Master's thesis

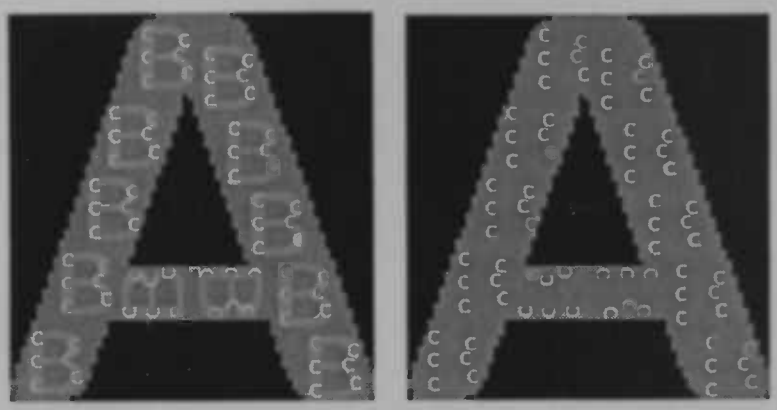
Learning scale invariant filters

N. J. Boersma

Institute for Mathematics and Computing Science
University of Groningen
The Netherlands

Supervisor: M. H. F. Wilkinson

31st August 2004



Contents

Abstract

Connected filters are image operators that work on the connected components (in the binary case), or flat zones (in the grey-scale or color case) of an image. We have developed a connected filter which can be trained to respond to certain types of shapes. Such a filter can for example be used for image recognition purposes.

A mathematical representation of the shape of an object is used in the form of moments. Since the shape of an object does not change under translation, rotation and scaling, the moments have to be invariant against these transformations.

A *Max-Tree* algorithm is used to extract the connected components from an input image such that the shape filter can be applied to them individually. A standard *Max-Tree* algorithm is adapted, such that it implements a nearest neighbor classifier which can handle vector attributes like moments.

In this report, we will present implementation details and a comparison between two different shape descriptors: Hu moment invariants (1962) and Krawtchouk moment invariants (2003). All insights that were gathered during the project are presented as well.

Chapter 1

Introduction

Connected filters are image operators that work on the connected components (in the binary case) or flat zones (in the grey-scale, or color case) of an image. A component is removed from the image when this component meets a certain criterion. For example, when the area of a component is used as a criterion, the connected filter can be set to delete all components that are larger than this given area. Such an area filter is relatively easy to implement, because the area of each component is fixed and can be calculated very easily.

However, the criterion (or attribute) of a connected filter is not always easy to calculate. A shape criterion, for example, is very difficult to calculate, because the shape of an object is not always fixed. People can easily distinguish between shapes and can sort objects by their shape properties, for example, by removing the round shapes from the rectangular shapes. Decisions can be made whether certain odd-shaped components are more rectangular than round or vice versa. A computer can only do computations on a component, so it is essential to find a mathematical representation of the shape of an object.

When a shape filter is used for the purpose of object recognition in an image, the filter should be provided with an example of an object to which the filter should respond. This *reference image* should represent the shape to which the filter should respond very well. Since small fluctuations in the shape of an image are allowed (due to noise or, when using the filter for character recognition, the use of different fonts), it is better to use multiple reference images of similar shaped objects and train the filter to respond to all objects which are similar shaped. The next step would be to use reference images of different shapes (*classes* of shapes) and train the filter in a way that it can be used as a classifier that can determine to which class each component belongs.

Goal of my research project is to develop such a connected shape filter, which can be trained to respond to certain types of shapes. As a representation of the shape of a component, different variations of *moments* [4] are used. The filter should also be invariant to translation, scaling and rotation. In this way the filter will require a smaller number of reference images. Making the filter invariant to these transformations is not trivial. Connected area filters, for example cannot be made scale invariant, because the area of a component changes when scaling the component. The shape of an object does not change when scaling the object, however; therefore shape filters can be made scale invariant.

In this report, we will first give some background information on morphological image operators, which can be used or are being used in my implementation of the shape filter. After that, we will discuss the method that is used to extract the connected components from an image, and how this method can calculate the desired properties of these components in an efficient way. This method will also perform the filtering step. The third part of this report consists of methods to represent the shape of a component. All these methods are based on the concepts of moments. The last part consists of a description of which classifier is used to create a trainable connected

filter. Results of all implemented methods, as well as a comparison between these methods will be given at the end of each section.

Chapter 2

Morphological image operators

2.1 Introduction

In this section we will explain some of the basic morphological image operators. Since many of these operators are best understood when applied to binary images (they were even originally designed for binary images), the images to be processed have to be *thresholded* first. Thresholding converts a grey scale image to a binary image. Binary images are images that contain only foreground pixels (e.g. white pixels) and background pixels (e.g. black pixels). Binary images can be represented by the point set of foreground pixels. Since morphological operators are set operators, we will represent binary images by their set of foreground pixels.

Thresholding [2] is not a morphological image operator because it is not a set operator. A 2-dimensional grey-scale image $f(x, y)$ can be binarized by separating the foreground pixels (for example from our object of interest) from the background pixels (other non-interesting parts of the image).

A threshold T is chosen to binarize the image; pixels with a grey-level value larger than T become foreground pixels and the other pixels become background pixels:

$$\begin{aligned} f_T(x,y) &= 1, \quad when f(x,y) \geq T \\ f_T(x,y) &= 0, \quad when f(x,y) < T \end{aligned} \tag{2.1}$$

An example of thresholding can be seen in figure 2.1 in which a grey-scale image with grey levels $0 - 255$ is thresholded with threshold $T = 80$.

Figure 2.1: An example of thresholding an image. The left image is the original image and the right image is the original image after thresholding with threshold $T = 80$

2.2 Binary Dilation

The dilation $X \oplus B$ [2] is the point set of all possible vector additions of pairs of elements, one from each of the set X and B:

$$X \oplus B = \{p \in M : p = x + b, x \in X \text{ and } b \in B\} \tag{2.2}$$

where M is the Euclidean 2D space.

The dilation operator is commutative, associative and invariant to translation over a vector h. In the following three equations commutativity, associativity and translation invariance are defined respectively:

$$\text{Commutativity: } X \oplus B = B \oplus X \tag{2.3}$$

$$\text{Associativity: } X \oplus (B \oplus D) = (X \oplus B) \oplus D \tag{2.4}$$

$$\text{Translation invariance: } X_h \oplus B = (X \oplus B)_h \tag{2.5}$$

An example of a binary dilation is shown in figure 2.2.

