Sign Language Recognition

by

Nicholas Westlake (CAI)

Fourth-year undergraduate project in
Group F, 2011/2012

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: _____ Date: _____

Technical Abstract

Introduction

In this report, I present the development of a system for recognising individual signs in sign language when performed in front of a webcam. The purpose of this is to enable signs to be looked up quickly in a sign language dictionary, particularly for signers (sign language users) who may know how to perform the sign but don't know the equivalent word in a written language.

Such a system would greatly assist the translation effort of written media into sign language, through assisting native signers (typically those born deaf) to work on translations. This is required for creating high quality sign language translations for a number of reasons, outlined in the report.

Progress on the System

Significant progress has been made on the system. In particularly the following components have been developed:

1. A pixel based skin segmenter — the skin segmenter is based on Jones (2002) and uses two histograms to model the respective probability distributions of the RGB values of skin pixel and all pixels based on training images. The probability of each pixel being skin can be determined using these distributions. The skin segmenter runs in linear time and has achieved a 97.6% success rate.

2. An edge-based template matcher — the edge-based template matcher uses an implementation of oriented chamfer matching based on Liu (2010) to identify hand sign shapes. The system stores templates as a series of straight lines and exploits the use of integral images to achieve matching performance in linear time for the total number of straight lines across all templates. The system has achieved a 40% success rate for a series of hand gestures on different backgrounds; further work is required to make the system more robust for suitability for the aim of the project.

3. A finger tip classifier — the classifier uses the skin probabilities output from the skin segmenter as features and a boosting classifier to identify finger tips in an image. The system has been integrated into an application designed to allow a

user to type on a "paper keyboard" in front of a webcam. The classifier performs very reliably and, along with the skin segmenter, allows the application to run in real-time.

4. A machine learning based edge detector — the detector is based on Dollar (2006) and uses an 8 by 8 window, Haar features and gradient features (obtained using the Schar operator), and a boosting classifier to identify edges in images. The detector learns the edges based on a set of images and corresponding edge masks. For the purpose of detection object outlines, the system slightly outperforms the Canny edge detector, achieving a 95.7% accuracy rate against the 92.3% accuracy rate of the Canny edge detector, but is not robust enough to use as an input to the chamfer matcher due to gaps in lines. Further work is proposed to achieve this.

Further Work

The following suggestions are made in order to improve the achieve the aims of the project:

- Use the skin segmenter to remove background clutter from the input image of the edge-based template matcher
- Use the normalisation approach used by Ma (2010) to improve the template matcher
- Use the finger tip classifier, and develop a palm classifier, to transform the detected hands images so as to line up with that in the templates of possible signs before using the edge-based template matcher
- Use a larger window and more features to improve the operation of the edge detector
- Develop a 3D parametric model of the hand as used in Stenger (2004)
- Implement a classifier that works on the motion of hands
- Tie the system together to allow signs to be trained on a single video of each sign and to classify signs based on a single performance in front of a webcam

Contents

1 Introduction

1.1 Motivation for the Project

1.1.1 The need to convey information to a wide audience modern society

Modern society has been built upon the ability to convey information to a wide audience. From academic papers to railway timetables, we depend on being able to obtain the information we need, and to convey information others need or want.

The conveying of such information takes place almost exclusively through written word and verbal broadcast (radio and television). This presents a considerable problem for deaf people: not only are they unable to hear the verbal broadcasts, but they struggle to learn to read and write in the written language of the society in which they grow up. Because they rely on lip-reading to learn the nation's spoken language, they severely struggle and tend to leave school with the reading age of an infant, for example a 1977 study demonstrated that deaf school-leavers had an average reading age of 8 to 9 years [11].

1.1.2 The problems involved in conveying information to a wide audience through sign language

Contrary to common belief, sign languages are languages in their own right, independent from the spoken language(s) in the same country in which the sign languages are commonly used. They exhibit all the features of spoken languages such as their own unique "vocabulary", grammar and methods for conveying additional meaning (akin to tone, emphasis and body language in spoken languages).

Modern sign languages came into existence when deaf people started being educated together at schools and began to communicate to each other through signs and facial expressions. Over time, a consensus of signs and expressions for conveying meaning emerged creating a sign language: a language independent of the nation's official spoken language [33][26].

This has two major consequences:

1. Just as there is often no "one-to-one" mapping between words or characters in two spoken languages, so there is often no "one-to-one" mapping between signs in a

given sign language and words or characters in a given spoken language.

2. There is no formal way of conveying the sign language in written form — storing or conveying information to a large audience through sign language is impossible in the absence of video storage and transmission.

1.1.3 The use of computers to make this possible

Considerable effort has been made in the past and indeed is still being made to translate written publications such as educational materials, farming, nutrition and health manuals and scripture into other written languages of the world. Until recently it has been impossible to carry out similar effort for sign languages due to the aforementioned problems. However computers, with their ability to capture and store large volumes of quickly accessible images and video, have made this viable. In particular, the internet has made it possible for long-distance collaboration for such projects.

One particular issue still remains — the lack of "dictionaries" for sign languages. This has hampered the effort to effectively translate material into sign languages. Having a bi-lingual dictionary is particularly important for allowing native signers with reduced spoken language ability to participate in the process. Their participation is vital for three reasons:

1. Native signers are best placed to create translations which convey the information correctly since they have a full command of the language.

2. Knowledge of some signs may be limited to native signers.

3. Other native signers can tell if a signer is native or not, and are far less likely to be willing to view media produced by a non-native signer.

SIL International [1] are attempting to address this through development of a desktop sign language "dictionary" application (see figure 1.1) that allows signs to be recorded in video format and then tagged based on the features in the sign, such as motion and standard hand-shapes. This makes the searching of signs in a dictionary by sign possible, though cumbersome.

[1] http://www.sil.org

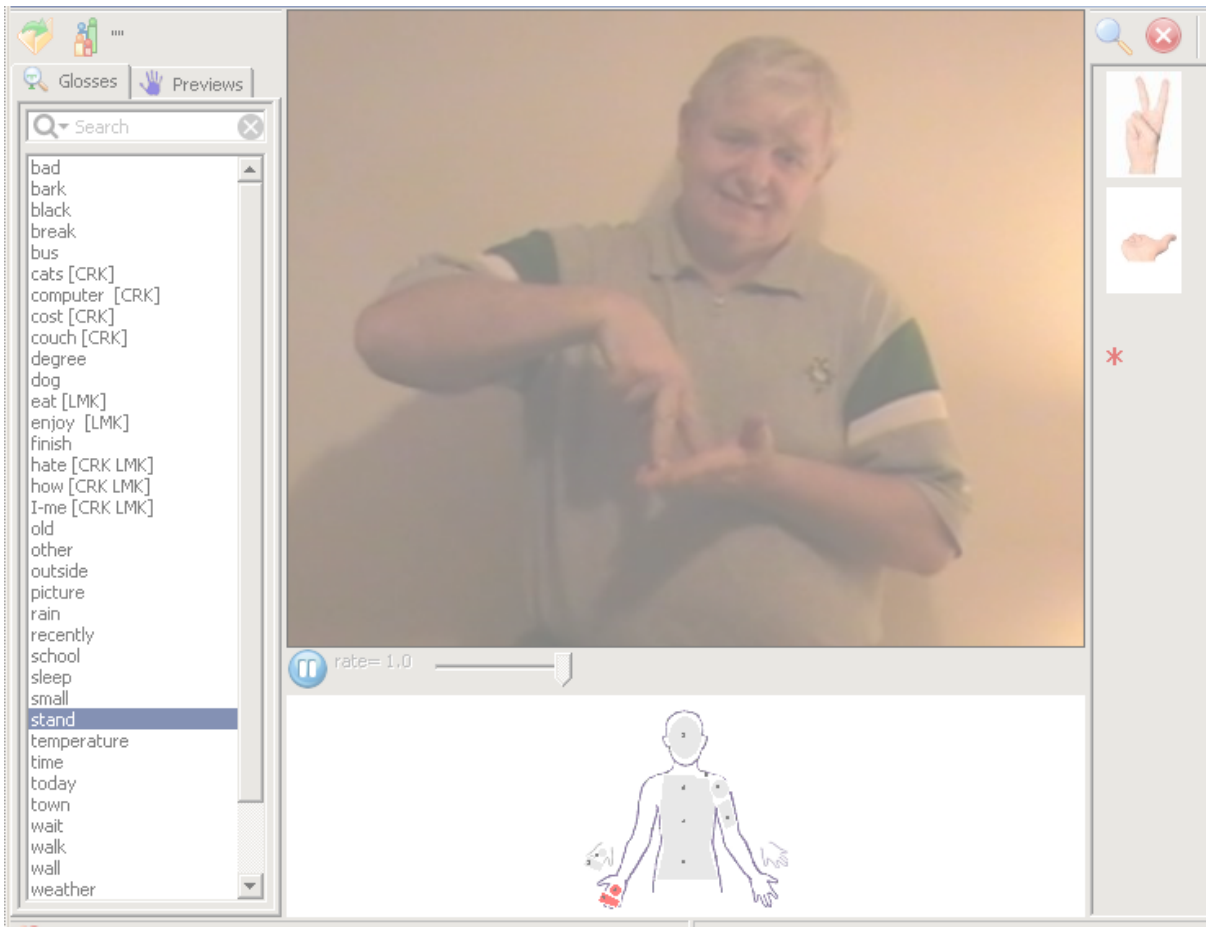


Figure 1.1: SIL's existing sign language dictionary application

1.2 Aim of the project

The previous section (1.1.3) mentioned the development of computer sign language "dictionaries" using stored videos of sign languages. As far as I am aware, all such systems only allow searching for a sign using the nearest equivalent word or phrase in a spoken language, or by standard features of the sign. Ideally, however, a signer should be able to search for a sign by performing in front of a webcam (as shown in figure 1.2[2]), and this forms the basis of my project.

More specifically the system should be able to achieve the following:

- Recognise signs based on a limited number of videos of a given sign (possibly only one) to allow the dictionary to be expanded without an excessive amount of participation by different signers

- Operate based on videos and live input from a normal (2D) webcam to allow the system to be used without expensive hardware (such as the Kinnect™[3])

[2]Modified Creative Commons picture by *SignVideo, London, U.K.*
[3]http://www.xbox.com/en-US/kinect/

Figure 1.2: An demonstration of how a user would use the system

- Not require users to wear any particular garments or stickers when using the system to increase participation

2 Overview of the System

2.1 System Components

The overall task of recognising sign language can be split into multiple sub-tasks, which must be performed:

Segmentation

Segmentation, in the context of computer vision, refers to the grouping of related pixels in an image and the partitioning of parts of the image [17, p. 301-302]. Within the project, the following tasks require a form a segmentation:

- Separating skin pixels from non-skin pixels in an image
- Identifying individual hands in and image
- Identifying the outline of the hands in an image

Template matching

Template matching refers to identifying instances of an object (or part of an image) that is known to fit a particular pattern (the template) [17, p. 495-496]. Within the project, the follow tasks require a form of template matching:

- Identifying whether a given hand-shape is present (i.e. any hand in the image is forming a particular shape)
- Identifying individual parts of a hand, such as finger tips, in an image

Motion Classification

Motion classification is the identification of or application of labels to motion of an object, e.g. "moving up" "moving side-to-side", "circular" etc. Identifying the motion of each individual hands through a sequence of images (frames) would use this.

A finished system would likely use segmentation initially to identify the hands in the picture. This could involve multiple stages such as skin-pixel detection initially and then indication of location (or hypothesised locations) of the individual hands.

Template matching would then be carried out, possibly directly on the outline of the hand, or on skin pixels to identify parts of the hand such as the palm, fingers and finger-tips. A multi-layer approach could be used, with template matching first used to

identify the actual locations of each hand from the hypothesised locations and then used to identify the sign, if any, formed by the hand. Alternatively, individual parts of the hand or the outline of the hand could be detected based on the output of the skin-pixel classifier, and used for template matching of signs.

