Signature Recognition using Image Processing and Artificial Intilligence

PROJECT REPORT

By

Majdoleen Sameer Abu-Taqa

Tawaddod Nabeel Sous

2009-2010

ABSTRACT

-------------

**In the 21st century, it has become a trend that machines are replacing the man in many fields.**

**But, there are a lot many fields still untouched to machines. But the evolution of modern computers and the development of the branch of Artificial Intelligence, gives man a chance to make a machine that can really replace man from the field which, up till are considered to be the fields reliable on human intellectual power.**

**Signature is the characteristic of the particular person & hence used globally for identifying a person, validity of the documents signed, banking etc. Up till now, in banks where signature of a person is the basic code for transaction, the validity of the signature is generally checked by a man.**

**This is a project, which simulates the ability of a man to recognize a signature from the standard signature he has. We have tried to implement a system which recognizes the signature. We deal with the signature as an image which is scanned through scanner.**

**The image undergoes different normalization techniques and then we extract some features of it to be used as inputs to the fuzzy.**

Introduction

-------------

**Signature of a person is an Important Biometric Attribute of a human being and is used for authorization purpose for decades. With a lot of computing power available with modern computers there is a vast scope to develop fast algorithms for signature recognition. There is a lot of research work is being conducted in this field.**

**Various approaches are possible for signature recognition with a lot of scope of research. In this project we deal with an Off-line signature recognition technique, where the signature is capture and presented to the user in the format of image only. We use various image processing techniques to extract the parameters of signatures and verify the signature based on these parameters.**

**-The Signature Recognition System had passed through a series of steps as the following:**

**\*Preprocessing:**

**1-Select Signature image:**

**As the person signs his signature is scanned using a scanner and inserted into the system as an RGB image regardless to the pen color which is used in signing process.**

**2-Invert Image:**

**The NOT of two images is carried out by performing the inversion operation on the corresponding pixels of the image to produce the output pixel value. The inversion technique can be used to get the negative of the image.**

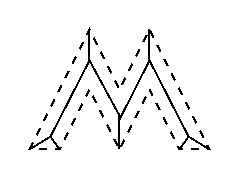
**3-Gray Scale Image**

**Grayscale images are images without color, or achromatic images.**

**The levels of a grayscale range from 0 (black) to 1 (white).**

**4-Binary Image:**

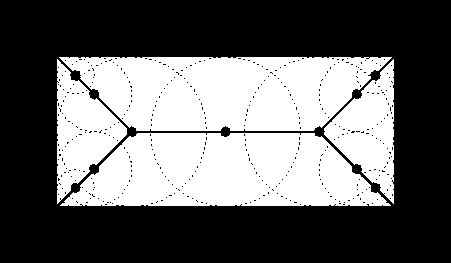
**A binary image is a** [**digital image**](http://en.wikipedia.org/wiki/Digital_image) **that has only two possible values for each** [**pixel**](http://en.wikipedia.org/wiki/Pixel)**.**

**\*Apply Skeletonization Filter**

**Brief Description:**

**Skeletonization is a process for reducing foreground regions in a** [**binary image**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/binimage.htm) **to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. To see how this works, imagine that the foreground regions in the input binary image are made of some uniform slow-burning material. Light fires simultaneously at all points along the boundary of this region and watch the fire move into the interior. At points where the fire traveling from two different boundaries meets itself, the fire will extinguish itself and the points at which this happens form the so called `quench line'. This line is the skeleton. Under this definition it is clear that** [**thinning**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm) **produces a sort of skeleton.**

**Another way to think about the skeleton is as the loci of centers of bi-tangent circles that fit entirely within the foreground region being considered. Figure 1 illustrates this for a rectangular shape.**