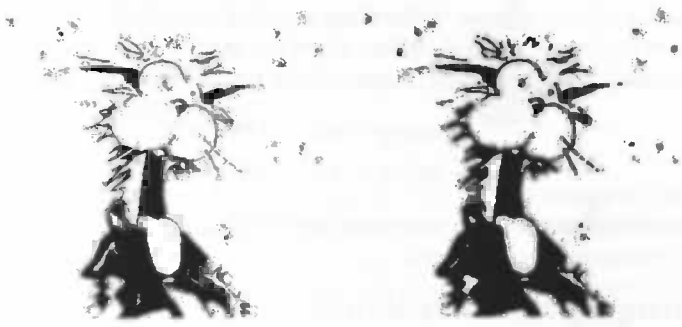


Figure 2.2: An example of a binary dilation performed on an image. The left image is the original image and the right image is the original image after dilation

A *conditional dilation* [2] is a binary dilation, where the dilated image is compared to a certain reference image, such that image parts that are not present in the reference image, also not appear in the conditional dilated image. This is implemented by an and operator. The resulting image contains only foreground pixels that are present in both the dilated image and the reference image.

2.3 Binary Erosion

The erosion $X \ominus B$ [2] is the dual operator of dilation. This operator combines two point sets using vector subtraction of the set elements; the result of the erosion is given by those points p for which all possible $p + b$ are in X:

$$X \ominus B = \{p \in M : p + b \in X \text{ for every } b \in B\} \tag{2.6}$$

An example of a binary erosion can be seen in figure 2.3.

Figure 2.3: An example of a binary erosion performed on an image. The left image is the original image and the right image is the original image after erosion

2.4 Openings and Closings

When an image is first eroded and then dilated, the original image is not obtained again. Erosion followed by dilation of two images is therefore another morphological transformation, called the *opening* [6]. When performing the dilation before the erosion, a *closing* [6] is obtained. Openings and closings are used for eliminating image details smaller than the structuring element.

$$\text{Opening } X \circ B = (X \ominus B) \oplus B \tag{2.7}$$

$$\text{Closing } X \bullet B = (X \oplus B) \ominus B \tag{2.8}$$

In figure 2.4 are examples of binary openings and closings.

2.5 Opening by reconstruction

Opening by reconstruction [7] is a connected filter (see section 2.7), because it deletes components from the image that do have a certain property (which is related to the width of the component). When performing an opening by reconstruction, the image is first eroded multiple times until the components that have to be deleted are all gone. The remaining parts of the other components are then reconstructed, by using multiple conditional dilations. An opening by reconstruction therefore does not change the shape of the connected components. An example of an opening by reconstruction can also be seen in figure 2.4.

2.6 Operator properties

An (morphological) image operator α_r is called:

- *anti-extensive* if the operator does not add anything to the binary image X, but can only remove pixels from this set, i.e.

$$\alpha_r(X) \subset X, \tag{2.9}$$

- *increasing* when it has the following properties on two images X and Y:

$$X \subset Y \Rightarrow \alpha_r(X) \subset \alpha_r(Y), \tag{2.10}$$

- *idempotent* when there is no additional effect on the image when the operator is applied multiple times:

$$\alpha_r(\alpha_r(X)) = \alpha_r(X), \tag{2.11}$$

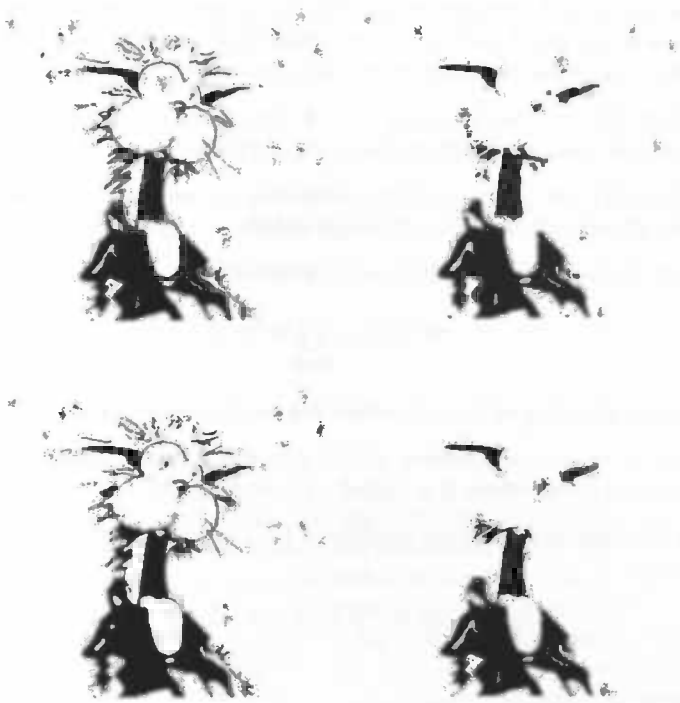


Figure 2.4: An example of a openings and closings. The top left image is the original image. In the top right image we can see the same image after binary opening and in the bottom left is the original image after closing. The bottom right image is the result of an opening by reconstruction, where the image is first eroded two times and then conditional dilations were applied until the resulting image did not change anymore.

- *scale-invariant* if it does not matter whether the image is scaled first and then processed, or first processed and then scaled.

$$\alpha_r(X_\lambda) = (\alpha_r(X))_\lambda, \tag{2.12}$$

- *connected* when for any binary image X, the set difference $X \backslash \alpha_r(X)$ is exclusively composed of connected components of X or of its complement X^c.

An image can be *partitioned* by subdividing the image into its connected components. Partitioning of an image can be done on binary and grey-scale images, as well as on color images, as long as there are differences in color between the connected components. A connected image operator can only merge flat zones and/or assign new color (or grey-level) values to them.

2.7 Connected Morphological Operators

In this section we will give the definitions of other important image operators. These operators are called *connected operators* because they work on the connected components [7] of an image.

A binary connected opening [6] of X at a certain point $x \in X$ extracts the connected component of X containing x. This can be done, for example, by taking an image with only the selected pixel and then performing multiple conditional dilations until the connected component is obtained. The notation for a binary connected opening at a point x is Γ_x.

Binary area openings [6] are based on binary connected openings. A binary connected opening first extracts the connected component to which a point x belongs. The binary area opening then performs on every pixel such a binary connected opening and will remove a connected components if the size of the obtained connected component is smaller than a threshold value λ.

Attribute openings [1] are used to make use of other filtering criteria than size and width. A criterion (attribute) is used on which the filter should respond.

Attribute thinnings [1] use non-increasing attributes to use the shape rather than size criteria. This type of filter allows extraction of all image details of a given shape, regardless of their sizes.

A binary attribute thinning Φ^T of a set X with criterion T is given by:

$$\Phi^T(X) = \bigcup_{x \in X} \Phi^T(\Gamma_x(X)) \tag{2.13}$$

The binary attribute thinning of X is therefore the set of connected sets which all satisfy T.

A shape operator is an image operator which is scale, rotation and translation invariant. If a shape operator is also idempotent, it is called a shape filter [8].