Motion classification would take place on the identified location of the hand, or locations of individual parts of the hand. The motions of the hands, and fingers relative to the hands would be labelled.

The output of the template matchers and motion classifier would then be used to identify the sign, or hypothesise possible signs, using a classifier (such as a decision tree) or model (such as a hidden markov model).

2.2 Summary of Progress to date

To date I have produced the following components:

1. A pixel-based skin segmenter for identifying which pixels are likely be to skin pixels.
2. An edge-based template matching system (using chamfer matching) for detecting individual hand signs.
3. A real-time finger-tip detection system.
4. An application allowing a piece of paper, with a printed keyboard on, to be used as a keyboard to demonstrate the performance of the finger-tip detection system.
5. A machine-learning based edge classification system to provide a cleaner edge input for the edge-based template matching system.

3 Skin Segmentation

3.1 Available Segmentation Techniques

Multiple methods are available for segmentation. They can be divided into three categories:

Cluster based

Cluster based methods work by either recursively dividing larger groups of pixels into smaller groups of pixels (divisive clustering) or joining smaller clusters together into larger clusters (agglomerative clustering). The former begins with all pixels forming a single cluster and the latter with each individual pixel forming a cluster [17, p. 301-302]. An example of such a method is the use of K-means algorithm [31]. Such methods are particularly good for finding clusters in an image without prior knowledge of what the image contains.

Model based

Model based methods involve fitting a model, or multiple models, to an image [17, p. 329]. An example of such a model includes the Hough transform which can identify shapes such as lines and circles in an image [22].

Probabilistic

Probabilistic based methods seek to explain the data using a global model and fitting the parameters so as to maximimise a probability (such as the likelihood in Maximum Likelihood estimation)[17, p. 354]. In the context of skin segmentation, the global model indicates which pixels are skin and which are not.

The majority of skin segmentation methods are probabilistic and a full discussion of such methods is available in [24]. Roughly speaking, the methods divide between those that operate individually on each pixel and those that operate on the image as a whole. The former operate by sweeping over the image, pixel by pixel and hence run in linear time. They classify each individual pixel based on the pixel colours values itself, or features extracted within a window around the pixel (e.g. [23] as used below and [12]). The latter typically operate by finding a model that maximises the probability formed by combining two or more of the following probabilities (often expressed as an energy or cost function for convenience): individual pixel probabilities, the probability based on

local texture and presence of nearby edges and the probability of the model based on the number of transitions between skin and non-skin pixels (e.g. [20], [5], [32] (as used in MS Office^{TM}), [25] and [4]).

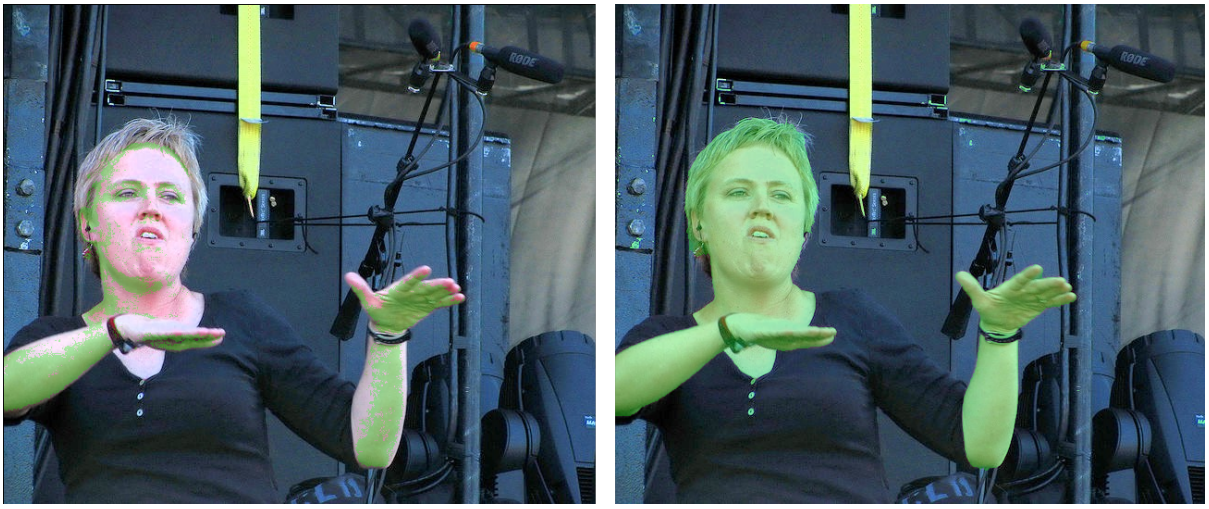


Figure 3.1: A comparison of the outputs of a pixel-based segmentation algorithm (left) and one that operates on the whole image (right): the green overlay indicates areas classed as skin

Figure 3.1[1] demonstrates the difference in output between a pixel-based segmentation algorithm (based on [23] and outlined in section 3.3) and one that operates on the image as the whole (an implementation of GrabCut [32]). The former has correctly classified most of the arm and almost all of the background, but has performed badly on the face and lighter parts of the arm (where ambiguity exists between skin and other light surfaces). The latter has correctly classified the arms and the face, but has misclassified the hair, facial features, watch and many parts of the background. On the whole the latter appears better.

qWhile the large number of misclassified pixels in the output of the formal algorithm appear to make it unsuitable for recognition, it should be noted that the image displays the thresholded output, while the pixel based probabilities are almost certainly larger for the ambiguous areas as for the background. A robust classifier will be able to use this to correctly identify parts of the body such as hands and the face in the image (as demonstrated with finger tips in section 5). Indeed, the output of the pixel-based classifier was used as the input for the GrabCut algorithm [32] used here since an initial hypothesis of the foreground and background components must be specified for the algorithm to work.

[1]Creative Commons image by *incendiarymind* used as input

3.2 Choice of Method for Skin Segmentation

Although the methods operating on the whole image typically result in a better output (as demonstrated in figure 3.1), in particular one that yields better well-defined regions and much less or no noise, they require many iterations or maximisation steps to run and hence run in polynomial time. Having well-defined regions and less noise is important in applications such as movie editing but is less important for object recognition tasks as a good classifier can function with an imperfect or noisy input.

Since the system uses requires skin segmentation for recognition, namely for recognising hands and individual parts of the hands and will operates on multiple frames of a given video, I considered it more desirable to use method resulting. I chose to use one of the former, pixel based methods. This enables the resulting skin segmenter to run in real-time (see section 5.5).

Of the methods that operate on individual pixels, the one demonstrated by Jones (2002) [23] appears to offer the best performance [24]. Therefore I chose to implement this algorithm.

3.3 Operation of the Skin Segmentation System

The skin segmenter I have implemented is based on [23]. It classifies individual pixels based on the probability of each pixel being skin. This is calculated by using two discriminative models, one modelling probability density of the colour (in the RGB colour space) of skin pixels, $p(\text{rbg}|\text{skin})$, and another modelling that for all pixels, $p(\text{rbg})$.

The histogram method (see [3, p. 120-122]) is used to model the two probability distributions with a uniform histogram bin size. This enables the probability density estimates to be obtained in constant time [35, p. 15-17], though at the expense of $O(N^3)$ memory requirements where N is the number of histogram bins in each dimension of the RGB colour space. A uniform bin size also means that volumes of the colour space occupied by few pixels offer an inaccurate estimate of the probability distributions while volumes occupied by many pixels offer an imprecise estimate [35, p. 15-17]. The former is less of an issue if a sufficient number of training images are used and the latter is not too great an issue since the probability distributions of colour in images tend not to have discontinuities.

The RGB colour space is used without any normalisation since the intensity (mean of the RGB values) provides a greater level of discrimination between skin and non-skin pixels [23]. The variance in intensity values due to different lighting conditions is modelled well by the histogram model if a sufficient range of training images under different lighting conditions are used. The lack of normalisation also prevents the pixel colour values from parts of image which are overexposed or underexposed (seen as "colourless" white or dark spots in images) from affecting the probability distribution estimates of colour values

under better lighting conditions.

Training of the models is carried out using a series of images with skin and backgrounds masks indicating the locations of skin in the image. Pixels colour values are then added to the respective 3D histograms. The estimate of the probability density of a given colour is then calculated as follows:

$$p(\text{rgb}) = \frac{n(\text{rgb})}{T_n} \tag{3.1}$$

where $n(\text{rgb})$ is the number of observations in the histogram bin corresponding to the given RGB value, and T_n is the total number of observations in the histogram.

Classification of an individual pixel is then performed by looking up the pixel value in both histograms. The posterior probability of the pixel being skin is calculated as follows:

$$p(\text{skin}|\text{rgb}) = \frac{p(\text{rgb}|\text{skin})}{p(\text{rgb})} p(\text{skin}) \tag{3.2}$$

where $p(\text{rgb}|\text{skin})$ and $p(\text{rgb})$ are obtained from the respective histograms using equation 3.1 and $p(\text{skin}) = \frac{T_{\text{skin}}}{T_{all}}$ where T_{skin} and T_{all} is the total number of observations in the histogram of skin pixel observations and both histograms respectively.

A pixel is classified as "skin" if the probability of it being so according equation 3.2 is greater than 0.5. Alternatively the image of skin probabilities can be used as features for a classifier with suitable windowing, as demonstrated in section **??**.

3.4 Performance of the Skin Segmentation System

The performance of the skin segmenter was tested against a dataset provided by Boston Univesity Image and Video Computing Group[2]. The dataset consists of 21 video sequences, in which every fifth frame contains two masks: one labelling pixels as skin and one labelled as background (pixels labelled as neither skin nor background are considered ambigious and not used in the training or testing of the system). Figure 3.2 shows one of the training images with the masks overlaid.

Figure 3.2: One of the images used for training the skin segmenter with the skin mask (green) and background mask (red) overlaid

[2]http://csr.bu.edu/colortracking/pami/Data/

To assess how the system would be likely to perform with unseen video sequences, I trained the system using the first 17 sequences and then tested the system using sequences 18 to 21. The resulting pixel classifications were compared against the skin masks. Table 3.4 shows the result of the performance test.Figure 3.3 demonstrates the output of skin segmentation.

The segmenter achieved 97.6% accuracy; the false positive rate was very low, at 0.3%, but at 2.1% the false negative was larger than would be desired and is particularly noticeable in the results of the algorithm. The false positives results are noticable as patches of background with other skin coloured objects such as the brown tree trunk marked as skin. The false negative results are visible as gaps in the the skin area and tend to be prevalent in areas where the skin has been exposed to too high or too low lighting. Here the system struggled because the white glows and darker patches looked too similar to other poorly lit or overly exposed objects.

Classification	**Pixel count / %**
Correctly identified as skin	365,342
Incorrectly identified as skin	44,883
Correctly identified as non-skin	15,652,899
Incorrectly identified as non-skin	349,307
Correctly identified	97.6 %
False positive	0.3 %
False negative	2.1 %

Table 3.1: The performance of the skin segmentation system

Overall, the system performs well given that it operates on individual pixels. The system could be improved, but to achieve a significant improvement would segmenter would have to operate on the whole images and would run much slower; such a system would probably not be able to run as part of a real-time application (as demonstrated in section 5).

Figure 3.3: Demonstration of skin segmentation: top, the original image; middle, the labelled skin mask (ground truth); bottom, output of the skin segmenter

4 Edge-based template matching system

4.1 Available template matching techniques

A number of different techniques for template matching are available. In order for a given technique to work effectively for hand shape recognition the following criteria must be satisfied:

1. Since the surface of the hands has very little variation in colour or intensity, and most of the perceived variation in appearance is due to lighting conditions, the technique must be not be dependant on detecting variations within the hand.

2. Since the hands will have a varying background, depending on what is visible behind them (as demonstrated in figure 4.1), the technique must be robust to this.

Figure 4.1: Techniques for hand shape recognition must be robust to the variation of the background as the two images of the same hand in different positions illustrate

A number of techniques are outlined below, along with a brief discussion of their possibility for use for recognising signs formed by hands based on the requirements outlined above. The discussion focuses on features rather than classification algorithms (such as a support vector machine or boosting).

