the performance obtained while running test applications. We also focus on the effect which various design choices have on the overall performance.
The accuracy results will attempt to measure how correctly the problem features are identified. More specifically, we look at the accuracy of monitor detection, hand silhouette segmentation and fingertip detection.

5.1 Testing Environment

All experiments were run on an quad core Intel i7 3.2Ghz processor with 6GB of RAM. Two test applications were used for the experiments.
The first testing application is a simple drawing program. The system draws a pink square at each detected fingertip position for each frame. The canvas of the drawing program is a constant light green colour. A user can draw with either one fingertip or many. The benefit of this application is that the trajectory of fingertips can be easily viewed since they are drawn onto the canvas.
The second test application which was used is a 3 dimensional deformable terrain environment renderer.This software allows a user to fly around a computer rendered terrain and modify it in real time. The benefit of this application is that the images on screen are dynamic and the texture is not constant, thus taxing hand segmentation algorithms.

5.2 Performance

During design and development many optimisations and design choices were made to achieve the highest possible performance. Here we analyse the overall performance achieved by the system. The effects of individual optimisations are studied as well a brief theoretical analysis of the factors which affect run time.

5.2.1 Frame rates

The one of the main goals during the design and implementation of this system was to achieve interactive frame rates. This was necessary in order for the system to provide useful input to

a program. An experiment was run to measure the frame rates achieved by this system at various resolutions. The system was run for a period of 30 seconds. Frame rates were recored during running and the average frame rates were computed. The results of this experiment are recorded in table 5.2.1.

These experiments were run with the held camera at a near constant position facing the mon-

Resolution	Max HW FPS	Average FPS	Percentage Utilisation
352x288	30	28.523	95.08%
640x480	30	28.229	94.10%
800x600	25	24.037	96.15%
960x720	10	9.328	93.28%
1600x1200	5	4.435	88.70%

Table 5.1: Frame rates for different resolutions.

itor. The only movements involved were a small amount of camera jitter. The *Max HW FPS* column represents the maximum reported hardware frame rate for each resolution. It should be noted that while the hardware reports these rates, the effective rate is usually slightly lower and subject to constant variation.
These results show that the system is achieving very good frame rates even at high resolutions. It is interesting to note that the frame rates are not significantly affected by image resolution. The maximum hardware frame rate at 352x288 and 640x480 resolutions is the same and a similar effective frame rate is achieved even though the system needs to process 3.03 times as many pixels in the latter case. This suggests that the software is either not bound by the processing of pixels or that processing is occurring fast enough for for frame reading to be the limiting factor. This shall be studied further in the following section.

5.2.2 Time analysis

Time analysis was performed to identify the time contributions of each of the stages involved in the processing of a frame. The total time to complete each task was recorded in milliseconds for each frame which was processed. For this section the resolution used was 640x480 and the drawing test application was used. The different stages which were timed are:

Total time The total time to process a single frame

Feature Extraction time The time taken to extract features. This includes edge detection and line extraction through Hough transform.

Monitor Detection time The time taken to detect a monitor from the list of lines.

Hand Detection time The time taken to detect and parameterise a hand.

The figure 5.1 shows the timing values for a standard sequence of frames. The hand is present in these frames and the camera is at a approximately stationary.
As can be seen, the majority of time is spent in the feature extraction step. Feature extraction

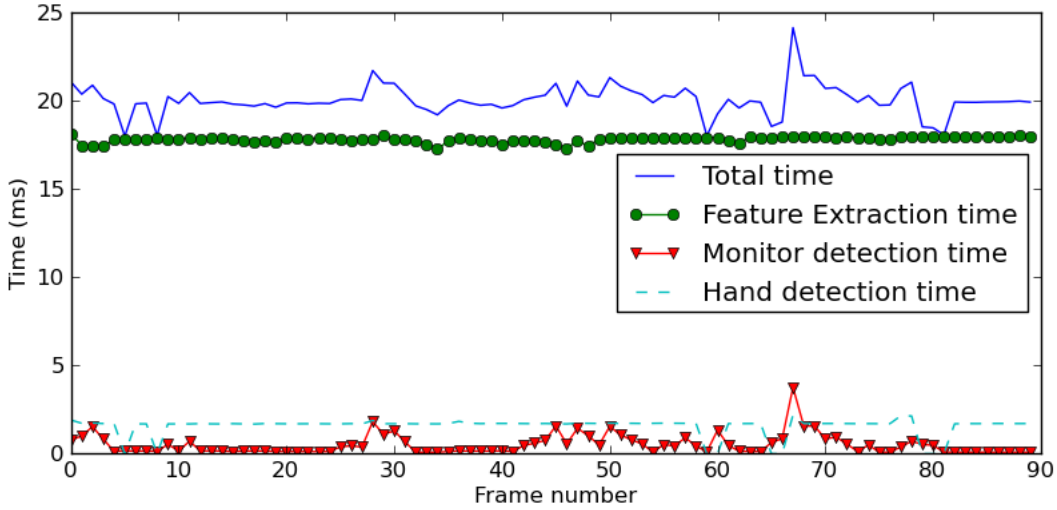


Figure 5.1: Timing graph for a sequence of 640x480 frames, all optimisations present.