A binary shape distribution is a set of operators $\{\beta_r\}$ which are anti-extensive, idempotent and invariant to scaling.

$$\text{Idempotence: } \beta_r(\beta_s(X)) = \beta_{max(r,x)}(X) \tag{2.14}$$

Chapter 3

The *Max-Tree*

3.1 Introduction

In this chapter, an explanation of the *Max-Tree* algorithm will be given which can be used to implement grey-scale attribute filters. The *Max-Tree* algorithm extracts the connected components from an image, which are then stored in the nodes of the *Max-Tree*. For every connected component, the attribute on which the filter responds is also calculated (for example area). First a brief explanation about the *Max-Tree* is given, along with an algorithm to construct the *Max-Tree*. After that, an adaptation is made to the *Max-Tree* and an optimized version of the algorithm is presented. In the conclusion of this chapter, the reason why the *Max-Tree* algorithm is used in the research project is explained.

3.2 *Max-Tree* data structure

The *Max-Tree* is a data structure that stores the connected components of an *input image*. Every separate layer in the tree contains connected components of the same grey level value and every node in the tree contains the pixels from one connected component. This can be seen in figure 3.1.

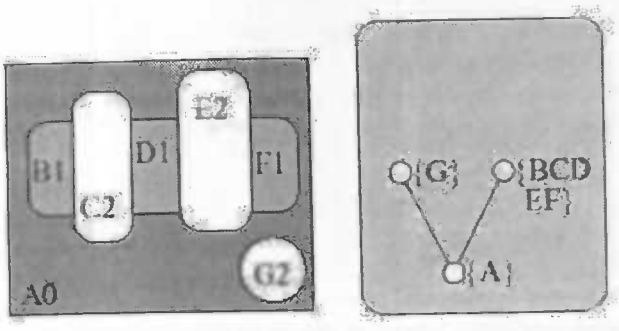


Figure 3.1: An example input image of seven connected components (A to G) in three different grey level values (0 to 2) and the constructed *Max-Tree*
(from Salembier *et al.* [7]).

When applying a connected attribute filter to the *Max-Tree* , the filter is simply applied to every node in the tree and therefore to every connected component. The *Max-Tree* is constructed in a way that every node can contain attribute information about the connected component it holds.

This attribute information is calculated while building the tree, if possible. Later in this report it can be seen that sometimes multiple passes over every node in the tree are needed to calculate the desired attribute information. When the attribute filter is applied to a certain node in the tree, the attributes of the node are tested against the criterion of the filter, and a decision is made if the node has to be deleted from the tree (and therefore the connected component from the input image).

3.3 The *Max-Tree* algorithm

From the previous section and the image in figure 3.1 it can be seen that the *Max-Tree* contains the connected components of an image. From now on, the grey level value of a node (i.e. the connected component that is stored in the node) is denoted by h. Since multiple nodes of the same grey level value can exist in a *Max-Tree* , they are numbered. By definition: Every node C_h^k in the tree represents the k'th connected component at grey level value h.

The *Max-Tree* algorithm first determines the lowest grey level value h_{min} present in the image and creates a node $C_{h_{min}}^0$. This is the root node of the tree. Then the image is thresholded at grey level value h_{min}, such that all pixels of grey level value h_{min} are assigned to be background pixels and all other pixels (that do have a grey level value larger than h_{min}) are assigned to be foreground pixels. A *temporary node* TC_h^k is created for every connected component of foreground pixels. This step is illustrated in figure 3.2. These child nodes are temporary nodes, because the pixels that are contained in the node still have different grey level values and do not meet the *Max-Tree* criterion (where every node at level h only contains pixels of grey level value h) yet.

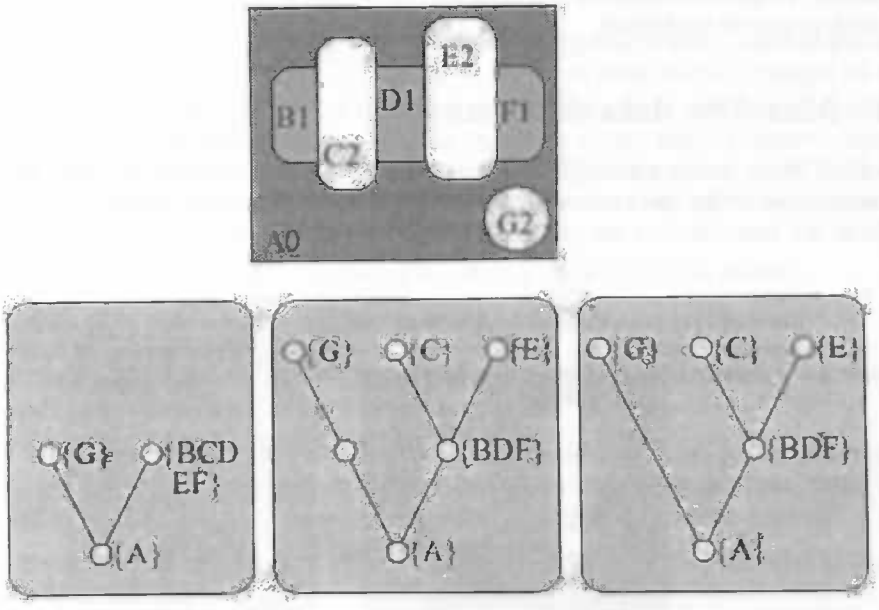


Figure 3.2: An example input image of seven connected components (A to G) in three different grey level values (0 to 2). At the bottom left the *Max-Tree* is shown after thresholding at grey level value 0. The background component (A) is stored in the root node and the two other components (G and BCDEF) are stored in child nodes. At the middle picture, the *Max-Tree* is show after thresholding at grey level value 1. The final tree is in the bottom right picture. (From Salembier et al. [7])

After this step, the algorithm is recursively repeated on the created temporary child nodes. The threshold is set at the *local background* of the node (the lowest grey level value found within the

node) and new temporary child nodes are created for every connected component with a grey level value larger than this threshold. This step can also be observed in figure 3.2. Note that after this step the temporary node which was handled now becomes a normal node of the *Max-Tree* , since there are only pixels left of the grey level value that matches the level of this node in the tree.

The last step of the algorithm is to delete empty nodes from the tree.

For this research project a slightly different version of the *Max-Tree* will be used. When thresholding a node in the *Max-Tree* , the foreground pixels do not move to new nodes but they are *copied* to the new nodes. In this way we can apply the shape filter to objects that contain of different components. For example, when considering the image in figure 3.2, the pixels of the components B,C,D,E and F will stay in the right node at level $h = 1$ and the pixels of nodes C and E are copied to two different nodes at level $h = 2$. This way, the shape filter can be applied to the components C, E, and also to the component BCDEF.