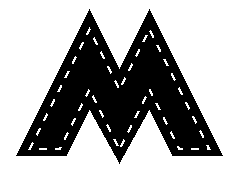


**Figure 1** Skeleton of a rectangle defined in terms of bi-tangent circles.

**The terms medial axis transform (MAT) and skeletonization are often used interchangeably but we will distinguish between them slightly. The skeleton is simply a binary image showing the simple skeleton. The MAT on the other hand is a** [**graylevel image**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm) **where each point on the skeleton has an intensity which represents its distance to a boundary in the original object.**

**How It Works:**

**The skeleton/MAT can be produced in two main ways. The first is to use some kind of morphological** [**thinning**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm) **that successively erodes away pixels from the boundary (while preserving the end points of line segments) until no more thinning is possible, at which point what is left approximates the skeleton. The alternative method is to first calculate the** [**distance transform**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm) **of the image. The skeleton then lies along the singularities (i.e. creases or curvature discontinuities) in the distance transform. This latter approach is more suited to calculating the MAT since the MAT is the same as the distance transform but with all points off the skeleton suppressed to zero.**

****

**\*Dilation**

**Brief Description:**

**Dilation is one of the two basic operators in the area of** [**mathematical morphology**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/matmorph.htm)**, the other being** [**erosion**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm)**. It is typically applied to** [**binary images**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/binimage.htm)**, but there are versions that work on** [**grayscale images**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm)**. The basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground** [**pixels**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/pixel.htm) **(i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller.**

**How It Works**

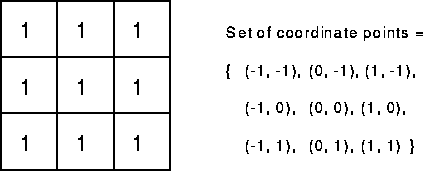
**Useful background to this description is given in the** [**mathematical morphology**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/matmorph.htm) **section of the Glossary.The dilation operator takes two pieces of data as inputs. The first is the image which is to be dilated. The second is a (usually small) set of coordinate points known as a** [**structuring element**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm) **(also known as a** [**kernel**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/kernel.htm)**). It is this structuring element that determines the precise effect of the dilation on the input image.**

**The mathematical definition of dilation for binary images is as follows:**

**Suppose that X is the set of Euclidean coordinates corresponding to the input binary image, and that K is the set of coordinates for the structuring element, Let Kx denote the translation of K so that its origin is at x. Then the dilation of X by K is simply the set of all points x such that the intersection of Kx with X is non-empty.**

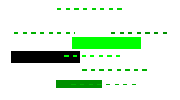
**The mathematical definition of grayscale dilation is identical except for the way in which the set of coordinates associated with the input image is derived. In addition, these coordinates are 3-D rather than 2-D.**

**As an example of binary dilation, suppose that the structuring element is a 3×3 square, with the origin at its center, as shown in Figure 1. Note that in this and subsequent diagrams, foreground pixels are represented by 1's and background pixels by 0's.**

****   
Figure 1 A 3×3 square structuring element

**To compute the dilation of a binary input image by this structuring element, we consider each of the background pixels in the input image in turn. For each background pixel (which we will call the input pixel) we superimpose the structuring element on top of the input image so that the origin of the structuring element coincides with the input pixel position. If at least one pixel in the structuring element coincides with a foreground pixel in the image underneath, then the input pixel is set to the foreground value. If all the corresponding pixels in the image are background, however, the input pixel is left at the background value.**

**For our example 3×3 structuring element, the effect of this operation is to set to the foreground color any background pixels that have a neighboring foreground pixel (assuming** [**8-connectedness**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/connect.htm)**). Such pixels must lie at the edges of white regions, and so the practical upshot is that foreground regions grow (and holes inside a region shrink).**

**\*Sub Images**

**The pixel subtraction operator takes two images as input and produces as output a third image whose** [**pixel values**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/value.htm) **are simply those of the first image minus the corresponding pixel values from the second image. It is also often possible to just use a single image as input and subtract a constant value from all the pixels. Some versions of the operator will just output the absolute difference between pixel values, rather than the straightforward signed output.**