has two stages itself. The first is the Sobel filter convolution. For each pixel in the input image the system must apply two 3x3 Sobel filters in the x and y directions. The resulting edge values are stored in a n array. The second stage of the feature extraction is the Hough transform. Even though this voting scheme has been optimised the system must still cast 20 votes for each edge edge pixel on the monitor border.
Hand detection is consistently the second largest time contributor. This process includes a connected component analysis which is implemented recursively instead of a queue based implementation. After this a tracing algorithm is performed to chain the hand contour. Finally the contour is iterated to detect fingertips. The whole hand detection stage therefore scales on the size of the segmented hand within each frame.
The monitor detection step takes very little time on average, however, at certain points the time peaks and has a larger affect on the overall processing time. As mentioned in the implementation chapter, this algorithm has a complexity of $O(N^4)$ where N is the number of lines within an image. Many techniques have been implemented to ensure that a solution is found as soon as possible. However it is apparent that at some points the system needs to examine many candidate quadrangles before a solution is found. This causes the monitor detection time to be more variable than the feature extraction and hand detection steps. The average time taken to process a frame is approximately 20 ms or 0.02s. At this resolution the camera produces 30 frames per second. In this case the system will not struggle to maintain real time frame rates since the total time taken to process 30 frames will be $30 \times 0.02 = 0.6s$. Hence a fair amount of time is spent waiting for the next frame to arrive.

Looking at the timing values in table 5.2 we see similar results for all resolutions. Invariably the most time consuming part of the algorithm is the feature extraction stage. This stage scales strongly with resolution. This is to be expected since this is the only stage which processes all pixels within an image. The hand detection stage also scales with resolution. The processing requirements of this stage are related to the size of the segmented hand and hence we expect this scaling relationship.

Resolution	Total Time(ms)	Feature Time(ms)	Monitor Time(ms)	Hand time(ms)
352x288	7.210	6.010	0.151	0.927
640x480	20.041	17.825	0.508	1.560
800x600	31.916	27.408	1.342	2.975
960x720	39.157	36.131	0.406	2.509
1600x1200	108.722	95.604	2.815	8.668

Table 5.2: Timing values for different resolutions.

5.2.3 Monitor Detection

All timed stages run in a fairly constant time except for monitor detection which tends to have spikes in its performance. Here we analyse the monitor detection in detail and determine the reasons for this varying performance. Figure 5.2 shows the number of different kinds of checks which occur during monitor detection in each frame. These values show the number of checks before a correct solution is found. The different kinds of checks are:

Number of Parallel line pairs After line detection the system builds a list of parallel line pairs. Each of these pairs contain two lines which are approximately parallel.

Number of Angle Comparisons During monitor detection the system compares the angle between two parallel line pairs in order to find two perpendicular parallel line pairs. To do this the system must check the angle between two line pairs.

Number of Candidate quadrangles If two line pairs are approximately perpendicular the system accepts them as a candidate quadrangle. Each candidate quadrangle undergoes a border lumminence check before it can be accepted as a solution. This lumminence check is far more computationally expensive than the initial angle comparison.

It is clear that the number of steps taken to find the monitor is fairly inconsistent. For some frames the total number of checks taken before a correct solution is found is fairly low while in other frames this value can be over 1000 border checks. This behavior can be understood if we analyse the nature of the border detection process. Suppose $P_i = (l_a, l_b)$ and $P_j = (l_c, l_d)$ are two pairs of parallel lines and let the list of parallel lines be of size N. The system attempts to find parallel pairs that are perpendicular and are surrounded by a dark border. This is accomplished by comparing every pair of parallel pairs.
Suppose then that the parallel pair P_0 is composed of two lines which are not on the monitor boundary. Still P_0 must be compared with $P_1, P_2, P_3, \ldots, P_N$ even though all these comparisons will fail since P_0 is not part of a correct solution. Suppose this again is the case for P_1 then we can see how this process will lead to a large number of checks.
As mentioned in the implementation chapter, sorting is used to increase the probability of the correct solution appearing near the beginning of the array however this is not always totally effective. However if we look at the worst case for the number of angle comparisons we find that this optimisation has helped greatly. Let N be the length of the list of parallel pairs. The

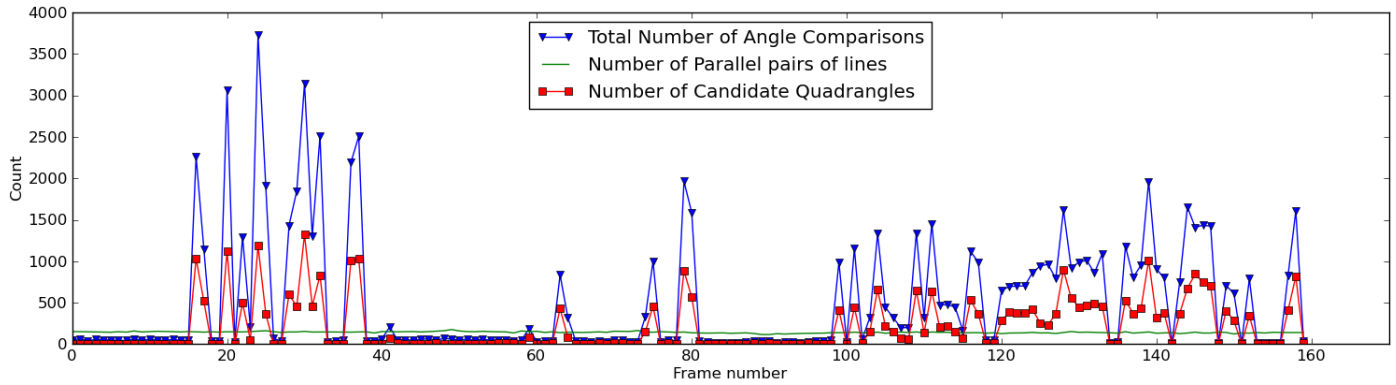


Figure 5.2: Monitor detection search values for each frame.