Corner Matching

Corner detectors such as Harris (1987) [21] can be used to identify corners in an

image. The corners detected in an image can then be matched against a template of corners. The hand does not contain any distinct corners due to its curvature, and so the use of corner detectors for this purpose is prohibited.

Edges

Edge detectors such as Canny (1986) [9] can be used to extract the edges in an image. The edges in an image can then be matched against an template of edges. The edges capture the outline of the hand, along with the boundaries between objects in the background and larger changes between intensity within the hand (eg due to lighting conditions or veins). Therefore, with further processing, an edge detector is suitable for detecting hand shapes.

Haar Features

Haar features are calculated based on the difference between the sum of pixel intensity values between two or more rectangular regions (see section 7.3.1 where their use is discussed for edge detection). A classifier can use the haar featues at a given location and orientation to detect matches based on the haar features in the template. While they have been successfully used for face detection and recognition, the poor amount of variation within the hand would prevent their use.

Normalised cross-correlation

The normalised cross-correlation can be used to detect the similarity between two patterns [14]. Many algorithms exist for using the normalised cross-correlation for matching images or images patches, such as [41]. The normalisation step reduces the effect of changes in lighting conditions on the performance. Such methods would not be robust for hand shape recognition due since variation of the background would affected the normalised cross-correlation too greatly.

Scale-invariant features

Scale-invariant features such as the scale-invariant feature transform algorithm [28] and the speeded up robust feature algoritm [2] can be used to identify and provide descriptions for "features" within images that match well across different images, even if the different images show the feature at different scales, orientations and lighting conditions. However, the methods require large variations within objects in order to accurately locate features between images, and hence are unsuitable for hand shape recognition.

As noted above the only effective techqiue is edge-based matching, and this method would therefore be used for the purpose.

In order for an edge-based template matching algorithm to work effectively for hand shape detetections, it must be robust to small changes in the edge location as figure 4.2

illustrates. An example of such as system is chamfer matching, which is outlined in the next section.

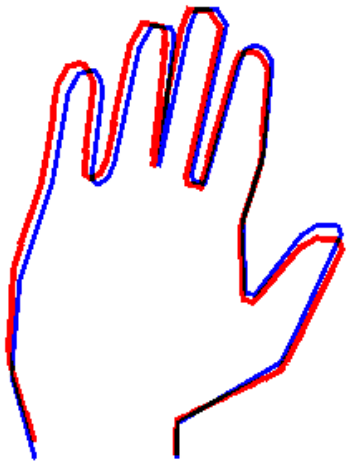


Figure 4.2: The red template clearly matches the blue edge outline but the edges do not overlap in the majority of locations due to minor variations. To be effective, an edge-based template matching system must be robust to such variations.

4.2 Outline of Chamfer Matching

Chamfer matching has been widely exploited in computer vision for matching objects from 2D images [1]. A Chamfer matcher calculates a cost of a given template (set of edges or edge pixels) against a given location in an image based on the closeness of the match. The exact cost function depends on the implementation. To find the best match in an image, the match cost of a given template must be calculated at all locations. For scale and orientation invaiance, this must be repeated across all possible scales and orientations.

The original chamfer matching algorithm [39] uses an edge detector to detect which pixels form edges in an input image (forming an edge image) and matches a template formed of edge pixels against the input edge image. The value of the chamfer distance of the nearest edge pixel in the input edge image is summed for each pixel in the template image yielding the cost of the match. This occurs for every location in the input image, and the best match is the location which yields the lower cost. The best match is then considered an actual match if the cost of the match low enough. The use of the chamfer distance rather than the euclidean distance allows the the cost of determining the distances to be used using distance transforms (see section 4.3.2).

This algorithm suffers poor performance when matching against background clutter, since templates often match against background clutter better than a correct match due to the large number of edges in background clutter. The algorithm can be improved by using additional information, such as the orientation of edges.

4.3 Operation of the Implemented Chamfer Matching Algorithm

The Chamfer Matching algorithm I chose to implement is based on Liu (2010) [27]. The algorithm generates a cost between a template and input edge image based both on the chamfer distance and orientations of edge pixels, at a given location in the image, roughly equal to the following:

$$C = \frac{1}{n} \sum_{\mathbf{u_i} \epsilon U} \min_{\mathbf{v_i} \epsilon V} \left(|\mathbf{u_i} - \mathbf{v_i}| + \lambda \left| \phi \left(\mathbf{u_i} \right) - \phi \left(\mathbf{v_i} \right) \right| \right) \tag{4.1}$$

where n is the the number of edge pixels, U is the set of all the edge pixels in the template, V is the set of all the edge pixels in the image, with co-ordinate system relative to the top left of location in the input image against which the template is being matched, λ is an arbitrary scalar that adjusts the relative cost of a differecet in orientation against the cost of a unit pixel distance and ϕ is a function that maps the orientation of the edge at the edge pixel into an integer value (i.e. mapping the full 180°of available orientations into N evenly spaced bins).

The algorithm exploits integral images to increase the efficiency of chamfer matching compared to similar implementations that include orientation in the cost calculation. The algorithm matches straight line segments rather than edge pixels and runs in linear time for the number of line segmen rather than linear time for the number of edge pixels. This enables it to run faster that previous chamfer matching algorithms.

4.3.1 Line Segment Finding

The first step of the implemented algorithm is to locate the line segments forming the edges in the input image. As with the implementation described by Liu (2010) [27], I use the Canny Edge Detector [9] (as implemented in OpenCV) to obtain an edge image. The implementation described by Liu (2010) then uses an unspecified version of RANSAC (see [16]) to determine line segments from the edge image. RANSAC, however, removes much of the fine detail, e.g. the curves around the fingers and so I developed my own algorithm. As far as I am aware, this algorithm is not documented elsewhere.

The operation of the algorithm is documented in algorithm 1. The algorithm scans across the image in reading order (from left to right, top to bottom) looking for three connected edge pixels. Once three connected edge pixels are found, the algorithm moves along connected edge pixels to find the best fitting combination (straightest) of three edge pixels. The algorithm then extends the line segment by adding the best fitting pixel (pixel which leads to the straighest line) at either end until the mean squared error of the last three pixels, against the line of best fit for all the added pixels, is greater than one unit pixel. The line segment is then stored, and the edge pixels corresponding to the

Algorithm 1 A description of the line segmentation algorithm in pseudo-code

```
for all pixels in the image looped over from left to right line by line from top to bottom do
    if pixel is edge pixel then
        Add pixel to an empty ordered pixel set
        if at least two adjacent edge pixels then
            Add the two neighbouring edge pixels to opposite ends of the ordered set
            while mean squared error reduced do
                Add the best fitting (using linear regression) neighbouring pixel to the correct end of the set
                Remove the pixel at the other end of the set
            end while
            Undo one step to yield the best fitting (straightest) three pixels
            while more adjacent edge pixels not added to the set do
                Add the best fitting neighbouring pixel at either end of the set to the correct end of the set
                Calculate the line of best fit
                Calculate the mean squared error of the last three pixels added at the same end as the last added pixel
                if calculated mean squared error > one unit pixel length then
                    Remove the edge pixels from the edge image and store the line segment
                    Break out of while loop
                end if
            end while
        else
            Remove (stray) edge pixel from the edge image
        end if
    end if
end for
```

line removed.

Linear regression using least mean squares of perpendicular distance is used to find the best fitting neighbouring pixel: the pixel which, in addition to the existing pixels, produces the lowest mean squared error. To store a line segment, the orientation is calculated from the gradient output of the linear regression (constrained to be between 0°and 180°). The orientation is mapped to an integer so that only a finite number of orientations are possible (as required by the implementation). The start and end of the line segment is calculated as the intersection between the calculated line of best fit and the perpendicular lines passing through the start and end pixels.

The algorithm is linear in algorithmic complexity. This can be proven as follows:

- The algorithm scans across the image initially testing each pixel once. This scanning operation is therefore $O(N)$ where N is the number of pixels.

- When the algorithm discovers an edge pixel while scanning, it processes the connected edge pixels. However, it will make at most two visits to each edge pixel before the edge pixel is removed: firstly, while moving along connected edge pixels looking for the best fitting three edge pixels; secondly, while extending the line by adding edge pixels to either end. Across the entire image, both these operations will therefore also be $O(N)$.

- Finding the best fitting neighbouring pixel at every stage involves examining a fixed number of pixels at each end of the line, invariant of the line size. This is therefore a $O(1)$ operation[1].

Therefore the operation is $O(N) + O(N)O(1) = O(N)$ in algorithmic complexity.

[1]This is not quite true, as the linear regression operation, called for each new pixel, runs in linear time according to the number of pixels. However, this could be made to run in constant time by adding and subtracting from cached results. This would, however, result in a reduction in a decrease in the precision of the fit due to the increased error from combinding addition and subtraction operations.

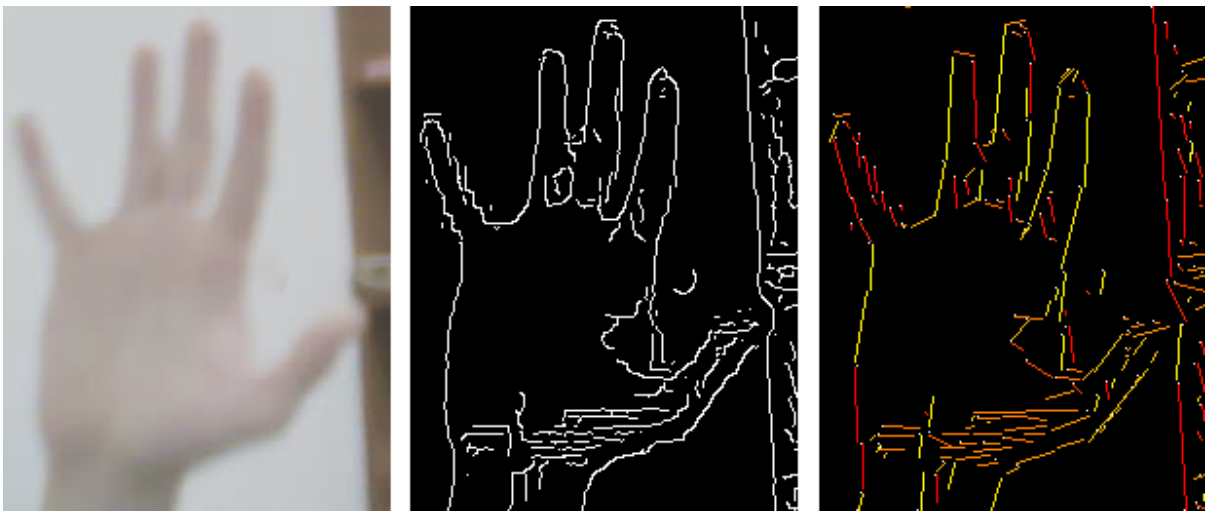


Figure 4.3: From left to right: the image input into the line segmenter; the edge image produced after processing with the Canny edge detector; the output of the line segmentater

Figure 4.3 demonstrates the output of the line segment finder. The colours of the line segments refer to the orientation of the line segment, while white pixels are stray edge pixels which could not be formed into a line segment. There is noticeable straightening of curves, however much of the finer detail from the edge image is preserved. In some places such as the right hand side of the left most finger, a less local edge finder would be more suitable and would correctly result in one single joined line. However, such an algorithm would be unlikely to render the area around the bases of the other three fingers correctly. The fact that the hand is still clearly recognisable in the rightmost image gives confidence for using the output of the line segment finder as the input to the next step.

The line segments extracted from the line segment finder can be stored as templates for matching a given set of edges. To match the templates against an input image, further procressing is required, as is outlined in the next two sections.

4.3.2 Production of Integral Images

To match the templates (formed by a combination of line segments) against the input images, integral images must be produced (one for each possible line segment orientation). Integral images calculate the integral or sum of each pixel up to a given point in a given direction along a image, across each of the possible lines in that direction. These allow the cost of a single line segment to be obtained by subtracting the value at the end of the line segment in the imput image from the value at the start of the line segment.