3.4 Thresholding

The thresholding used to build the *Max-Tree* can be either strict or soft. When using strict binarization, a new node is created in the tree for every component of a different grey level value than the background. When using soft binarization on the other hand, small fluctuations in grey level values are accepted within each connected component. This can be very helpful when, for example, the image was anti-aliased. When strict binarization is used on an anti-aliased image, a large number of small connected components would be detected on the border of the foreground component, since there is a transition around the border from foreground to background pixels. Soft binarization could solve this problem if the transition is not too sharp.

When strict binarization is used to create the *Max-Tree* , the following criterion is used for each node C_h^k:

$$C_h^k = \{(i,j) \in TC_h^k \text{ such that } f(i,j) = h\} \tag{3.1}$$

This means that the final node C_h^k will only contain the pixels (i,j) from the temporary node TC_h^k if and only if the grey level values $f(i,j)$ are exactly h. The remaining pixels for which $f(i,j) > h$ are stored in new temporary child nodes.

When soft binarization is used to create the nodes of the *Max-Tree* , the criterion is adapted to allow small fluctuations in grey level value: According to Salembier et al. [7] this soft binarization is defined by:

$$\begin{aligned} C_h^k = \{&(i,j) \in TC_h^k \text{ such that} \\ &\text{either } f(i,j) = h \\ &\text{or } \exists (i',j') \in C_h^k \text{ and neighbor of} \\ &(i,j), ||f(i,j) - f(i',j')|| \leq \Delta\} \end{aligned} \tag{3.2}$$

with Δ a bound on the gray level fluctuations allowed. This means basically that pixels of grey level h are allowed in the node C_h^k, as well as pixels that have grey level value within a distance Δ of neighboring pixels that are already accepted to belong to C_h^k. This basically means when an object is filled with a gradient coloring, the component can still be classified as one node in the *Max-Tree* , because the amount of the transitions in grey level value is very small. Please note that this definition of soft binarization is recursively defined in C_h^k. This may cause problems, concerning the complexity of the algorithm. Since this soft binarization is not used in the research project, it is not necessary to try and solve this.

3.5 Optimizing the *Max-Tree* algorithm

The objective of this section is to describe a fast implementation of *Max-Tree* creation. The implementation relies on the use of a hierarchical FIFO (first-in-first-out) queue, that is a set of FIFO queues, where each individual queue is assigned to a particular grey level value h. These queues are used to define an appropriate scanning and processing order of the pixels. Four functions are defined on the queues:

- **hqueue-add(h,p)**: Add the pixel p (of grey level h) in the queue of priority h.
- **hqueue-first(h)**: Extract the first available pixel of the queue of priority h.
- **hqueue-empty(h)**: Return *true* if the queue of priority h is empty.
- **number_nodes(h)**: Return the number of nodes C_h^k at level h.

`ORI(p)` denotes the original grey level value of pixel p and `STATUS(p)` stores the information of the pixel status. This status can be "not-analyzed", "in-the-queue" or assigned to node k of level h in which `STATUS(p)` $= k$.

A recursive flooding algorithm is used for *Max-Tree* creation. The pixel with the lowest grey level value h_min is used as a starting point for this flooding algorithm. This algorithm has two basic parts: the first part actually performs the propagation and the updating of the `STATUS`, whereas the second part defines the parent-child relations.

3.6 Filtering techniques for *Max-Tree's*

When the *Max-Tree* is constructed the attribute filter can be applied to every node of the *Max-Tree*. To determine whether a node (flat zone) C_h^k has to be removed from the tree (image) its attribute ($S(C_h^k)$) is tested against a reference attribute r and a decision is made whether the node should be deleted or not. For example, when applying an area filter which should remove components which have an area smaller than 100 pixels, the attribute $S(C_h^k)$ of a node contains the area of the component in that node, and this area will be deleted if the value of this attribute is smaller than 100.

Five different criteria can be used to determine whether a node should be deleted:

- *Min* A node C_h^k is removed if $S(C_h^k) < r$ or if one of its ancestors is removed.
- *Max* A node C_h^k is removed if $S(C_h^k) < r$ and all of its descendant nodes are removed as well.
- *Viterbi* The removal and preservation of nodes is considered as an optimization problem. For each leaf node the path with the lowest cost to the root node is taken, where a cost is assigned to each transition. This can be implemented by Trellis-construction [7].
- *Direct* A node C_h^k is removed if $S(C_h^k) < r$; its pixels are lowered in grey level to the highest ancestor which meets S.
- *Subtractive* A node C_h^k is removed if $S(C_h^k) < r$; its pixels are lowered in grey level to the highest ancestor which meets S. The descendants of this node are lowered by the same amount as C_h^k itself.

When increasing attributes (like area) are used as a criterion, these filtering techniques are all the same. Since shape is a non-increasing attribute, these techniques will result in different output images. For the research project, the *direct* criterion will be mostly used, because we want to observe every node by itself and a node should not be deleted or changed if one of its ancestors or children does not meet the shape criterion. The *subtractive* method would also work, but this filtering method reduces the contrast of the image, which the *direct* criterion does not.

Algorithm 1 function flood(h)

```
{part 1}
while not hqueue-empty(h) do
  p ← hqueue-first(h)
  STATUS(p) ← number_nodes(h)
  for every neighbor q of p do
    if STATUS(q) == "Not-analyzed" then
      hqueue-add(ORI(q), q)
      STATUS(q) ← "In-the-queue"
      node-at-level(ORI(q)) ← true
      if (ORI(q) > ORI(p)) then
        m = ORI(q)
        repeat
          m ← flood(m)
        until m = h
      end if
    end if
  end for
end while
number-nodes(h) ← number-nodes(h) + 1

{part 2}
m ← h - 1
while m ≥ 0 and node-at-level(m) = false do
  m ← m-1
end while
if m ≥ 0 then
  i ← number-nodes(h) - 1
  j ← number-nodes(m)
  parent of node i at level h ← node j at level m
else
  node i at level h is root node
end if
node-at-level(h) ← false
return m
```

3.7 Example: moment of inertia

A possible connected attribute filter to use is a filter which uses the elongation criterion. For calculating the elongation of a flat zone, the moment of inertia of the flat zone is needed. The 2D moment of inertia $I(C)$ of a component C with area $A(C)$ is defined by [6]:

$$I(C) = \frac{A(C)}{6} + \sum_{x \in C} (x - \bar{x})^2 \tag{3.3}$$

And in 3D, the moment of inertia is defined by:

$$I(C) = \frac{V(C)}{4} + \sum_{x \in C} (x - \bar{x})^2 \tag{3.4}$$

where $V(C)$ denotes the volume of the object. It can be seen from these equations, that when a pixel is added to a node in the *Max-Tree* , that the moment of inertia attribute of that node can instantly updated

The moment of inertia can be used in an attribute filter when taking elongation as the filtering criterion. Elongation of objects can be represented by the shape-dependent ratio $S = \frac{I}{A^2}$ in $2D$ and $S = \frac{I}{V^{(5/3)}}$ in $3D$ which has a minimum for a sphere and increases rapidly with elongation [9].