maximum number of angle comparisons is then $\frac{N(N+1)}{2}$. Using the average parallel pair list size of $N = 139$ as in figure 5.2 we find that the worst case number of line comparisons is 9730. This value is over twice as large as the maximum number of angle comparisons for our dataset. However when part of the monitor is out of the frame the system must first exhaust all on-screen lines before an off-screen detection can take place. In this case the number of angle comparisons is greatly increased. However the number of candidate quadrangle checks is not increased significantly since the monitor being off-screen does not affect the number of quadrangles within an image. Figure 5.3 illustrates this. The monitor leaves the screen at approximately the 12th frame. Immediately the angle comparison count rises as the system searches all possible on screen lines for a quadrangle before continuing to a partial detection.

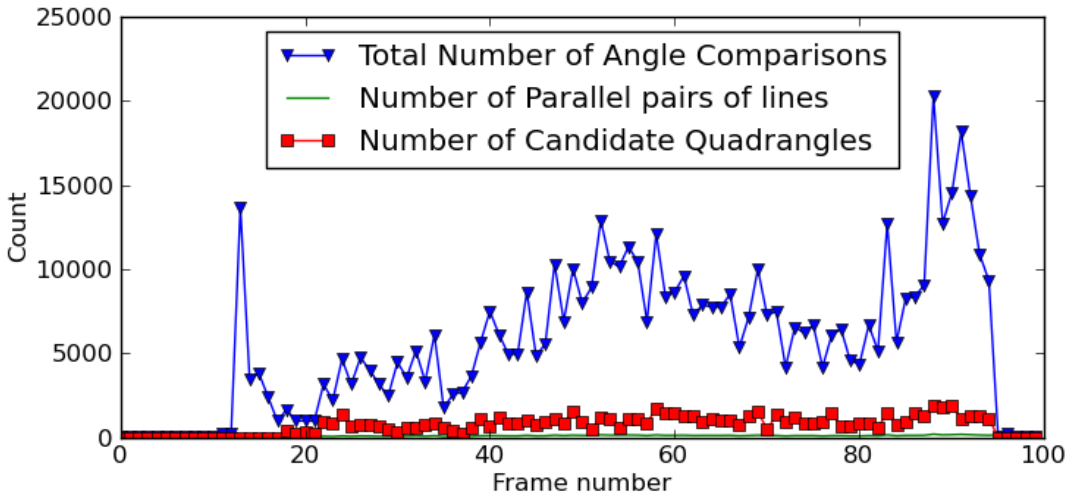


Figure 5.3: Monitor detection search values for a monitor partially off frame. Monitor leaves the frame in the 12th video frame.

These results suggest that an improved iteration method can be developed. While the current pair search method does not inhibit performance it is conceivable that this could happen under certain circumstances. An improved search method would not process every pair in its list order but select pairs to compare on more intelligent criterion thus reducing the total number of comparisons before a solution is found.
It is worth mentioning that the environment can have a significant effect on performance. Since speed up is accomplished through sorting of lines, if many strong lines exist within the environment they will lead to wasteful checks. However, the kind of line which would cause such hold ups is fairly limited due to the lumminence threshold requirements.
The effect of the monitor detection optimisations becomes apparent when they are turned off. Removing line sorting and the 25 line cut off lead to frame rates of less than 1fps where the biggest contribution to processing time was the monitor detection.
The 'on off' optimisation in the feature extraction also had a similar impact. Turning this off increased processing time significantly. This caused large processing lags due to the extra votes as well the additional lines which were detected.

5.2.4 Performance Conclusions

In this section we showed results from frame rate experiments. Initial frame rate results showed good performance. More detailed time analysis revealed that the majority of time is spent on feature extraction however monitor detection can have significant effects. Experimentation showed that without monitor detection optimisations the system could not achieve real time performance. Total frame processing times showed that the system processes frames faster than they are produced.
The monitor detection algorithm was studied in depth. This analysis showed how the monitor detection time is inherently variable since it relies on the order of processing. While line sorting has beneficial effects, in certain frames the monitor detection had to check many candidate quadrangles suggesting an improved iteration method.
The performance of the system during use was satisfactory with minimal latency and a high frame rate.

5.3 Accuracy

System accuracy was the second main goal in design and implementation. In this section we explain our accuracy criterion, results obtained as well as reasons for various results. Results are analysed in three areas: Monitor detection, Hand Silhouette detection and Fingertip detection.