The following steps occur to produce the integral images for each orientation, from the line segments produced by the line segment finder:

1. The line segments are rendered (as pixels) in (floating point) pixel images, one for each possible orientation, based on the orientation of each line. Bresenham's line

$\sqrt{2}$	1
1	0

Table 4.1: The kernel used for the distance transform

algorithm [7] is used to draw each line. Initially the value for points lying on line segments is 0.0, and other points are set to a given maximum possible value (cap), representing points being 0 and "infinite" distance from an edge respsectively. This is the required input for a distance transform. To avoid the lack of a edge pixel at a small number of more places massively increasing the cost of an otherwise good match, the maximum possible value (cap) is set relatively lwo. Figure 4.4 demonstrates the output for two different orientations.

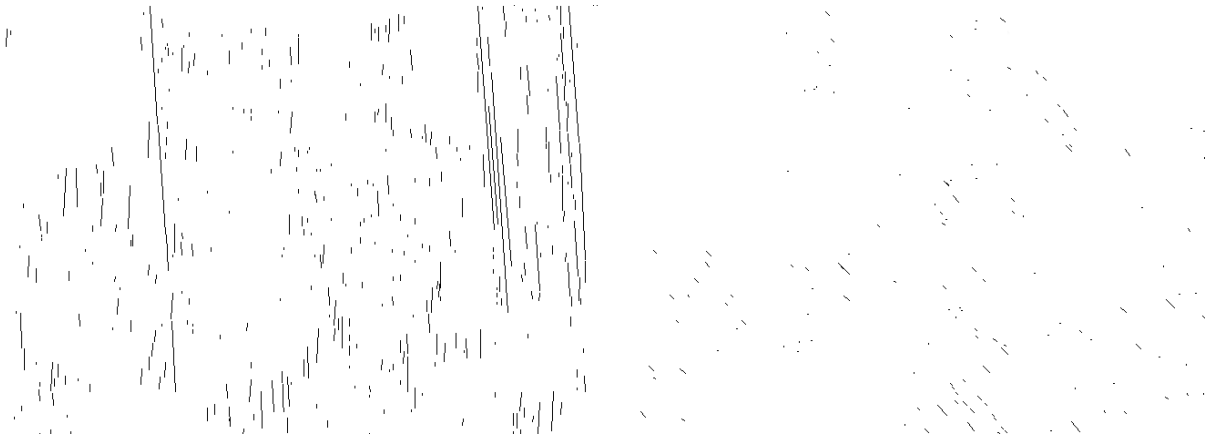


Figure 4.4: Images for two different orientations produced by rendering the respective line segments

2. The next step is to carry out a chamfer distance transform (see [15]) on each image. This performed using using two linear sweeps: one from left to right, top to bottom; and one in reverse. The kernel (table 2) is swept over the image, and the value of the pixel at the bottom right of the kernel location is updated with the smallest value of an adjacent pixel plus corresponding value in the kernel combined. The transpose of the kernel is used for the reverse sweep, with the pixel in the top left position being updated. This generates images in which each pixel's value is the chamfer distance in unit pixel lengths to the nearest edge pixel (or cap, whichever is smallest). The use of chamfer distance rather than euclidean distance makes the linear sweep possible, and is the reason for the name "chamfer matching".

   Figure 4.5 shows the result of the distance transform — the lines appear to have been "smoothed" over the image. The maximum possible distance has been set low enough as to result in much of the image being continuously white, as mentioned in the previous point.

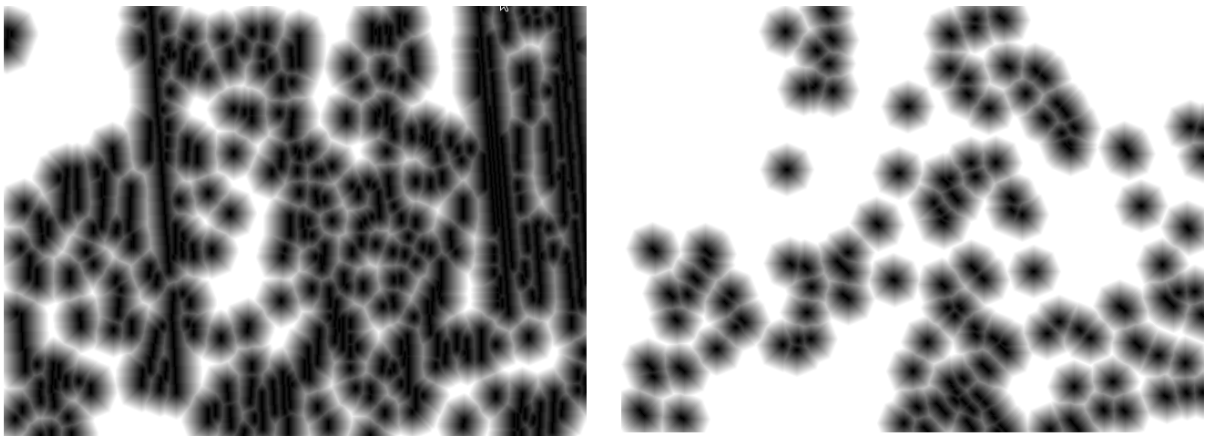


Figure 4.5: The distance transform images for the same two orientations

3. Next the algorithm generates an orientation distances transform. For each pixel location, across all the distance transform images, starting with the smallest orientation and progressing in order of orientation, the value of the pixel in the distance transform image is compared with the value of the same pixel in the distance transform image corresponding to the next smaller orientation (or looped around to the largest orientation). If this value plus the fixed cost, (λ above) is smaller, the pixel value is reduced accordingly.

   This continues across all pixels until one and a half sweeps have been made across all possible orientations. This process then occurs in reverse: from larger to smaller orientations, again with one and half sweeps. One and a half sweeps are required in each direction to ensure that the new images satisfy equation 4.1 when masked by a template, summed and normalised. (Fewer than one and a half sweeps will result in some pixels having too high a cost and continuing the sweep would not result in any pixel values being updated).

   Figure 4.6 shows the results of the orientation transform on the distance transform images — lines from other orientations can be seen in the image, at a higher value cost than in the original orientation distance transform image (i.e. whiter in the image representation).

4. The orientation distance transform images satisfy equation 4.1 when masked by a template, summed and normalised and so could be used for finding the cost of a given line segment in a given location by summing all the pixel values that lie along the line segment in the given location. However, as mentioned earlier, this process is speeded up by integrating along each orientation distance transform image. Integration is performed by summing along lines in the image on a pixel by pixel basis: the value each new pixel visited in orientation distance transform image is added to the sum and stored in the same pixel location in the intergration transform image.

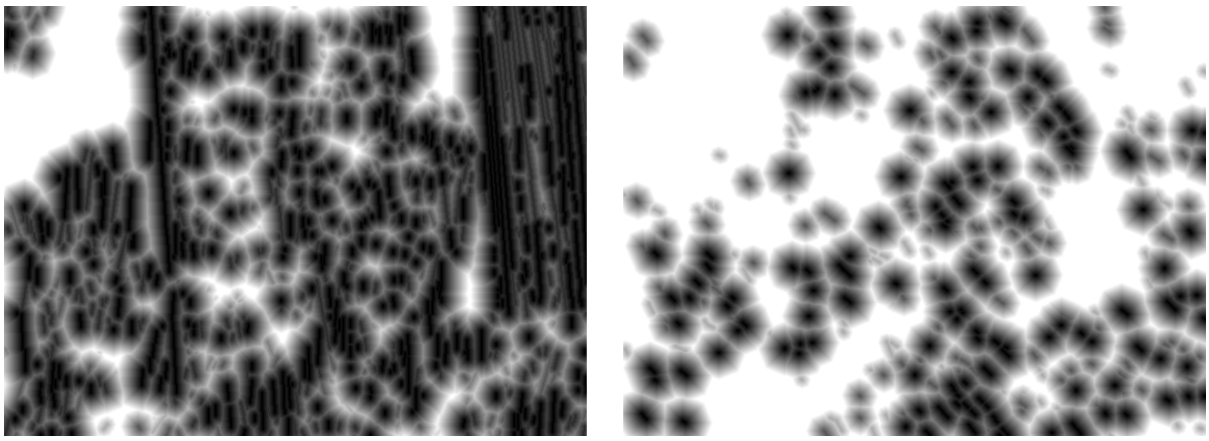


Figure 4.6: The orientation distance transform images for the same two orientations

Bresenham's line algorithm [7] is used to determine the path, since the lines produced tessellate when both the start point and end point of the line are translated along the axis furthest from the line in terms of angle (either the x or y axis depending on the angle of the line; both for 45°). For angles between 0°and 45°, and between 135°and 180°, the lines are drawn from top to bottom; for angles between 45°and 135°, the lines are drawn from left to right. The start and end points always line along the same line, hence the drawn lines will often start or end outside the image, in which case the algorithm simply continues without altering the sum result or storing any values. Figure 4.7 illustates this. Figure 4.8 shows the intergral images produced by integrating the orientation distance transform images.

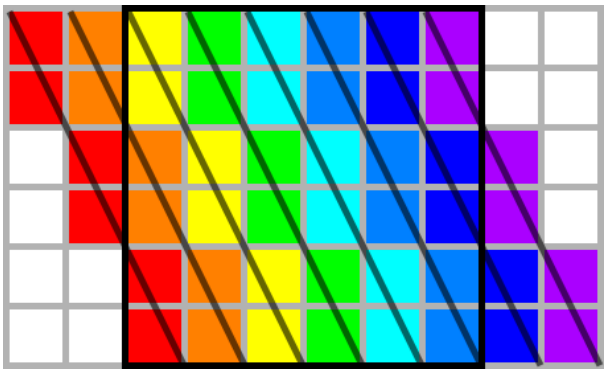


Figure 4.7: The different paths along which one orientation distance transform image is intergrated: the different colours indicate the different paths, and demonstate the fact that the lines tessellate; the image lies within the black square, while individual pixels are bordered with a light grey lines; the lines being approximated are overlayed as translucent black lines.

4.3.3 Template Matching Against the Intergral Images

As mentioned earlier the templates for the implemented chamfer matching system are simply a collection of line segments produced by the line segmenter; the cost of each

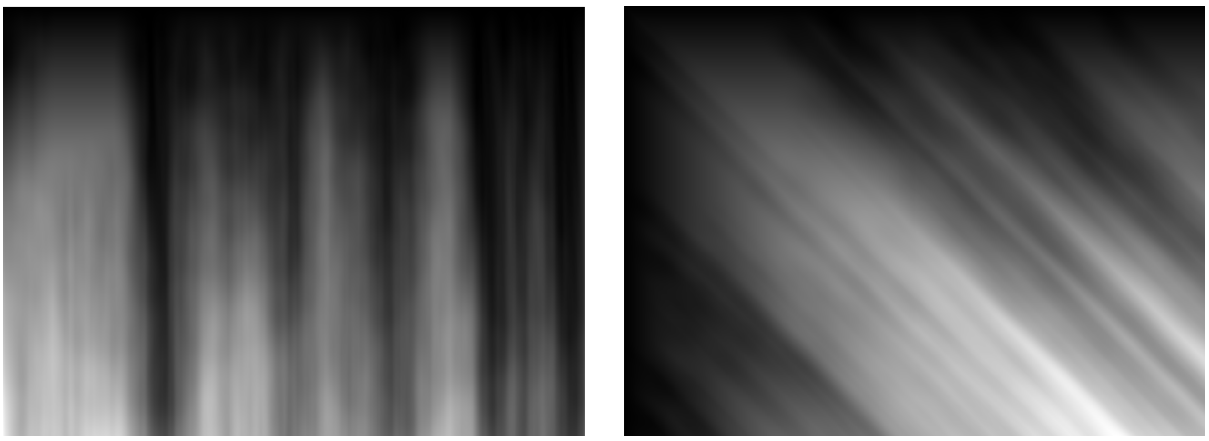


Figure 4.8: The integral images for the same two orientations

individual line segment can be obtained by looking up the value of the start and end points of the line segment in the corresponding intergal image for the orientation of the line.