Chapter 4

Moment-based shape descriptors

4.1 Introduction

In this chapter we will describe three different moment-based shape descriptors: geometric moments, Hu moment invariants[3] and Krawtchouk moment invariants [10]. Hu moment invariants and Krawtchouk moment invariants are based on geometric moments and are tested for their invariance against scaling, rotation and translation. In this chapter, the definition, as well as the characteristics of the different types of moments will be described.

4.2 Geometric moments

The set of geometric moments of a two-dimensional image $f(x, y)$ is defined [3] by:

$$M_{nm} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^n y^m f(x, y) dx dy \tag{4.1}$$

where n and m can take on all positive natural numbers and $(n + m)$ is called the order of the moment. Geometric moments can be understood as a measure for the shape of an object which depends on the locations of the foreground pixels in an image and the order of the moment. In the binary case, the background pixels (with $f(x, y) = 0$) do not contribute to the value of the geometric moments. Since the foreground pixels contribute a factor 1 (because $f(x, y) = 1$ in binary images), the geometric moments of an image can also be calculated by summing over all foreground pixels:

$$M_{nm} = \sum_{\Omega_{in}} x^n y^m dx dy \tag{4.2}$$

where Ω_{in} is the set of foreground pixels. When M_{00} is calculated, the equation is simplified with the area of the image in pixels as the result.

4.3 Hu moment invariants

Since moments will be used as a shape descriptor in the research project, it is very important that these moments are invariant to three image transformations: translation, rotation and scaling. This is because none of these transformations have an effect on the shape of the image, so the moments that are calculated should not change under any of these transformations.

The *geometric* moments of an image change if the image is translated. This difference can be seen in figure 4.3. This effect can be compensated when the center of mass of the image is calculated. The image is then translated such that the center of mass will be at the origin of the image:

$$\mu_{n,m} = \int \int_{\Omega_{in}} (x - \bar{x})^n (y - \bar{y})^m dx dy \tag{4.3}$$

This set of moments, which are invariant to translation, are called *central moments*.

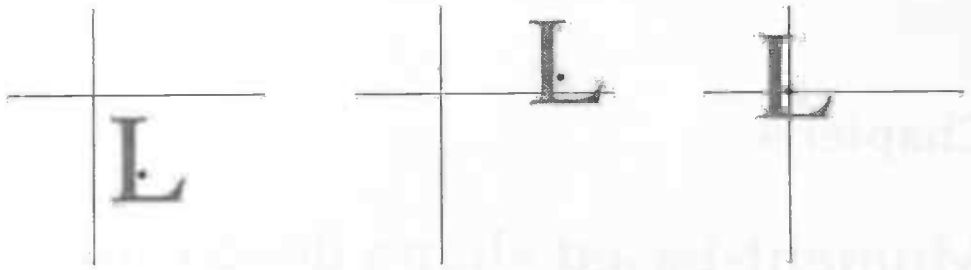


Figure 4.1: Translation invariance: the first image is the original image of the letter L. In the second image the L has been translated. The moments of these two images will not be identical, unless they are both translated to the origin as can be seen in the third image. The dot in the images represents the center of mass of the image.

Rotation of an object also does not affect the shape of the object. However, also this transformation causes the moments of the image to change. This can be seen in figure 4.3. To make the system rotation invariant, both reference and input images have to be aligned. This is done by locating the longest axis in each image. A rotation angle θ of this axis can be calculated. In the calculation of the moments, this rotation angle θ is added:

$$\tilde{\mu}_{n,m} = \int\int_{\Omega_{in}} [(x-\bar{x})\cos\theta + (y-\bar{y})\sin\theta]^n [(y-\bar{y})\cos\theta - (x-\bar{x})\sin\theta]^m dxdy \tag{4.4}$$

where

$$\theta = \frac{1}{2}\tan^{(-1)}\frac{2\mu_{11}}{\mu_{20}-\mu_{02}} \tag{4.5}$$

A problem that can occur with this approach is that the axis are aligned, but have a rotation angle

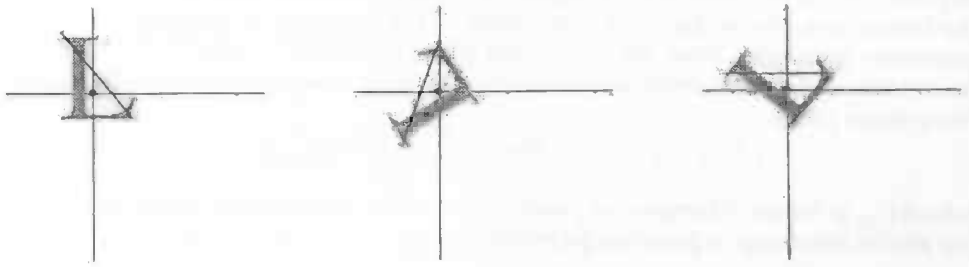


Figure 4.2: Rotation invariance: the first image is the original image of the letter L. In the second image the L has been rotated. The moments of these two images will not be identical, unless they are both rotated, such that the longest axes through the images align, as can be seen in the third image.

of 180 degrees between them. As we will see later, this will become a problem and a solution is presented.

Scale invariance is also a very important property of a shape-descriptor, since the size of an object does not determine its shape. This can be solved by applying a normalization factor proportional to some power of the area of the image (M_{00}). When this normalization factor is applied to

Order	n	m	Left image	Right image
0	0	0	222.0	468.0
1	0	1	3639.0	21918.0
	1	0	2478.0	18660.0
2	0	2	76839.0	1062022.0
	1	1	40539.0	875814.0
	2	0	35338.0	827996.0
3	0	3	1828575.0	53174628.0
	1	2	870747.0	42520526.0
	2	1	577521.0	39043290.0
	3	0	582642.0	39572436.0

Table 4.1: Regular geometric moments for the images of figure 4.3.

equation 4.4, the following definition of moment invariants is obtained:

$$\nu_{n,m} = M_{00}^{-\frac{n+m+2}{2}} \int\int_{\Omega_{in}} [(x-\bar{x})cos\theta + (y-\bar{y})sin\theta]^n \cdot [(y-\bar{y})cos\theta - (x-\bar{x})sin\theta]^m f(x,y) \tag{4.6}$$

We have now obtained a set of geometric moment invariants, which are independent of rotation, scaling and translation. This set of moment invariants are the Hu moment invariants [3].

4.4 Examples

As an illustration and an example, the regular geometric moments of the images in figure 4.3 as well as the moment invariants of these images are calculated.