**How It Works:**

**The subtraction of two images is performed straightforwardly in a single pass. The output pixel values are given by:**

**Eqn:eqnsub1**

**Or if the operator computes absolute differences between the two input images then:**

**Eqn:eqnsub2**

**Or if it is simply desired to subtract a constant value C from a single image then:**

**Eqn:eqnsub3**

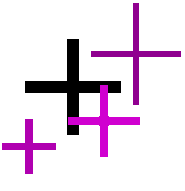
**If the pixel values in the input images are actually vectors rather than scalar values (e.g. for** [**color images**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/colimage.htm)**) then the individual components (e.g.** [**red, blue and green components**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/rgb.htm)**) are simply subtracted separately to produce the output value.**

**Implementations of the operator vary as to what they do if the output pixel values are negative. Some work with image formats that support negatively-valued pixels, in which case the negative values are fine (and the way in which they are displayed will be determined by the display** [**colormap**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/colmap.htm)**). If the image format does not support negative numbers then often such pixels are just set to zero (i.e. black typically). Alternatively, the operator may** [**`wrap'**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/wrap.htm) **negative values, so that for instance -30 appears in the output as 226 (assuming** [**8-bit pixel values**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm)**).**

**If the operator calculates absolute differences and the two input images use the same pixel value type, then it is impossible for the output pixel values to be outside the range that may be represented by the input pixel type and so this problem does not arise. This is one good reason for using absolute differences.**

**\*Color Filling**

**Give each image resulted from the previous step a certain color.**

**\*Add images**

**Brief Description**

**In its most straightforward implementation, this operator takes as input two identically sized images and produces as output a third image of the same size as the first two, in which each** [**pixel value**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/value.htm) **is the sum of the values of the corresponding pixel from each of the two input images. More sophisticated versions allow more than two images to be combined with a single operation. A common variant of the operator simply allows a specified constant to be added to every pixel.**

**How It Works**

**The addition of two images is performed straightforwardly in a single pass. The output pixel values are given by:**

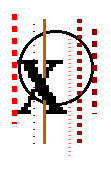
**Eqn:eqnadd1**

**Or if it is simply desired to add a constant value C to a single image then:**

**Eqn:eqnadd2**

**If the pixel values in the input images are actually vectors rather than scalar values (e.g. for** [**color images**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/colimage.htm)**) then the individual components (e.g.** [**red, blue and green components**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/rgb.htm)**) are simply added separately to produce the output value.**

**If the image format being used only supports, say** [**8-bit integer pixel values**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm)**, then it is very easy for the result of the addition to be greater than the maximum allowed pixel value. The effect of this depends upon the particular implementation. The overflowing pixel values might just be set to the maximum allowed value, an effect known as** [**saturation**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/wrap.htm)**. Alternatively the pixel values might wrap around from zero again. If the image format supports pixel values with a much larger range, e.g. 32-bit integers or floating point numbers, then this problem does not occur so much.**

****

**\*XOR**

**Brief Description**

**The XOR function is only true if just one (and only one) of the input values is true, and false otherwise. XOR stands for eXclusive OR. As can be seen, the output values of XNOR are simply the inverse of the corresponding output values of XOR.**

**The XOR (and similarly the XNOR) operator typically takes two** [**binary**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/binimage.htm) **or** [**graylevel images**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm) **as input, and outputs a third image whose** [**pixel values**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/value.htm) **are just those of the first image, XORed with the corresponding pixels from the second. A variation of this operator takes a single input image and XORs each pixel with a specified constant value in order to produce the output.**

**How It Works**

**The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them, or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the XOR operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in** [**bitwise fashion**](http://homepages.inf.ed.ac.uk/rbf/HIPR2/logic.htm#bitwise)**.**

**\*Fuzzy**

**Count Red, Blue, Black and Green pixels in the image resulted from the last step after making a boundary box around the signature area only, and use them as inputs to the Fuzzy system that we had built, and according to the rules we wrote the result will be obtained.**