To normalise the cost over all templates and scales, the sum of all line segment costs is divided by the sum of the line segment lengths. To achieve location and scale invariance, the algorithm carries this out for each template over a number of locations and scales as follows:

- The algorithm scales a template by scaling the start and end points of each line segment and rounding the value to an integer. The current implementation examines all scales between 0.4 and 3.0 at intervals of 0.2.

- For each scale, the template is matched at points along the image from left to right; top to bottom. Every row is examined, however the algorithm skips four pixels as it moves from left to right if a high cost (poor) match is detected. This reduces the number of operations required to find the best match [2].

4.4 Performance of the Edge-Based Template Matcher

The performance of the edge-based template matcher was tested against the Jochen Triesch Static Hand Posture Database [3]. The dataset consists of pictures of a number of hand gestures: each hand gesture is set against three different backgrounds: a blank (totally white) background, a dark "noisy" background and a complex background (containing vaious other objects), as figure 4.4 demonstrates. The exact pose of the hand

[2] With suitable data structures, pixels could be skipped if a poor match is yielded at a nearby location in two dimensions, rather than along the same row, to futher optimise the algorithm. Give or take some error introduced by the algorithm, the maxium decrease in cost in any direction should be equal to one unit pixel (when normalised), and this fact can be used to vastly reduce the number of locations attempted.

[3] http://www.idiap.ch/resource/gestures/

varies slightly between backgrounds for the same gesture, as does the rotation of the hand with respect to the camera position.

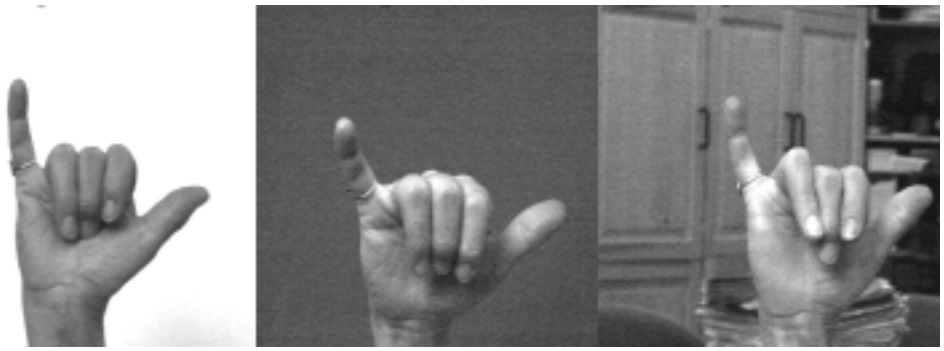


Figure 4.9: One of the hand gestures in the dataset set against the three respective backgrounds: from left to right; a blank background; a dark "noisy" background; a complex background

The images containing gestures against the blank background were used to form templates of each gesture (effectively training the system with a single image). The system then attempted to identify the gestures in the remaining images (those set against the noisy and complex backgrounds).

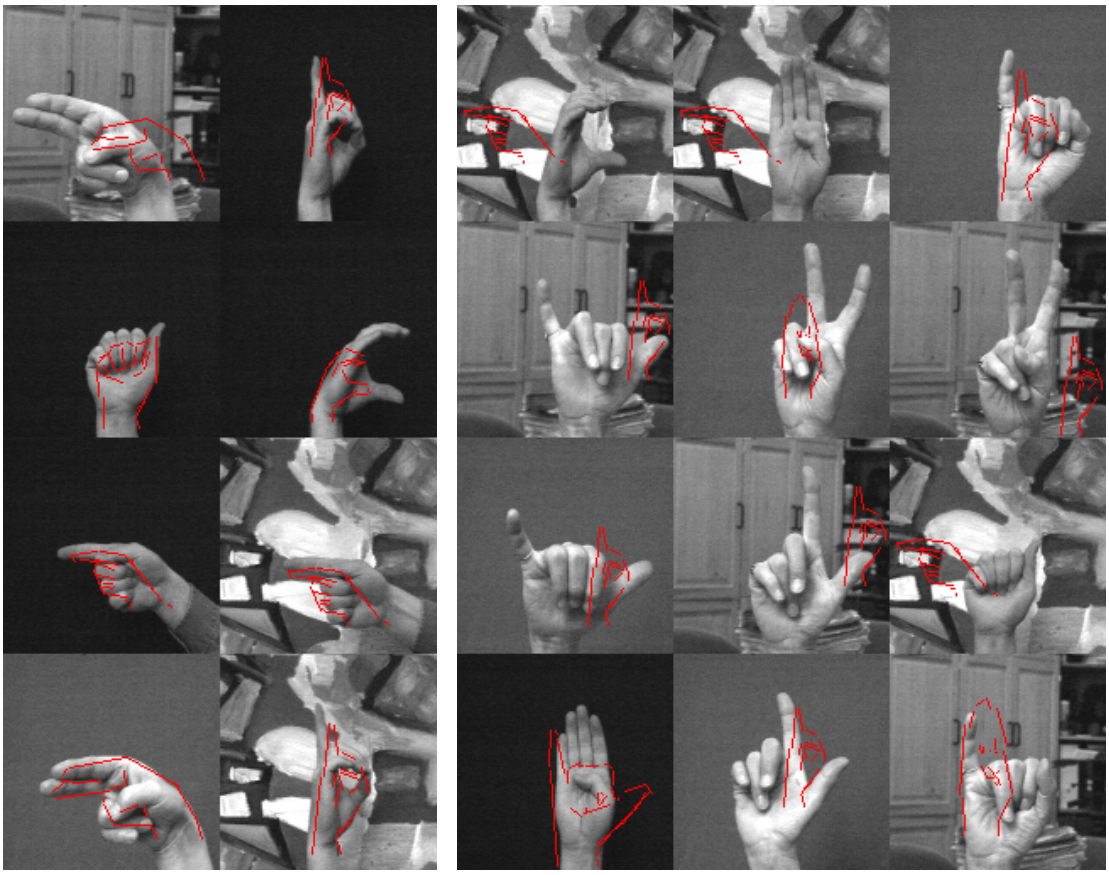
Table 4.2 shows the results of the attempted classification. The output of the edge-based template matcher, showing each image with the best matching template overlayed, is shown in figure 4.10.

	Number of gestures by background		
Result	**Noisy background**	**Complex background**	**Total**
Correct	5	3	8
Incorrect	5	7	12
Correct classification rate	50%	30%	40%

Table 4.2: The results of the edge-based template matcher attempting the classify the images in the gesture data set

The results show poor that the edge-based template matcher performed poorly, achieving only a 40% overall correct classification rate. This was caused by the slight differences in pose and orientation between the template image and test images: as a result another template matched better against edges found in the background clutter or the gesture. Even where a correct match was obtained, the gesture tended no to fit ever close to the template, again due to the slight difference in pose.

In order to be effective, the system needs to be more robust to the slight changes in orientation and pose demonstrated. Suggestions for how to achieve this can be found in section 8.2.2.



(a) Gestures correctly identified (b) Gesture incorrectly identified

Figure 4.10: The output of the edge-based template matcher: the best matching template, location and scale is overlayed over the image as a red outline

5 Real-time Finger-tip Classifier

5.1 Introduction

The pixel-based skin segmenter I developed, as outlined in section 3.3, was demonstrated in section 3.4 for the purposes of skin segmentation, i.e. indicating which pixels in an image were likely to be skin pixels. It was also mentioned that the individual pixel probabilities, rather than being thresholded at 0.5, could be used as the input to a classifier to detect hands or parts of the hands. Here I demonstrate the use of such a classifier for detecting finger-tip.

Figure 5.1 shows the raw output of the skin segmenter, indicating the probabilities of each pixel being skin. Although the probabilities are imperfect (e.g. some parts of the hand marked almost black despite being skin), the shape is still recognisable as a hand, and the finger tips can be made out. This means that despite the removal of information from the input image, the remaining information (i.e. skin probabilities) can still be used to identify where the finger-tips are in the picture. The advantage of using the skin segmenter probabilties over the raw RGB pixels is that the classifier is more robust to changes in lighting conditions and skin colour changes (providing the skin segmenter is sufficiently robust).

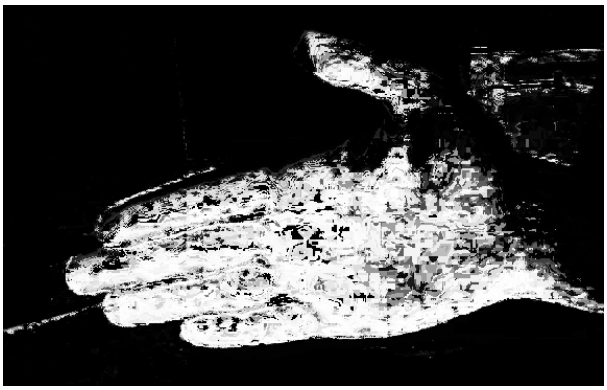


Figure 5.1: The raw output of the skin segmenter, indicidating the probability of each pixel being skin: from zero (totally black) to unity (totally white)

5.2 Methodology

For the purpose for which the finger-tip detector was initially developed (the "paper keyboard" application as outlined in section 6), the rough scale of the finger-tip in the image does not vary. Hence, the classifier is designed to operate on a fixed-size window just large enough to capture the finger tip and enough of the surrounding area for reliable differentiation between finger tips and other hand features (as shown in figure 5.2). For a given window location, all the individual skin probability values for each pixel are combined to perform the feature vector.

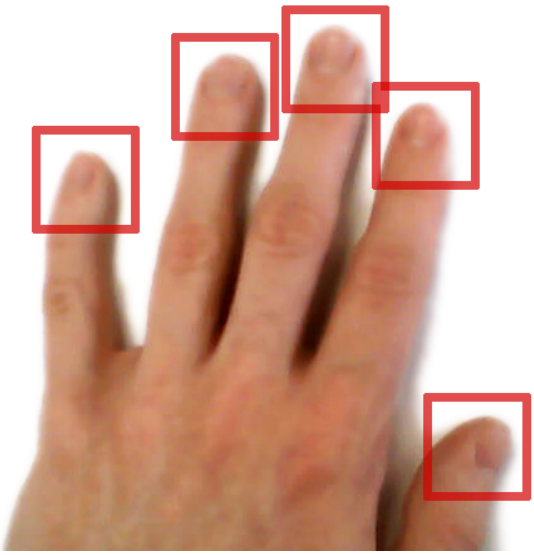


Figure 5.2: Fixed-size window locations enclosing a finger-tip

No pre-processing was used since the skin probabilities were considered not to contain any redundant information for the purposes finger-tip classification, nor was it believed that any noise removal, perhaps using local averaging operations, would benefit any classifier. Nothing was used to generate hypotheses for where the finger tip location — instead the classifier was run with the window at every possible location. However, to speed up the algorithm, an initial stage of classification was carried out at each given window location to quickly knock out false matches (see section 5.5).

5.3 Available Classifiers for Finger Tip Detection

The classifier is only required to determine whether or not a finger-tip was present inside a window (i.e. rather than to provide a probability value, etc). Three different classifiers were explored for this purpose and are outlined below:

Boosting

Boosting is a process in which a set of weak classifiers (classifiers which can label

data with a limited accuracy) are formed into an ensemble that together can much more reliably classify data [3, p. 657]. The most commonly used version is *AdaBoost* as developped by Feund and Schapire (1996) [18]. For every iteration, *AdaBoost* finds a weak classifier which minimises the weighted classification error. Initially all data points are given equal weighting for the classification error, but this is updated for each iteration so as to increase the weight of data points which have been previously misclassified so that each additional weak classifier seeks to improve the performance of the ensemble. When a set number of weak classifiers have been produced, these are combined to form a strong classifier with the weighting of each classifier so as to minimise the overall classificiation error.