Figure 4.3: Two sample images of the letter E. In the second image, the letter E has been translated, rotated and scaled relative to its position in the first image

In table 4.1 the standard geometric moments from the image in figure 4.3 are shown. Although both images represent the letter E, the geometric moments of the two images are not similar at all. However when using the moment invariants (table 4.2 the moments are still not identical but do show large similarities.

The error between reference image and input image of the moments from table 4.2 is caused by a phenomenon that we have to look out for: The rotation invariance aligns the longest axes through the input image and the reference image, but they are aligned, such that one of the images is 180 degrees rotated from the other image. The axes do align, but one of the two letters is upside down. This can for example be observed by looking at the M_{02} moment of the left image and the M_{20} moment of the right image. These are basically the same, which indicates that the images

Order	n	m	Left image	Right image
0	0	0	1.000	1.000
1	0	1	0.000	0.000
	1	0	0.000	0.000
2	0	2	0.349	0.162
	1	1	0.000	0.000
	2	0	0.001	0.384
3	0	3	0.007	0.023
	1	2	0.001	−0.001
	2	1	0.000	0.025
	3	0	0.000	−0.026

Table 4.2: Geometric moment invariants for the images of figure 4.3.

really were misaligned. When a reference image is used that is a rotated version of the left image in figure 4.3, the moments are even more similar than in table 4.2.

Due to this problem, it will sometimes be better to use two reference images. We will come back to this issue in chapter 5.6. It is a serious problem, because in some cases (especially when using rather small images), the error between the reference image and the object to be recognized will be even higher than the error between the reference image and the background. Of course this problem is solved, when using two reference images that are each other's mirror image.

4.5 Krawtchouk moment invariants

Recently, a new method for image analysis based on *Krawtchouk moment invariants* [10] has been presented. Krawtchouk moment invariants are based on Krawtchouk polynomials and the Hu moment invariants which were described in the previous section. Krawtchouk moment invariants and are supposed to be more robust against translation, rotation and scaling. In the chapter containing the results, it can be seen whether this is the case.

4.5.1 Krawtchouk polynomials

The n-th order Krawtchouk polynomial is defined by:

$$K_n(x|P,N) = \sum_{k=0}^{N} a_{k,n,p} x^k = {}_2F_1(-n,-x,-N;\frac{1}{p}) \tag{4.7}$$

where $x, n = 0, 1, 2...N, N > 0$ and $p \in (0,1)$. ${}_2F_1$ is a hypergeometric function defined by:

$${}_2F_1(a,b,c;z) = \sum_{k=0}^{\infty} \frac{(a)_k (b)_k}{(c)_k} \frac{z^k}{k!} \tag{4.8}$$

$$(a)_k = a(a+1)...(a+k-1) \tag{4.9}$$

Examples of Krawtchouk polynomials are:

$$\begin{aligned} K_0(x|p,N) &= 1 \\ K_1(x|p,N) &= 1 - \left[\frac{1}{Np}\right] x \\ K_2(x|p,N) &= 1 - \left[\frac{2}{Np} + \frac{1}{N(N-1)p^2}\right] x + \left[\frac{1}{N(N-1)p^2}\right] x^2 \end{aligned} \tag{4.10}$$

To ensure numerical stability on Krawtchouk moments, normalization can be applied to the Krawtchouk polynomials. Normalized Krawtchouk polynomials $\bar{K}_n(x|p,N)$ are deducted from regular Krawtchouk polynomials, which are divided by the square root of their norm $\rho(n|p,N)$ which is defined by:

$$\rho(n|p,N) = (-1)^n \left(\frac{1-p}{p}\right)^n \frac{n!}{(-N)^n} \tag{4.11}$$

Even with this normalization factor, stability of the Krawtchouk polynomials is not guaranteed. Therefore, in addition to normalizing the polynomials with the norm, the square root of the weight $w(x|p,N)$ is also introduced as a scaling factor:

$$w(x|p,N) = \binom{N}{x} p^x (1-p)^{(N-x)} \tag{4.12}$$

which is actually equal to the formula used for binomial probability distributions. The set of weighted Krawtchouk polynomials $\bar{K}_n(x|p,N)$ can now be defined by:

$$\bar{K}_n(x|p,N) = K_n(x|p,N)\sqrt{\frac{w(x|p,N}{\rho(n|p,N)}} \tag{4.13}$$

4.5.2 Recurrence relations for Krawtchouk polynomials

Krawtchouk polynomials have the following recurrence relationship [5]:

$$p(N-n)K_{n+1}(x;p,N) = [p(N-n) + n(1-p) - x]K_n(x;p,N) - n(1-p)K_{n-1}(x;p,N). \tag{4.14}$$

This can be rewritten as

$$K_{n+1}(x;p,N) = \left[1 + \frac{n(1-p)}{p(N-n)} - \frac{x}{p(N-n)}\right]K_n(x;p,N) - \frac{n(1-p)}{p(N-n)}K_{n-1}(x;p,N), \tag{4.15}$$

which yields

$$K_{n+1}(x;p,N) = \left[1 + \frac{n(1-p)}{p(N-n)} - \frac{x}{p(N-n)}\right]\sum_{k=0}^{N} a_{k,n,p}x^k - \frac{n(1-p)}{p(N-n)}\sum_{k=0}^{N} a_{k,n-1,p}x^k. \tag{4.16}$$

By reorganizing the summations we have

$$K_{n+1}(x;p,N) = \sum_{k=0}^{N}\left[1 + \frac{n(1-p)}{p(N-n)} - \frac{x}{p(N-n)}\right]a_{k,n,p}x^k - \frac{n(1-p)}{p(N-n)}a_{k,n-1,p}x^k. \tag{4.17}$$

This in turn is rewritten as

$$K_{n+1}(x;p,N) = \sum_{k=0}^{N}\left[a_{k,n,p} + \frac{n(1-p)}{p(N-n)}(a_{k,n,p} - a_{k,n-1,p})\right]x^k - \frac{a_{k,n,p}}{p(N-n)}x^{k+1} \tag{4.18}$$

By defining that $a_{k,n,p} = 0$ for $k < 0$ and $k > n$, and reordering summations we have

$$K_{n+1}(x;p,N) = \sum_{k=0}^{N}\left[a_{k,n,p} + \frac{n(1-p)}{p(N-n)}(a_{k,n,p} - a_{k,n-1,p}) - \frac{a_{k-1,n,p}}{p(N-n)}\right]x^k. \tag{4.19}$$

This yields a recurrence relationship

$$a_{k,n+1,p} = a_{k,n,p} + \frac{n(1-p)}{p(N-n)}(a_{k,n,p} - a_{k,n-1,p}) - \frac{a_{k-1,n,p}}{p(N-n)}. \tag{4.20}$$

Note that $a_{0,n,p} = 1$ and $a_{n,n,p} = a_{n-1,n,p}/(Np - np)$. By filling in the coefficients from (4.10) we can now derive

$$\begin{aligned} K_3(x;p,N) = 1 &- \left[\frac{3}{Np} + \frac{2}{N(N-1)p^2} + \frac{pN+2}{N(N-1)(N-2)p^3}\right]x \\ &+ \left[\frac{2}{Np} + \frac{2}{N(N-1)p^2} + \frac{2-2p}{N(N-1)(N-2)p^3}\right]x^2 \\ &- \frac{1}{N(N-1)(N-2)p^3}x^3, \end{aligned} \tag{4.21}$$

and so on for larger n.