5.3.1 Monitor Detection

To test the monitor recognition rate the program was run with the camera facing the monitor in full view. A frame was considered successful if the monitor was correctly recognised meaning that the four monitor corners were detected correctly. During these tests the monitor recognition rate was very good. The first tests kept the camera in fairly constant positions. These tests showed a 100% recognition rate. In every frame the monitor border lines were detected and the four corners were correctly positioned. Figure 5.4 shows correct detection of different monitors.

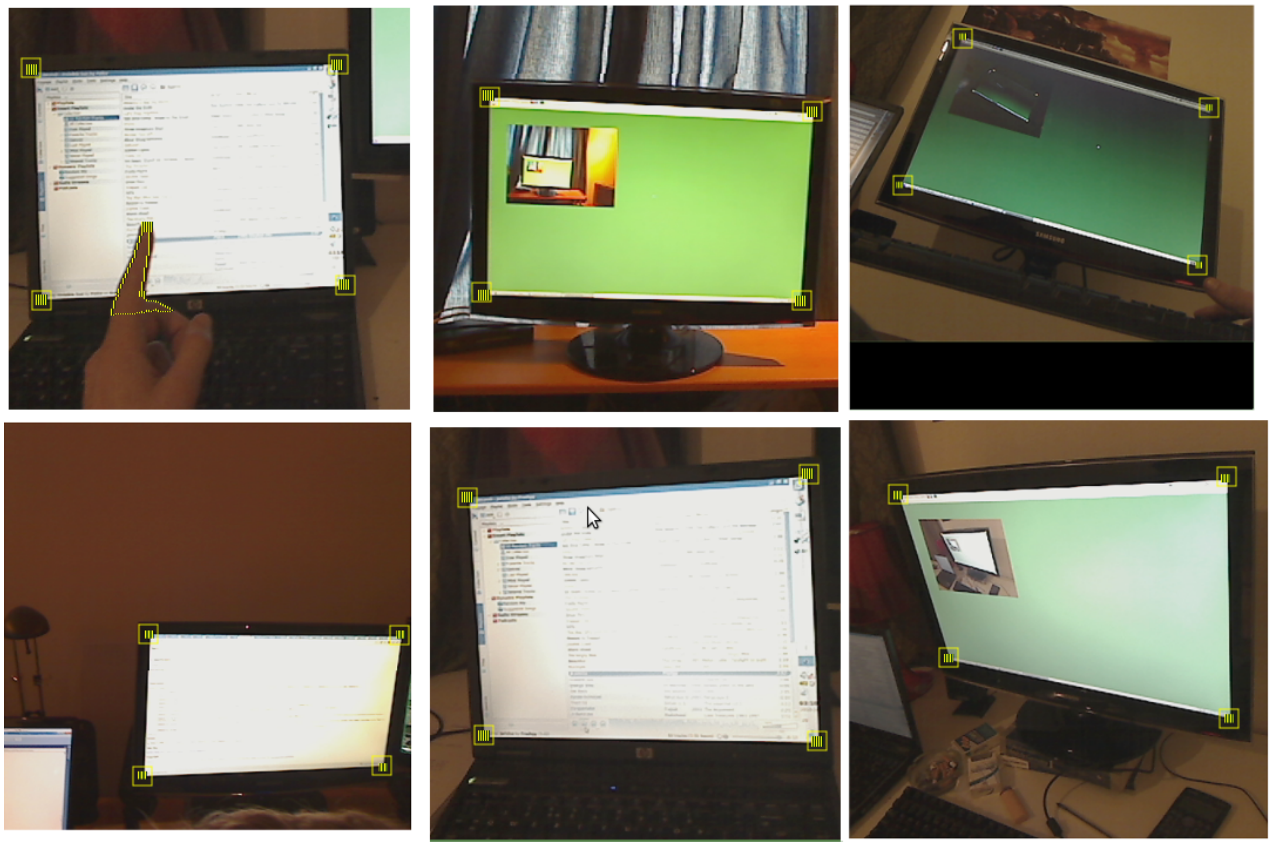


Figure 5.4: Monitor detection results for six different frames. Monitor corners represented by a solid yellow square. Note, the outer enclosing yellow squares have been added manually to aid viewing.

Testing under camera motion also showed good results. Detection only failed during rapid camera translations. Figure 5.5 shows one such frame. In these frames the images were blurred and the monitor boundary lines were no longer detectable.

Figure 5.5: Blurring due to rapid camera motion. Notice how the monitor border lines become blurred.

Partial Detections

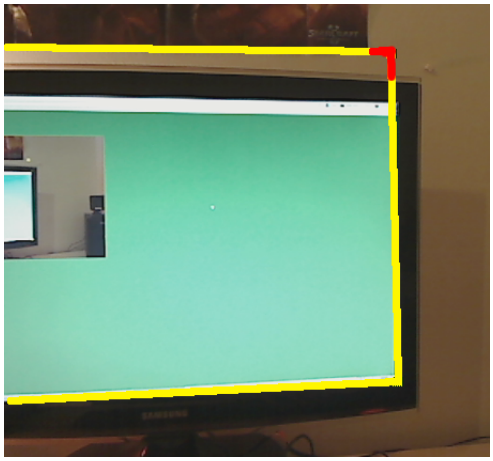


Figure 5.6: An example of an incorrect partial detection. This is caused by the lack of information which is out of frame.