Random decision forests

A random decision forest is a classifier formed by a combination of decision trees. Each decision tree within a forest is independantly used to classify the data, and then the modal class label is used as the output of the classifier. Each decision tree is trained on a random subset of the training data, independent of the subsets used to form other trees, but with the same distribution. Some algorithms also use a random selection of features for each decision tree. [6]

Support Vector Machines

A two-class Support Vector Machine (SVM) seeks to divide a feature-space into two partitions using a linear decision boundary that maximises the margins between the decision boundary and the closest training data point of each class (see figure 5.3). Where the data is not linearly separable, the algorithm seeks to minimise the sum total of the distances of all misclassified data points from the decision boundary. To allow a non-linear decision boundary, the input feature vector is often transformed into a feature space with larger dimensionality within which the SVM is trained. The linear decision boundary in the higher dimensional space then maps into a non-linear decision boundary in the original feature space. [3, p. 326-345]

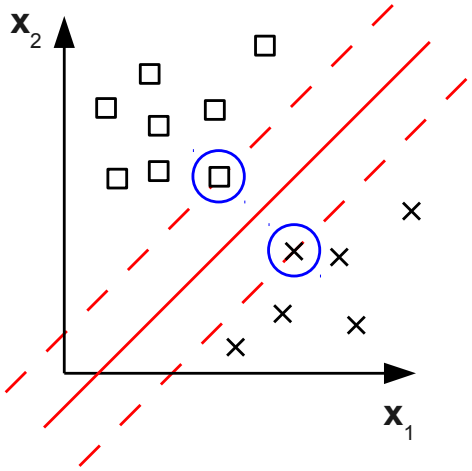


Figure 5.3: The linear decision boundary (continous red line) of a SVM maximises the margin between the boundary and the the support vectors (enclosed in blue circles); the squares and crosses represent the training data of the two respective classes

5.4 Operation of the Classifiers

The classifier was run on the skin probability images input from the "paper keyboard" application (see section 6). The scale of the finger tips is roughly fixed. This is beacuse the "paper keyboard" application transforms the image from the webcam such that the "paper keyboard" is always at the same scale, and the finger tips remain close to the "paper keyboard". An 8 by 8 pixel window size was chosen since this adequately captures the finger tips and part of the surrounding area, while not being too large as to too greatly increase the processing time of the classifier, preventing real-time operation. This results in a $8^2 = 16$ dimension feature vector. Figure 5.4 shows an example the skin probability image and demonstrates the window size.

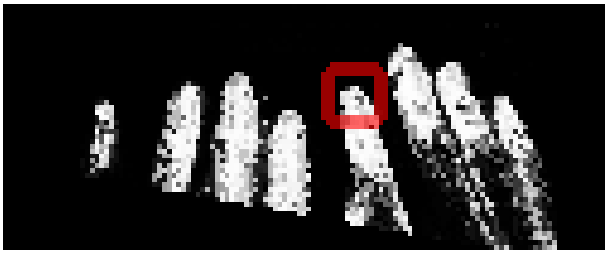


Figure 5.4: The skin probability image input into the classifer; the area within the red square is equal to the window size

The OpenCV machine learning framework was used to provide the three different classifiers[1]. A matrix of feature vectors and class labels (finger tip or non finger tip) extracted from a set a labelled training images was input into the three different classifiers

[1]http://opencv.willowgarage.com/documentation/cpp/ml._machine_learning.html

to train them. The trained classifier was then tested first against the the test images, and then tested for its initially developed purpose: the "paper keyboard" application.

The training images were labelled as follows:

- A 2 by 2 area in the centre of each finger tip was labelled as positive (finger tip).
- The remaining 6 by 6 area surrounding the centre of each finger tip was kept as an exclusion zone and not labelled.
- All other locations were marked as negative (background)

Figure 5.5[2] illustrates this labelling. The purpose of this labelling was to increase the robustness of the classifier — by labelling each finger-tip in four places, it was believed that the trained classiser would have a larger probability of correctly detecting newly seen finger tips and it was believed that the exclusion zone would decrease the sensitivity of the classifier to small changes in the finger-tip pattern or shape. Though it was believed that this would increase the likelihood of the classifier classifying finger tips in four of more places, this was not considered an issue as long as there was no overlap between classified finger tips location, since the situation could be resolved by taking the centroid of any connected locations identified as finger tips.

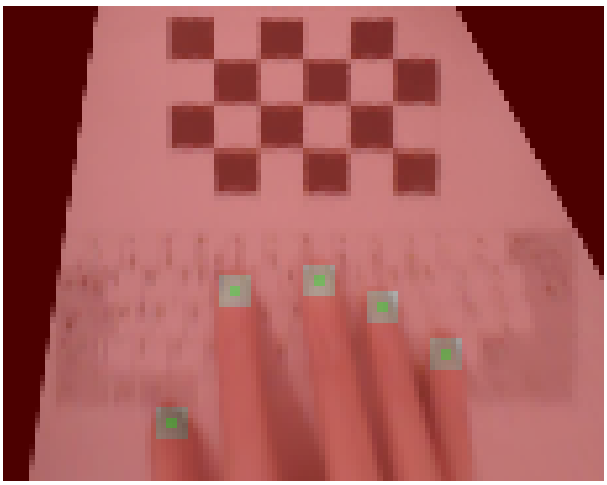


Figure 5.5: The labelled training image: red areas are labelled as background and green areas as finger tips

The following implementations of each classifier are used in OpenCV:

Boosting Real AdaBoost as documented in [19]

Random Decision Forests The original algorithm as documented in [6]

Support Vector Machines The algorithm as documented in [8] using a two degree polynomial kernel

[2]The scale is as used in the training.

5.5 Increasing the Efficiency for Real-Time Classification

The developed classifier relies on the collation of features into a feature vector for each location and a large amount of subsequent processing. A massive speed-up can be achieved using a cascade structure: running a series of tests on one or a few features designed to quickly reject most negative matches without rejecting any positive matches. This is successfully used in the Viola-Jones framework [40] to allow real-time matching using Haar Features (see 7.3.1). Since the majority of locations within the skin probability image contain no finger tips, such a structure would results in a large reduction in the time taken to classify an image.

Regrettably, the OpenCV implementation of Boosting does not include support for cascade structures, and time restrictions have not allowed me to develop such a system. Instead the average skin probability value of the four centre-most pixels (highlighted red in figure 5.6) is calculated on the forth row of each window is calculated. Only if this value meets a given threshold is the classifier run on the given window, effectively forming a two-level cascade. The threshold is set to be large enough as to yield no false negatives when testing against the training images and live (when running the "paper keyboard" application.

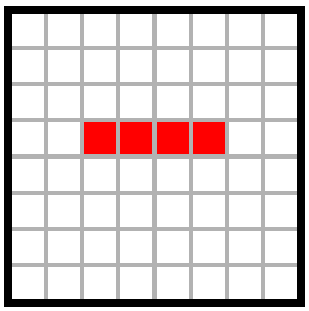


Figure 5.6: The four pixels used as part of the first level of the cascade

This feature results in a reduction of number of window locations on which the classifier is run by two orders of magnitude, and is essential for the real-time operation of the paper keyboard.

5.6 Performance of the Classifer

Since each classifier was trained on images prouduced from the "paper keyboard" application, and tested with the same images and by running the keyboard application, the performance of the classifier is discussed alongside the "paper keyboard" application in section 6.3.

6 “Paper Keyboard” Application

6.1 Introduction

While the eventual aim of the finger-tip detector is to be able to detect finger tips in any image, here I demonstate its use for “paper keyboard” application. The application allows users to type on a piece of paper, observed by a webcam, and have the letters appear on-screen. Figure 6.1 shows its potential use. The expected application would be mobile phones and other small mobile devices, particularly if the built-in webcam could be used, for which a larger size keyboard would improve the ease of typing but carrying a physical keyboard would impose a significant burden on the user.

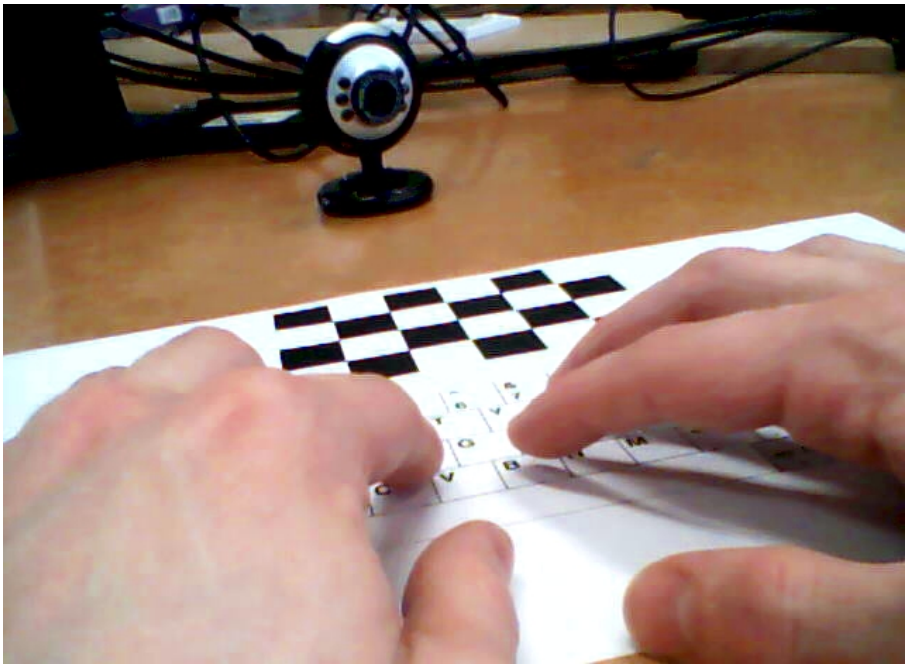


Figure 6.1: A user types on the “paper keyboard” in front of a webcam

6.2 Operation

The “paper keyboard” has three stages between capturing an image:

1. Image Transformation — the image captured from the webcam is transformed based on the location of the calibration pattern, such to generate the image that would be seen if looking down above the keyboard.

2. finger tip detection — the location of any finger tips in the image are detected using the classifier

3. Key "press" detection — the location of finger tips between images are processed to see if a keyboard press has taken place.

The three stages are detailed below:

6.2.1 Image Transformation

The "paper keyboard" is designed to be able to be operated with a webcam pointing at any angle[1], as long as the keyboard (and calibration pattern) is in view. While it would be possible to detect the keyboard location by detecting parts of the keyboards or smaller QR codes, these would require additional processing power, and so a single calibration pattern composed of chessboard squares is used (see figure 6.2). The pattern is relatively large to increase the accuracy of the transformation and make it robust to slight creases in the piece of paper.

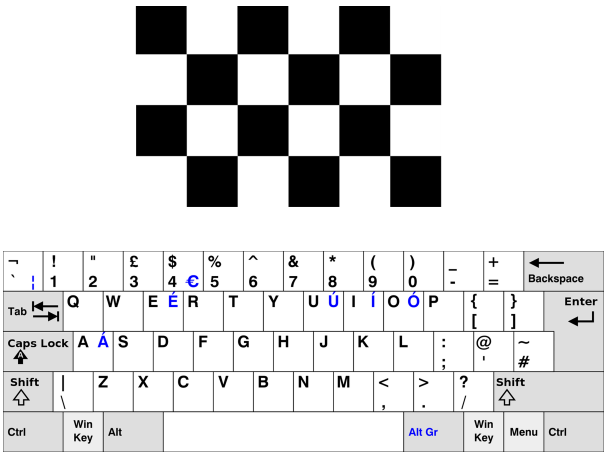


Figure 6.2: The layout of the "paper keyboard", including calibration pattern (above)

The standard OpenCV commands[2] are used for detecting the locations of the chessboard squares in both the image captured from the webcam (as demonstrated in figure 6.3) and the reference keyboard layout image (i.e. figure 6.2). These is then used for calculating the homography (plane-to-plane tranformation) between the calibration patterns in the captured webcam image and the reference image. The procedure to detect chessboard squares takes around a second and so is not run on every webcam image. Instead it runs in parallel with the rest of the processing, and receives a new image each time it has calculated a new homography. This means that the system won't be immune

[1] In actual fact, the webcam must be behind the keyboard due to the rotational symmetry of the calibration pattern.