4.5.3 Krawtchouk moments

Krawtchouk moments can be used for local feature extraction, while other techniques provide us only with global feature extraction. For the definition of Krawtchouk moments, we will go to the same notation as used in [10]. This means that the images analyzed are images of N by M pixels. The Krawtchouk moments are based on Krawtchouk polynomials and are defined by:

$$Q_{nm} = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} \bar{K}_n(x;p_1,N-1)\bar{K}_m(y;p_2,M-1)f(x,y) \tag{4.22}$$

Krawtchouk moments can also be made invariant to scaling, translation and rotation. This is basically done in the same way as the Hu moment invariants and therefore, the same calculations can be used. However, for Krawtchouk moment invariants a slightly different definition of Hu moment invariants are used:

$$\begin{aligned} \tilde{\nu}_{nm} &= \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{N^2}{2M_{00}} f(x,y) \\ &\times \left([(x-\bar{x})\cos\theta + (y-\bar{(}y)\sin\theta\sqrt{\frac{N^2/2}{M_{00}}} + \frac{N}{2}\right)^n \\ &\times \left([(x-\bar{x})\cos\theta + (y-\bar{(}y)\sin\theta\sqrt{\frac{N^2/2}{M_{00}}} + \frac{N}{2}\right)^n \end{aligned} \tag{4.23}$$

where N is the width of the input image. The Krawtchouk moment invariants can now be defined by:

$$\bar{Q}_{nm} = [\rho(n),\rho(m)]^{-\frac{1}{2}} \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i,n,p_1}a_{j,m,p_2}\tilde{\nu}_{ij} \tag{4.24}$$

where $a_{k,n,p}$ are coefficients determined by equation 4.7.

When we take a closer look at equation 4.24, we can see that it is dependent of the width of the image. For example, when we calculate the zeroth order moments $\tilde{\nu}_{00}$ we obtain:

$$\tilde{\nu}_{00} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{N^2}{2M_{00}} f(x,y) \times 1 \times 1,$$

which can be reduced to:

$$\frac{N^2 M_{00}}{2M_{00}} = N^2/2$$

So the Krawtchouk moment invariants will also be dependent of the size of the image. This is a side effect we do not want, because we will get extreme difficulties when comparing a reference image with a different sized input image, unless we take the same value for N. This is also not what we want, when the moments calculation is done as a pre-processing step, where the moments are stored with the images for later comparison.

When presenting the results obtained with Krawtchouk moment invariants, we normalized the moments vector from both the reference and the input image by dividing them by the length of the moments vector from the reference image.

4.5.4 Local feature extraction

The parameters p_i, which are used in the calculation of Krawtchouk moments and which were assumed to be given can be used to extract local features of an image. The parameter p_1 is used to shift the region of interest horizontally and p_2 can be used to shift this region vertically. As before, $p_i \in (0, 1)$. When using for example $(p_1, p_2) = (0.1, 0.1)$ the top-left region of the image will only be described by the Krawtchouk moment invariants.

With this information, we can use fewer moments when we create a feature vector, which consists of sets of moments of different region of interests of the image. In this way, the image can be described with more accuracy, without using larger order of moments.

4.5.5 Image reconstruction

When the Krawtchouk moments are calculated, they can be tested for accuracy by reconstructing the image they were calculated from. Since the Krawtchouk moments can be calculated by using a function of the image, the image can be calculated by using a function of the Krawtchouk moments:

$$f(x, y) = \sum_{n}^{N-1} \sum_{m}^{M-1} Q_{nm} \bar{K}_n(x|p, N-1) \bar{K}_m(y|q, M-1) \tag{4.25}$$

We have tested the Krawtchouk moments by taking the input image of figure 4.4, calculating the Krawtchouk moments and then reconstructing the image. The region of interest was chosen in the exact center of the image and hence $p_1 = p_2 = 0.5$. The results can be seen in figure 4.5. These results have been thresholded, since the input image was a binary image as well. The running time of this test was less than one tenth of a second, including the writing of the result image. This short running time is explained by taking a look at the complexity of the reconstruction algorithm. The complexity of Krawtchouk moments calculation is linear in the number of pixels, and the reconstruction step is also. Since we used a very small image in this case, the running time of the algorithm was, as expected, very low.

It can be seen that 12th order Krawtchouk moments can provide us with enough information about the image to perform an almost complete reconstruction of the image without much error; and even 7th order moments provide us with good information about the shape of the image.

Figure 4.4: The image used for testing the Krawtchouk moments. The image is a binary image of size 30x30 pixels.

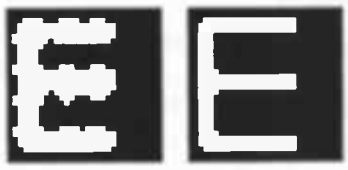


Figure 4.5: Results after image reconstructing. The left image was reconstructed with help of 7th order Krawtchouk moments and the right image was reconstructed with help of 12th order Krawtchouk moments. Both images are thresholded at grey level value 127.

Chapter 5

Creating the vector attribute shape filter

5.1 Introduction

In this section I will explain how the *Max-Tree* algorithm is adapted to support vector attributes like the moment invariants. The attribute information cannot be calculated fully during construction of the *Max-Tree* , so some extra calculations on the tree have to be performed, before the filtering steps can take place. In the examples and results of this section, we assume that all reference images used are binary images that only contain one connected component which represents our reference shape.

5.2 Handling vector attributes

Since the *Max-Tree* algorithm only uses single valued attributes, a change to the *Max-Tree* datastructure has to be made. Furthermore, when a pixel is added to a node in the *Max-Tree* , a whole set of attributes has to be updated. How this is done is shown in the following subsections. When the filter is applied to the *Max-Tree* , vector comparison has to be done in order to find a measure of error between the reference image, and the connected component stored in the node of the *Max-Tree*. This error can for example be measured by calculating the Euclidean distance between the vectors, or the angle between the vectors. Different methods of error calculation are used for the different types of shape attributes that were implemented. The details about this can also be found in the following to sections.

The first thing that has to be determined is the length of the attribute vector. Since moments are used a shape descriptor and the o'th order of moments is defined by a set of $o+1$ moments, the length of the attribute vector when using moments *up to* the order o will be $\frac{1}{2}o(o+1)$.