Partial detection are detections where one of the four monitor boundary lines is out of the video frame. This condition had various accuracy implications which we discuss here. Very rarely the system would incorrectly detect one of the four monitor lines. This did not occur in normal testing but only happened during partial detections when one of the borders was out of the frame. The cases where a line was incorrectly detected all had a similar nature. Incorrect detections only occurred when there was a dark straight line region very near and parallel to one of the monitor boundaries. An example of this can be seen in figure 5.6.

This is a case which relies not on the monitor itself, but on objects in the environment. The case shown in figure 5.6 is by its nature very difficult to handle. The incorrect line is positioned very near to a correct line. One check designed to stop this is described in section 4.6. This check performs well when the screen is totally in the frame however it is impossible to make this check when the monitor is partially out of the frame as in figure 5.6. As described in the implementation chapter, the system needs to allow some pixels on the monitor boundary to fall above the lumminence threshold during monitor detection. This is to deal with cases where the hand is on screen. A check was added to ensure that if there are two regions occluding the monitor, they can not be on opposite sides of the monitor. However when a partial detection takes place only three of the four lines undergo a border check since one line is out of frame. In this case the assertion can not be made and spurious lines are accepted as in figure 5.6.
Hence if a dark straight region is near one of the monitor boundaries during a partial detection the likely hood of a false detection is increased. The system can not ignore this line since the high lumminence region could be that of a users hand.
It should be noted that this case is a very specific one. The requirement for this kind of error is a dark straight object positioned near and parallel to a monitor boundary. The partial detection accuracy showed good results. As mentioned in the implementation chapter the system predicts the out of frame position of one of the four monitor boundary lines. The accuracy of this prediction can be measured by where the system determines a fingertip position to be during a partial detection and how closely this matches where the fingertip is perceived to be.

Testing showed that this produced accurate results when the out of frame line was not very far out of frame. In these cases a fingertip was mapped to a correct position on screen. However, as less of the monitor was in view the accuracy of this prediction degraded. This was observed by how the system incorrectly computed the on-screen fingertip position. In extreme cases the on-screen fingertip position was very far from the actual fingertip position.
These results are expected due to the weak assumption made for partial detections. The system assumes that the monitor geometry remains the same during partial detections, the only difference it assumes is that the monitor is translated out of frame. The true case is that as the monitor leaves the frame its image becomes scaled in the direction it is leaving. Furthermore, given only three of the four lines of the monitor it is impossible to correctly predict the position of the fourth - many possible configurations exist.

near lines reflections spurious detection position when correct speed stuff offscreen. bad hand point.

5.3.2 Hand Silhouette Segmentation

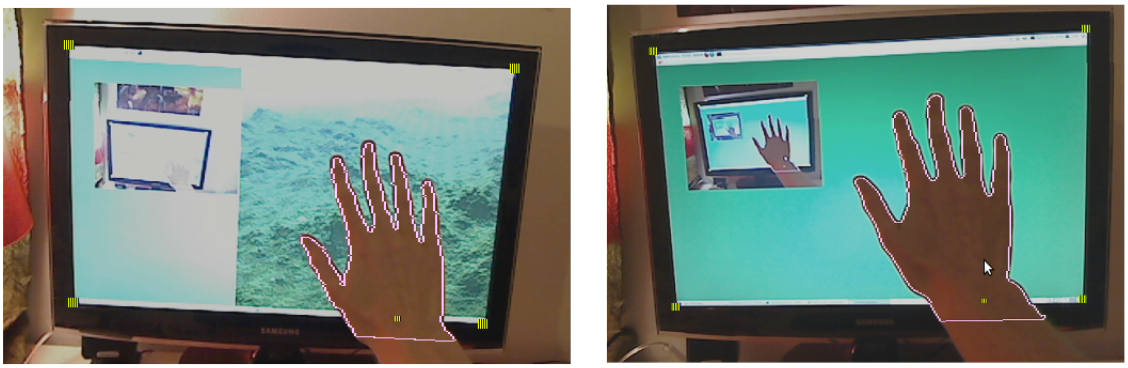


Figure 5.7: Segmentation results against the terrain and drawing application backgrounds.

Hand silhouette detection is a difficult problem. The constraints of this particular implementation, such as a dynamic background with varying illumination, add to the difficulties involved. Here the accuracy of the hand silhouette detection is analysed. Problems encountered during testing are detailed and discussed.
The hand silhouette detection was tested using the drawing application as well as the terrain application. These programs test the detection accuracy under different circumstances. The drawing test contains a constant colour background which is well illuminated. The terrain background is noisy and contains large variation in illumination.
Tests in the drawing application produced very good results. The segmentation of the hand was satisfactory, producing consistently accurate segmentations. Inaccurate segmentation was rare and occurred under a limited number of cases which shall be discussed later in this section.
Tests in the terrain application produced mixed results. Segmentation was accurate in the majority of frames however a number of cases lead to consistent inaccurate segmentation.