[2] *findChessboardCorners* and *findHomography* as documented on http://opencv.willowgarage.com/documentation/cpp/calib3d_camera_calibration_and_3d_reconstruction.html

to sudden shock movements of the paper, but will be immune to small movements in the paper over time. The cached homography is used to transform the webcam image such that the transformed image matches the reference keyboard layout[3].

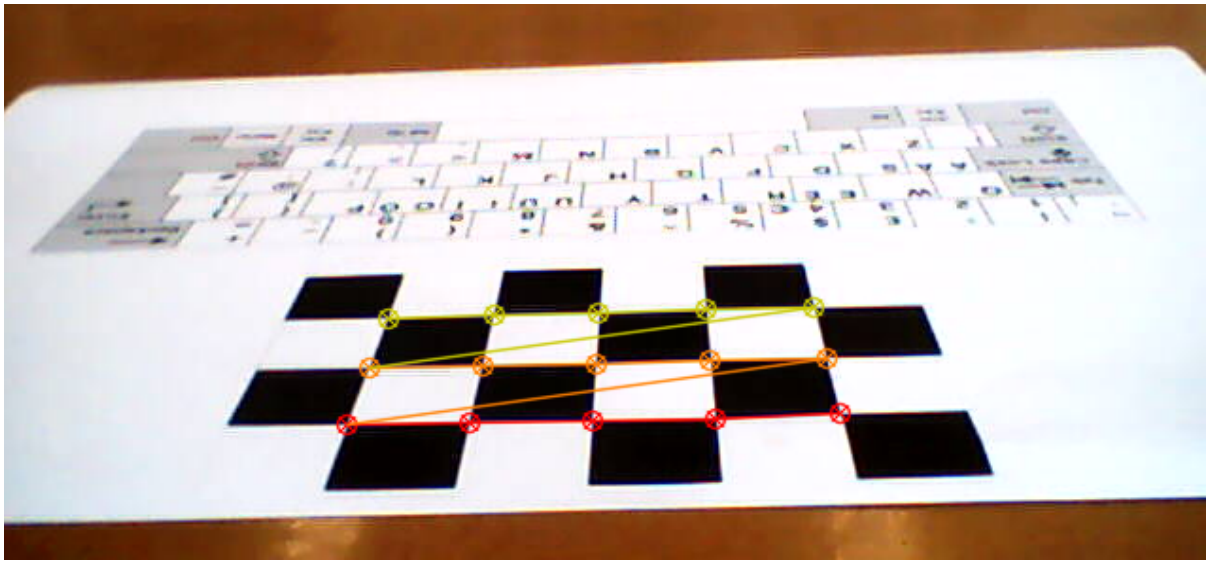


Figure 6.3: The chessboard squares (calibration pattern) detected in the webcam input image

Figure 6.4 shows the effect of the transformation. There is an almost perfect correspondance between the overlayed keyboard layout image and the transformed image captured by the webcam. (The fingers appear very much distorted since they are far from parallel with the plane formed by the piece of paper.)

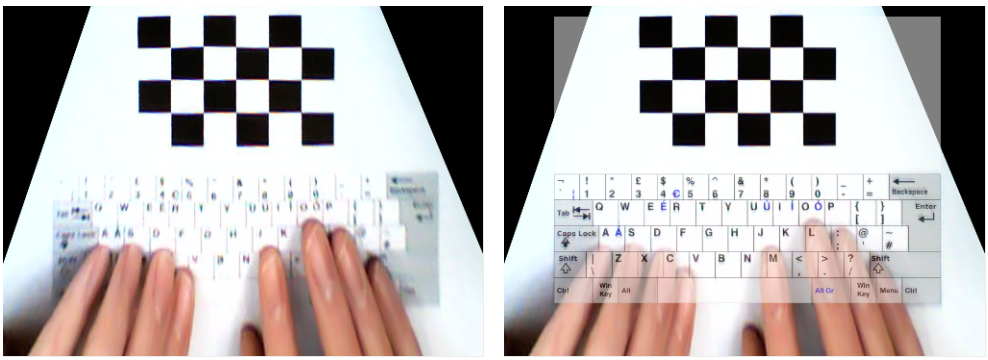


Figure 6.4: The webcam image after transformation: left, without and right, with, the original keyboard layout image overlayed

6.2.2 Finger Tip Detection

The figures shown in the previous section show all the images in high resolution for demonstration purposes. In actual operation, the webcam input images are transformed

[3]using the OpenCV *warpPerspective* command http://opencv.willowgarage.com/documentation/cpp/imgproc_geometric_image_transformations.html

such that the resulting image is 125 by 100 pixels in size. This enables the transform operation, and subsequent operations to run much quicker (approximately four times faster each time the dimensions are halved). The skin segmenter (as outlined in section 3.3) is then used to determine the probability that each pixel contains skin. The skin segmenter is only run on the section of the image containing the keyboard. Figure 6.5 demonstrates the scale of the the transformed image, and the resulting skin probabilities.

Figure 6.5: The scale at which the finger tip detector operates: left, the transformed imaged; right: the skin probabilities of each pixel

The resulting skin probability image is input to the finger tip classifier (see section 5.5 for details on the operation). The output of the classifier is an image with all located believed to contain a finger tip indicated, as shown in the middle of figure 6.6. To ensure that only one position is output per finger tip the following process takes place:

1. The prediction image is blurred slightly. Initially all locations identified finger tip have the value 255 while all other loctions have the value 0. The blurring operation smoothes the values out.

2. All non-zero value locations in the image are collated and sorted.

3. From the highest to lowest values, the locations are extracted one by one from the sorted list; however all pixel locations within a five pixel Manhattan distance of an extracted locations are removed from the list.

This results in a number of finger tips locations at least 5 pixel lengths apart and hopefully corresponding to each finger tip. The smoothing operation means that the extracted locations are centred on each "clump" of pixels.

Figure 6.6 shows the output of the skin segmenter, the output of the classifier and the extracted finger tip locations overlayed on the transformed image. All images are displayed at the scale at which the system operates.

In the final step, the extracted finger tip locations are mapped to their nearest keyboard keys, as shown in figure 6.7. The application is unfortunately unable to run fast enoguh to reliably detect keyboard presses based on the movement of the finger. Therefore, the application infers a finger remaining over the same key in two successive frames as a

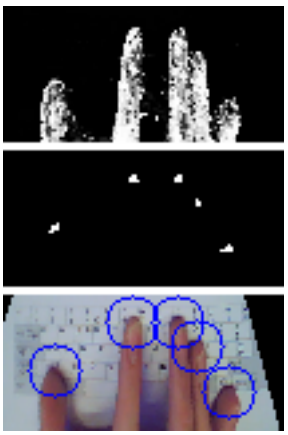


Figure 6.6: Operation of the finger tip classifier and extracted finger tip locations: top, image showing skin probabilities; middle, the output of the finger tip classifier; bottom, the finger tip locations extracted and overlayed on the transformed image

key press. The finger must move to another key and back before the key is re-detected. Unfortunately, the result is that the keyboard must be operated with one finger, or with one finger of each hand with only one finger moving at a time.

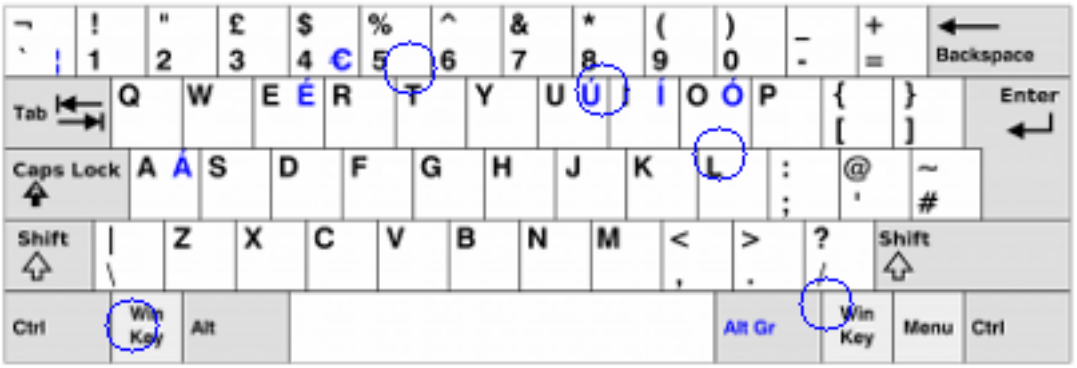


Figure 6.7: The finger tip locations overlayed over the keyboard image

6.3 Performance of the "Paper Keyboard" Application

The paper keyboard application performed differently with the three different classifiers. The different performanced is outlined below:

Boosting

The "paper keyboard" application performed very well with the boosting classifier implementation, correctly classifying all the training images and perforing well under live testing.

Random Decision Forest

The application did not perform at all well using the Random Decision Forest implementation, as all locations in the image were classified as not containing finger

tip, even in the training images. I believe that this is because the individual classifiers forming hte forest trained on different finger tip samples and so, though few of the classiers would likely have correctly identified the finger tip, because the majority of the classifiers would not have identified it, the modal label would always have been negative (non finger tip).

Supper Vector Machine

The application performaned fairly well with the Support Vector Machine, correcly classiftying all the training images, however many false positives were obtained under the live testing (as shown in figure 6.8). It is possible that this was because the large margin between the support vectors meant that sides of the fingers at angles not featured in the training images and shadow forms not seen in the training images were incorrectly classified.

Figure 6.8: The finger tip locations according to the SVM classifier, which correctly detecs all the finger tip locations but yields many false positives

As a result, boosting was chosen for use as the finger tip classifier for the final system (and was used to produce figures 6.6 and 6.7).

7 Edge Detection System

7.1 Introduction

The edge-based template matching system demonstated in section 4 struggles because templates match better against lines found in the background clutter or gestures other than the correct template matches against the gesture. One way to alleviate this is to identify which edges in an image form the outline of the hand and fingers and which form background clutter or other lines within the hand (such as shadows and veins). Here I demonstrate using machine learning for the purpose.

7.2 Operation of the Edge Detection System

Multiple approaches are available for the purpose, however I chose to implement a system based on Dollar (2006) [13]. The approach uses a AdaBoost classfier (as described in 5.3) and operates on a 50 by 50 size window and uses approximately 50,000 features for the feature vector obtained trough multiple methods.

In order to develop system which can train and run and on an ordinary desktop computer in reasonable time, the system I have chosen to implement uses an 8 by 8 window and two of the feature types used in in the approach by Dollar (2006) [13], Haar Features [30] and 2D gradients, using 512 featues between the two. The same framework used for the finger tip classifier outlined in section 5 is used, including the OpenCV boosting classifier (as described in section 5.4), meaning that skin probabilities could also be used (and would almost certainly be used if the system was developed for use with labelled colour gesture training images).

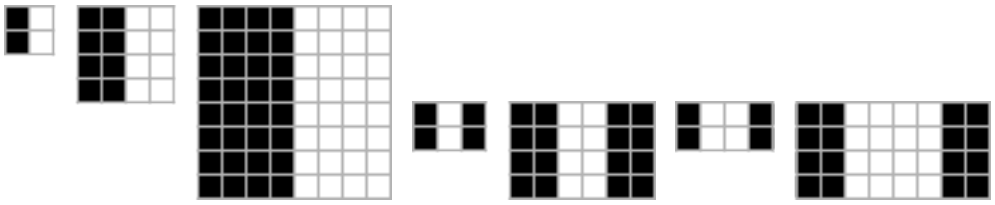
7.3 Operation of Feature Extractors

7.3.1 Haar Features

Haar (or Haar-like) features, as first presnted by Papageorgiou (1998) [30], are features so named because of their similarity to Haar wavelets [10, p. 5]. A Haar feature at a given location is calculated as the difference between the sum of pixel intensity values within one or more rectangles and the sum of pixels intensity values within one or more

different rectangles.

The Haar features I have implemented are edge features, single-line features, and double-line features. The basic templates of the features can be found in figure 7.1: the difference between the sum of the black and the sum of the white pixels in each case is calculated. The features templates are evaulated at all possible locations within the 8 by 8 window, and at both the orientation in the figure and rotated 90°, giving a total of 344 features.



(a) Edge feature (b) Single line feature (c) Double line feature

Figure 7.1: The three basic template of the different haar features implemented

7.3.2 Gradient features

The gradient features are the first derivative in the x and y directions respectively at a given point. To calculate the gradient features, the Scharr operators [34],

$$S_x = \frac{1}{32}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \text{ and } S_y = \frac{1}{32}\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

, are colvolved with the input image to calculate first derivatives in the respective directions.

The input image is then smoothed with a Gaussian kernel and sub-sampled by rejecting every other row and column and then convoluted again with the Scharr operators to yield the gradient features at half-scale. Finally this is repeated to achieve gradient features at quarter-scale. This provides a total of 168 features.

7.4 Performance of the Edge Detection System

The edge detection system was trained on a dataset produced by the Carnagie Mellon Univeristy[1] (as used in [37]). The dataset consists of 30 images with the outlines of objects hand labelled by a human (see figure 7.2 for an example). All the images are colour, but were converted to greyscale in order to test the system. The system was trained on 27

[1] http://www.cs.cmu.edu/ stein/occlusion_data/

of the images[2] and the remaining three images were used to test the system.

Figure 7.2: One of the training images used for training the edge detection system (left) and the associated labelled edge mask (right)

Table 7.4 shows the results of the edge detection system and the Canny edge detector [9] (for comparison) when run the test images. Figure 7.3 shows the output of the edge detection system for each of the test images, alongside the test image, labelled edge mask (ground truth) and the output of a Canny edge detector.

The system outperforms the Canny edge detector, achieving a higher rate of correctly identified pixels and a lower rate of false positives, though with a higher rate of false negatives. The system is not reliable enough, however, for the edge-based template matcher (developed in section 4) as there are gaps in many of the edges, which would prevent the line segmenter (see section 4.3.1) working effectively. The system also performs badly with textures such as grass and leaves, yielding a result that is barely better than the Canny edge detector.

	Pixel count / % by Edge Detection Method	
Classification	**Implemented system**	**Canny**
Correctly identified as edge	1,593	2,135
Incorrectly identified as edge	5,673	14,075
Correctly identified as non-edge	218,875	210,472
Incorrectly identified as non-edge	4,259	3,717
Correctly identified	95.7 %	92.3 %
False positive	2.46 %	6.10 %
False negative	1.84 %	1.61 %

Table 7.1: The performance of the edge detection system and the Canny Edge detector (for comparison) on the test images

To work effectively, the performance of the system needs to be improved. Suggestions for how to achieve this are outlined in section 8.2.1.

[2]Only one in ten of the locations not marked as an edge was used to train the system, in order to allow all the training data to fit in the 2Gb of RAM of the computer the system ran on.

Figure 7.3: The output of the edge detection system for the test images: from left to right, the input image; the labelled edge mask (ground truth); the output of the edge detection system; the output of a Canny Edge Detector (for comparison)

8 Conclusions

8.1 Outline of Achievement

The system as it currently stands has not met the goals as outlined in section 1.2, but substantial progress has been made towards meetings the goals of the system. In particular, I have achieved the following developments:

1. A pixel based skin segmentation sytem which is capable of identifying skin pixels in an image to a high level of accuracy (see section 3.4) and being used as the input for a finger tip classifier (see section 5) — this would be valuable for identifying the hands within an image, and improving the performance of the edge-detector, as required for the operation of the system.

2. An edged-based template matching system which is partially successfully at identifying hand gestures (see section 4.4) — with improvements to the system to improve its performance (as proposed in section 8.2.2) the system would be valuable for identifying hand signs in video frames: an essential step for sign language recognition.

3. A finger tip classifier, which has performed very well in the "paper keyboard" application (see section 6.3) — this could be used as part of a proposed method of dealing with slight changes in pose and orientation of the hand between images of the same sign as described in section 8.2.2

4. An edge-learning system, which slightly outperforms the Canny Edge detector [9] — if the system is improved, it would be valuable for improving the performance of the edge detection system (see section 8.2.2)

8.2 Proposed Further Work

8.2.1 Improving the Edge Detection system

As mentioned in section 7.2, the edge detection system could be improved by using the skin probabilities from the skin segmenter as features, as used for the real-time finger tip classifier. The system could also be improved by using a larger window and more feature extractors, as in [13].

The increase in processing time due to the increase in number of features could be mitigated by using a cascade architecture (as discussed in section 5.5).

8.2.2 Improving the Edge-based Template Matcher

The edge-based template matcher performed poorly because of slight differences in orientation and pose between images of the same sign (see section 4.4). Four improvements to the system to address this are outlined below:

1. By using an effective edge detection system, particularly when using skin probabilities as features, much of the background clutter will be removed from the input images. In addition, lines within the hand could either be removed or classified separately (i.e. with two classes of line detected in the system) using the system and separate templates for the outline of the hand and lines within the hand. While this would not directly deal with the problem of changes in pose and orientation of the hand between images of the same gesture, it would increase the system's robustness to such changes.

2. Each gesture could initially be matched using a skin pixel probability based classifier (similar to that developed for finger tip classification in section 5), before using the edge-based template matcher to confirm the gesture. A similar approach is used in Stenger (2004) [38].

3. The finger tip and palm locations in the image could be identified using a classifier similar to that developed for the "paper keyboard" application (see section 5) or using the edge-based template matching system and a set of templates with only few line segments. The finger tip and palm locations could then be used to transform the input images so as to line up with the finger tip and palm locations in the templates of signs with finger tips and palms in similar places. The template matcher would then be run on the resulting image. This would deal with the problem of changes in orientation and, to an extent, pose between images of the of the same sign.

4. Another method to improve the performance is the approach used by Ma (2010) [29]: for every template another template (the normalisation template) is produced which matches poorly aginst the (matching) template but matches well against background clutter and other signs used in the system. The cost of a template match at a given location and scale is then calculated as $C = \frac{C_m}{C_n}$ where C_m and C_n are the costs of the matching template and normalisation template respectively at the given location and scale.

Alternatively, a parametric 3D model of the hand could be used to generate 2D templates for edge-matching. The system would iteratively produce templates by projecting

the 3D model until the correct parameters of the hand have been determined. The parameters of the hand would then be used for classifying the signs. This approach is used in Stenger (2004) [38].

8.2.3 Tying the Whole System Together

In order to use an improved system for the original aim: recognising a single sign in isolation, the system will have to work on video streams rather than individual images. One or more of the frames of each training video of a sign will have to be identified as key frames, either by human labelling, motion cues or by detecting it based on the difference in hand shape compared to pre-stored shapes of hands moving between signs. The system would then match these against all or a sample of the frames in an input (test) video to find matches..

One issue for which I have not developed any systems is motion classification. One approach to achieve this would be to identify the motion of the hand by composing a feature vector composed of the relative movement of the hand position between frames and the absolute hand position (perhaps relative to the face, which can be easily detected using a range of techniques such as the Viola-Jones framework [40]), similar to the approach in [36]). The system would then form hidden markov models based on the input videos. Two key issues which would need to be overcome are being able to train the system with a single training video for each sign, and finding the start and of the movements associated with a given sign (if any) rather than the transitionary movement of the hands between signs. The latter could be achieved by a human, however.

Bibliography

[1] H.G. Barrow. Parametric correspondence and chamfer matching: Two new techniques for image matching. Technical report, DTIC Document, 1977.

[2] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[3] C.M. Bishop. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[4] Y. Boykov and G. Funka-Lea. Graph cuts and efficient ND image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006.

[5] Y.Y. Boykov and M.P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[8] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[9] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, 1986.

[10] C.K. Chui. *An introduction to wavelets*, volume 1. Academic Pr, 1992.

[11] R. Conrad. The reading ability of deaf school-leavers. *British Journal of Educational Psychology*, 47(2):138–148, 1977.

[12] A.P. Dhawan and A. Sim. Segmentation of images of skin lesions using color and texture information of surface pigmentation. *Computerized Medical Imaging and Graphics*, 16(3):163–177, 1992.

[13] P. Dollar, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1964–1971. IEEE, 2006.

[14] R. Duda. Pattern Classification and Scene Analysis, 1985.

[15] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. 2004.

[16] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[17] D.A. Forsyth and J. Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[18] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning Interntional Workshop*, pages 148–156. Morgan Kaufmann Publishes, Inc., 1996.

[19] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[20] DM Greig, BT Porteous, and A.H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 271–279, 1989.

[21] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[22] P.V.C. Hough and B.W. Powell. Machine analysis of bubble chamber pictures. *Il nuovo cimento*, 18(6):1184–1191, 1960.

[23] M.J. Jones and J.M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

[24] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern recognition*, 40(3):1106–1122, 2007.

[25] N. Komodakis. Image completion using global optimization. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 442–452. IEEE, 2006.

[26] J.G. Kyle, B. Woll, and G. Pullen. *Sign Language: the study of deaf people and their language*. Cambridge Univ Pr, 1988.

[27] M.Y. Liu, O. Tuzel, A. Veeraraghavan, and R. Chellappa. Fast directional chamfer matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1696–1703. IEEE, 2010.

[28] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[29] T. Ma, X. Yang, and L. Latecki. Boosting chamfer matching by learning chamfer distance normalization. *Computer Vision–ECCV 2010*, pages 450–463, 2010.

[30] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on*, pages 555–562. IEEE, 1998.

[31] S. Ray and R.H. Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, pages 137–143, 1999.

[32] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, pages 309–314. ACM, 2004.

[33] W. Sandler and D.C. Lillo-Martin. *Sign language and linguistic universals*. Cambridge Univ Pr, 2006.

[34] H. Scharr. *Optimal operators in digital image processing*. PhD thesis, Interdisciplinary Center for Scientific Computing, 2000.

[35] Robert Sedgewick. *Algorithms*. Addison-Wesley, 2nd edition, April 1988.

[36] T. Starner, J. Weaver, and A. Pentland. Real-time american sign language recognition using desk and wearable computer based video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(12):1371–1375, 1998.

[37] A.N. Stein and M. Hebert. Occlusion boundaries from motion: low-level detection and mid-level reasoning. *International journal of computer vision*, 82(3):325–357, 2009.

[38] B. Stenger, A. Thayananthan, P. Torr, and R. Cipolla. Hand pose estimation using hierarchical detection. *Computer Vision in Human-Computer Interaction*, pages 105–116, 2004.

[39] A. Thayananthan, B. Stenger, P.H.S. Torr, and R. Cipolla. Shape context and chamfer matching in cluttered scenes. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–127. IEEE, 2003.

[40] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2001.

[41] F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II. Ieee, 2006.

A Source Code

The systems are entirely developed in C++. The source code is available on request.

Statistics

(Generated using David A. Wheeler's 'SLOCCount'.)

Total Source Lines of Code 6,661

Development Effort Estimate (Person-Years) 1.47 (Basic COCOMO model)

B Risk Assessment Retrospective

The submitted risk assessment proved to be accurate, covering all the hazards actually encountered. In particular, back ache and leg ache frequnetly resulted from poor chair design due to the poor design of the many of the chairs used during the undertaking of the project. Eye strain and arm or hand ache was encountered occasionally but was mostly avoided by taking frequent breaks and by focussing frequently on distant objects such as those outside the building. To improve the risk assessment for subsequent projects, I would investigate the venue used for carrying out the work and assess the availability of well-designed chairs. I would also investigate how easy it is to focus on distant objects from a within deskjs in the room (i.e. whether a view of a far wall or outdoors view can be achieved without too much movement of the head). These investigations would allow met to produced a better informed assessment of the level risk from back ache, leg ache and eye strain.