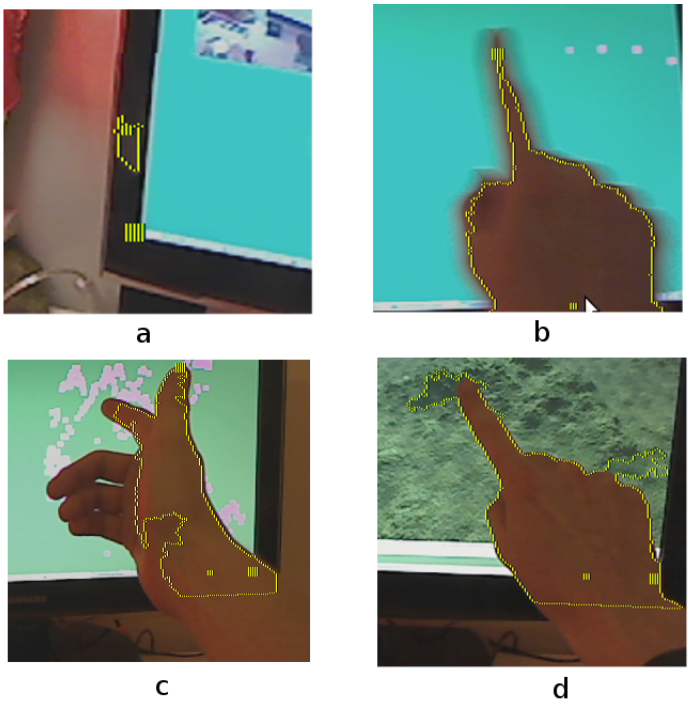


Figure 5.8: Various cases of Inaccurate segmentation: (a) Monitor reflections (b) Rapid hand translation (c) Shadowing (d) Leaking

Inaccurate Segmentation cases

While inaccurate segmentation was rare, it occurred under the following conditions:
Monitor Reflections When a bright reflection was present on the monitor border the system misclassified this region as a set of occluding hand pixels. This misclassification did not cause significant problems however since the reflection region did not extend onto the screen and hence did not create spurious fingertips. In certain cases these reflections could be dealt with by increasing the lumminence threshold. See figure 5.8 (a).
Fast Hand translation When a hand is moving rapidly through a frame the hand region becomes blurred. This lead to an incomplete segmentation where many pixels belonging to the hand were not classified as being in the hand pixel set. See figure 5.8 (b).
Shadowing If a users hand is illuminated in such a way that strong shadows are present on the hand silhouette the system does not segment the shadowed pixels. This leads to an incomplete segmentation. This could be fixed in almost all cases by adjusting the lumminence threshold (which is set too high). See figure 5.8 (c).
Leaking When the on-screen images were noisy or dark, the connected component analysis leaked onto the screen, classifying screen pixels as hand pixels. This was due to weak edge pixels in certain areas. This only occurred while testing in the terrain application. The effects of this problem could be minimised by adjusting the edge threshold in the connected component analysis, however, this led to slightly more noisy segmentations. See figure 5.8 (d).

5.3.3 Fingertip Detection

Fingertip detection results were obtained through testing with the drawing application as well as the terrain manipulation program. When hand silhouette segmentation was correct the results of fingertip detection were very good. In these cases there were no spurious fingertip detections. The only problems encountered in these cases were failures to detect fingertips. These shall be discussed further later in this section.

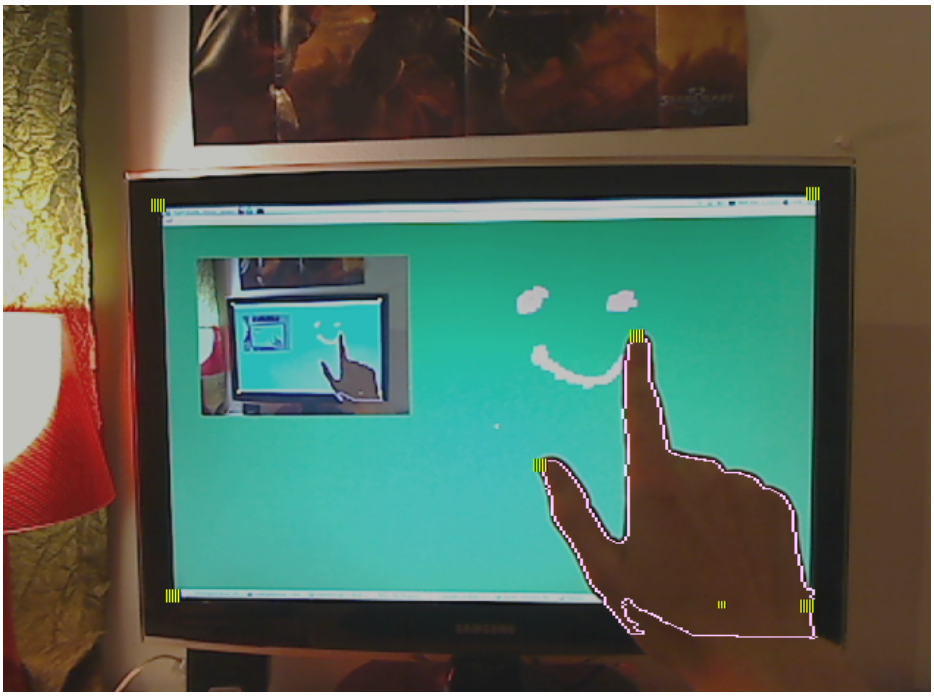


Figure 5.9: This figure shows fingertip detection results in the drawing application.

Firstly, we analyse the reasons for the lack of spurious fingertip detections. As mentioned in the implementation chapter, fingertip detection is accomplished through analysing a pseudo-curvature on the hand contour. A point can only be marked as a fingertip if it satisfies an angle constraint. The design of this constraint is such that only high curvature pixels can be marked as fingertip pixels. Due to the inherent geometry of a hand silhouette, only fingertip positions and the low points between fingers matched this criterion.
This is true of course for correctly segmented hand silhouettes. When the silhouette detection was incorrect the system sometimes registered false fingertips. This occurred mainly when the *leaking* problem was present. It should be noted that these false fingertip detections are caused by errors in the silhouette detection, not the fingertip detection itself.

Failed Detections
When segmentation was correct there were no spurious detections although there were cases where the system failed to detect a fingertip. This occurred in two specific cases:

Scale Problems These failures occurred when the scale parameter k defined in fingertip detection was not high enough. This error occurred only at high resolutions. The problem is

caused by insufficient distance between curvature vectors P_{i-k}, P_i, P_{i+k}. Increasing k fixed this problem.

Incorrect Palm centre distance To remove the fingertips detected at low points between fingers a distance check was added. Simply put, this check ensured that a fingertip was above a certain distance from the palm. However, in certain cases a true fingertip would be under this distance and hence the point would not be identified as a fingertip. Adjusting the palm distance threshold could fix this in certain cases.

5.3.4 Accuracy Conclusions

In this section the accuracy of the system was analysed. Experiments produced good results in monitor detection, hand silhouette segmentation and fingertip detection. Cases were misclassification were found and analysed, suggesting modifications to various portions of the process. In almost all cases the system performed accurately providing accurate results for input to the two testing programs.

Chapter 6

Conclusion

In this report we detailed the design of a vision based screen interaction system. A comprehensive review of relevant literature was performed. This review aided in design and implementation. The goals and constraints inherent in such as design were explained. This included the constraints introduced by the hardware, environment and processing capabilities of the systems involved.
We successfully implemented a software package which performs the tasks outlined in this report. This package produces usable data in real time. Two applications were used for testing of the system with satisfactory results. Many deviations were made from the initial design due to lessons learned during development. The reasons and effects of these deviations was explained leading to conclusions about the problem as a whole.
In detail analysis of both performance and accuracy measures was performed. The performance analysis showed very good results. In particular we showed that at all resolutions the system was able to process frames in real time. Timing data was produced for all stages of processing. These results gave important insight into the complexity and computational requirements of the problem. The monitor detection system was studied, explaining the nature of our achieved performance. This analysis suggested additional optimisations and modifications which could further increase the system performance.
Accuracy testing also produced very good results. Monitor detection for complete monitor images showed a 100% recognition rate. Similarly hand silhouette segmentation and fingertip detection produced very good results with almost perfect recognition rates. Furthermore, through testing, we were able to identify all the cases in which the system failed. These failures suggest various modifications which can be made in order to improve the detection accuracy.

6.1 Future Work

During testing various possible modifications to the current detection schemes became apparent. Firstly, the monitor detection routines could be greatly improved by a more intelligent quadrangle detection method. The current method required a large amount of optimisation to mitigate the factors of large searches. Some of these optimisations came at a cost, suggesting that alternative methods could increase recognition rates in ambiguous cases. Specifically a better pair matching algorithm could be developed which performs better than the current $O(N^4)$ algorithm.
Tracking solutions could be investigated as they may provide benefits in accuracy and computational efficiency. The current system does not make use of any spatial or temporal coherence,

even though many variables within the system are correlated across frames.
Hand detection routines were not model based. False detection errors could be addressed through the use of a model based hand recognition system. Such a system would not allow hand segmentation solutions which deviate significantly from an internal hand representation. Finally, the fingertip and monitor position data produced by the program is discrete by the nature of frame based video. The effect of this is a jump in pointer position between frames. Kalman filtering or other predictive techniques could be employed to obtain inter frame position estimates as well as reduce noise in measurement.

Bibliography

[Boult et al. 2000] Boult, X. G., Gao, X., and Ramesh, V. 2000. Error analysis of background adaption. 503–510.

[Bray et al. 2004] Bray, M., Koller-Meier, E., Müller, P., Gool, L. V., and Schraudolph, N. N. 2004. http://nic.schraudolph.org/pubs/BraKolMueVanetal04.pdf 3D Hand Tracking by Rapid Stochastic Gradient Descent Using a Skinning Model. In *First European Conference on Visual Media Production (CVMP)*, 59–68.

[Canny 1986] Canny, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6, 679–698.

[Chen et al. 2001a] Chen, Y., Rui, Y., and Huang, T. 2001. Jpdaf based hmm for real-time contour tracking. I:543–550.

[Chen et al. 2001b] Chen, Y., Rui, Y., and Huang, T. 2001. Jpdaf based hmm for real-time contour tracking. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, I–543 – I–550 vol.1.

[C.S. Chua 2000] C.S. Chua, H.Y. Guan, Y. H. 2000. Model-based finger posture estimation. In *Fourth Asian Conference on Computer Vision*, 43–48.

[Delamarre and Faugeras 2001] Delamarre, Q., and Faugeras, O. 2001. 3d articulated models and multiview tracking with physical forces. *Comput. Vis. Image Underst. 81*, 3, 328–357.

[Dementhon and Davis 1995] Dementhon, D. F., and Davis, L. S. 1995. Model-based object pose in 25 lines of code. *Int. J. Comput. Vision 15*, 1-2, 123–141.

[Dinstein et al. 1977] Dinstein, I., Shanmugam, K., and Haralick, R. 1977. Textural features for image classification. In *CMetImAly77*, 141–152.

[Duda and Hart 1972] Duda, R. O., and Hart, P. E. 1972. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM 15*, 1, 11–15.

[Elgammal et al. 2000] Elgammal, A. M., Harwood, D., and Davis, L. S. 2000. Non-parametric model for background subtraction. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, Springer-Verlag, London, UK, 751–767.

[Erol et al. 2007] Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., and Twombly, X. 2007. Vision-based hand pose estimation: A review. *Comput. Vis. Image Underst. 108*, 1-2, 52–73.

[Fernandes and Oliveira 2008] Fernandes, L. A. F., and Oliveira, M. M. 2008. Real-time line detection through an improved hough transform voting scheme. *Pattern Recogn. 41*, 1, 299–314.

[Gavrila and Davis 1996] Gavrila, D. M., and Davis, L. S. 1996. 3-d model-based tracking of humans in action: a multi-view approach. In *CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, IEEE Computer Society, Washington, DC, USA, 73.

[Greenspan et al. 1994] Greenspan, H., Belongie, S., Perona, P., Goodman, R., Rakshit, S., and Anderson, C. 1994. Overcomplete steerable pyramid filters and rotation invariance. 222–228.

[Huttenlocher et al. 1993] Huttenlocher, D., Noh, J., and Rucklidge, W. 1993. Tracking non-rigid objects in complex scenes. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, 93 –101.

[Kaltenbrunner and Bencina 2007] Kaltenbrunner, M., and Bencina, R. 2007. reactivision: a computer-vision framework for table-based tangible interaction. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, ACM, New York, NY, USA, 69–74.

[Kang et al. 2004] Kang, J., Cohen, I., and Medioni, G. 2004. Object reacquisition using invariant appearance model. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 4*, IEEE Computer Society, Washington, DC, USA, 759–762.

[Laws 1980] Laws, K. 1980. *Textured image segmentation.* PhD thesis, University of Southern California.

[Lu et al. 2003] Lu, S., Metaxas, D., and Samaras, D. 2003. Using multiple cues for hand tracking and model refinement. In *In International Conference on Computer Vision and Pattern Recognition*, 443–450.

[Mallat 1989] Mallat, S. G. 1989. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell. 11*, 7, 674–693.

[Monnet et al. 2003] Monnet, A., Mittal, A., Paragios, N., and Ramesh, V. 2003. Background modeling and subtraction of dynamic scenes. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, IEEE Computer Society, Washington, DC, USA, 1305.

[Nirei et al. 1996] Nirei, K., Saito, H., MOCHIMARU, M., and OZAWA, S., 1996. Human hand tracking from binocular image sequences.

[Paschos 2001] Paschos, G. 2001. Perceptually uniform color spaces for color texture analysis: An empirical evaluation. 932–937.

[Piazza and Fjeld 2007] Piazza, T., and Fjeld, M. 2007. Ortholumen: Using light for direct tabletop input. *Horizontal Interactive Human-Computer Systems, International Workshop on 0*, 193–196.

[Piper et al. 2002] Piper, B., Ratti, C., and Ishii, H. 2002. Illuminating clay: a 3-d tangible interface for landscape analysis. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 355–362.

[Rehg and Kanade 1993] Rehg, J. M., and Kanade, T. 1993. Digiteyes: Vision-based human hand tracking. Tech. rep., Pittsburgh, PA, USA.

[Rittscher et al. 2000] Rittscher, J., Kato, J., Joga, S., and Blake, A. 2000. A probabilistic background model for tracking. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, Springer-Verlag, London, UK, 336–350.

[Shafique and Shah 2005] Shafique, K., and Shah, M. 2005. A noniterative greedy algorithm for multiframe point correspondence. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 1, 51–65.

[Song et al.] Song, P., Winkler, S., and Tedjokusumo, J. A tangible game interface using projector-camera systems.

[Song et al. 1996] Song, K., Kittler, J., and Petrou, M. 1996. Defect detection in random color textures. 667–683.

[Stauffer and Grimson 2000] Stauffer, C., and Grimson, W. E. L. 2000. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 8, 747–757.

[Terzopoulos and Szeliski 1993] Terzopoulos, D., and Szeliski, R. 1993. Tracking with kalman snakes. 3–20.

[Wren et al. 1997] Wren, C. R., Azarbayejani, A., Darrell, T., and Pentland, A. P. 1997. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell. 19*, 7, 780–785.

[Wu and Huang 1999] Wu, Y., and Huang, T. 1999. Capturing articulated human hand motion: A divide-and-conquer approach. 606–611.

[Yilmaz et al. 2006] Yilmaz, A., Javed, O., and Shah, M. 2006. Object tracking: A survey. *ACM Comput. Surv. 38*, 4, 13.

[Zhong and Sclaroff 2003] Zhong, J., and Sclaroff, S. 2003. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, IEEE Computer Society, Washington, DC, USA, 44.