5.3 Handling geometric moments

When using geometric moments as a shape descriptor, only small adaptations are needed. However, since both Hu moment invariants, as well as Krawtchouk moment invariants can be calculated from geometric moments, it is better to implement this shape filter in a separate step.

Recall that geometric moments of a binary image are defined by:

$$M_{nm} = \sum \sum_{\Omega_{in}} x^n y^m \tag{5.1}$$

This calculation can be used, because each node in the *Max-Tree* represents only one binary connected component. When a node is created, it will contain only the one pixel (x, y) that forced the algorithm to create the node. Therefore the moment of order (n, m) of that node can be set to $x^n \cdot y^m$.

Every time a pixel (x, y) is added to this node, the moment can just be updated, by adding x^n, y^m to the moment of order (n, m). After the construction of the *Max-Tree* is complete, every node will contain a set of moments as its attribute information and these sets of moments are equal to the geometric moments of the component stored in these nodes.

When a filter is applied to the nodes in the *Max-Tree* , a comparison between the moments from the reference image and the moments from every node in the tree has to be done. The most intuitive way to calculate the error E between two vectors $\vec{v}$ and $\vec{w}$ is to calculate the Euclidean distance between them:

$$E(v, w) = \sqrt{\sum_{i=1}^{\text{length}} (\vec{v}_i - \vec{w}_i)^2} \tag{5.2}$$

When a threshold is set as the maximum error allowed, all components with an error smaller than this threshold will be "similar enough" to the reference image and therefore deleted from the *Max-Tree*.

5.4 Handling Hu moment invariants

Hu moment invariants cannot be easily calculated during tree construction, because the angle of rotation of a component, as well as its center of mass are not known during tree construction. Therefore another approach will be used which will require us to do some post-processing on the *Max-Tree*.

During tree construction, every node will be updated when a pixel is added to that node. However, the Hu moment invariants cannot be calculated directly during *Max-Tree* construction. Therefore only the center of mass of the component that is contained in a node will be updated during this phase as well as the total area (M_{00}) of that component. When the *Max-Tree* is built, only attribute information in the form of the center of mass of every component is available. With this information, the angle of rotation for every node can be calculated with help of equations 4.3 and also the scaling factor can be calculated. This done in a first pass through every node in the *Max-Tree*. 4.5. In a second pass through every node in the *Max-Tree* , the Hu moment invariants can now be calculated with help of equation 4.6. The pseudo-code for this algorithm which calculates the vector attribute that contains Hu moment invariants is show in algorithm 2

It is possible to calculate Hu moment invariants directly from the geometric moments that were calculated in the previous section [3]. This would improve the running time a lot since this implementation does not require us to do calculations on every pixel in a node, but just on the set of geometric moments of that node. However, for every order of moments, different equations have to be used to calculate the Hu moment invariants from the geometric moments. Because the number of moments used has to stay variable for testing purposes, the more inefficient implementation was chosen. Another advantage of using a step by step calculation of Hu moment invariants is that the code stays more structured and is easier to check.

5.5 Handling Krawtchouk moment invariants

Krawtchouk moment invariants are calculated in a very similar way as Hu moment invariants. From the previous chapter we have seen that a modified version of the Hu moment invariants is used for the calculation of the Krawtchouk moment invariants (equation 4.24). Therefore we

Algorithm 2 Hu moment invariants calculation in the *Max-Tree* algorithm

{During Tree construction}
for all pixel (x, y) in the image **do**
  Update the center of mass, as well as the area of the node in which (x, y) is stored
end for
{Pass 1}
for all nodes n in the *Max-Tree* **do**
  Calculate angle θ of node n
end for
{Pass 2}
for all nodes n in the *Max-Tree* **do**
  Calculate Hu moment invariants ν for node n using equation 4.6
end for

will re-use the algorithm for Hu moment invariants and add an extra pass through the *Max-Tree* algorithm to calculate the Krawtchouk moment invariants from the Hu moment invariants. The new algorithm can be seen in figure 3

Once again, a big optimization can be made here, since the Hu moment invariants can be calculated directly from the geometric moments of an image. However the complexity of the algorithm stays linear in the number of pixels in the image.

Algorithm 3 Krawtchouk moment invariants calculation in the *Max-Tree* algorithm

{During Tree construction}
for all pixel (x, y) in the image **do**
  Update the center of mass, as well as the area of the node in which (x, y) is stored
end for
{Pass 1}
for all nodes n in the *Max-Tree* **do**
  Calculate angle θ of node n
end for
{Pass 2}
for all nodes n in the *Max-Tree* **do**
  Calculate modified Hu moment invariants $\tilde{\nu}$ for node n
end for
{Pass 3}
for all nodes n in the *Max-Tree* **do**
  Calculate Krawtchouk moment invariants from modified Hu moment invariants for node n using equation 4.24
end for

5.6 Implementing a nearest neighbor classifier

To modify the *Max-Tree* algorithm such that it can be used as a nearest neighbor classifier, we basically only need to add the option of providing multiple reference images to the algorithm.

The algorithm will read all reference image and will create an array of feature vectors. These feature vectors contain the desired type of moments of the reference images. When the *Max-Tree* is built and the connected filter is applied, the moments of a node are compared to the feature vectors. This will result in a number of measured errors. If the smallest error measured is within the maximum error set by the user, the node will be deleted from the *Max-Tree's*

There is another implementation of the nearest neighbor classifier that can be used. Once again, multiple reference images are used, but for each reference image, the class to which the image belongs is stored as well. The algorithm can therefore be supplied with a number of reference images that all represent a shape X and also a number of reference images of shape Y. In this way, the *Max-Tree* algorithm will work as a classifier and can separate nodes from shape X and Y from each other. Nodes which are of neither shape class will just be preserved.

A third variant of the nearest neighbor algorithm is that not the one nearest neighbor will determine the shape class of a node, but that the N nearest neighbors determine it. If, for example, three of the nearest neighbors are of class X and seven are of class Y, then the node will be classified as being of shape Y.

5.7 Complexity

The complexity of the *Max-Tree* algorithm, which includes image segmentation, filtering and image restitution is of order $O(GN)$ where N denotes the number of pixels in the input image and G denotes the number of occupied grey levels in the *Max-Tree* . This complexity is achieved when the components lie on top of each other (where every component is slightly smaller than the component under it) and the component with the highest grey level value is on top. In the filtering step, we will now have to go through every node below the node that is currently being analyzed. Since, in the worst case, there are N nodes, the complexity of the algorithm is $O(GN)$.

When using an input image of approximately 2200×2000 pixels, the algorithm will take about 44 megabytes of RAM. This is about ten times the image-size. The running time of the algorithm as presented on this input image is about one and a half minutes on a computer with a *Pentium 4* 2.8 GHz processor an 512 megabytes of RAM.

When a large number of reference image is used, for example when the algorithm is used as a nearest neighbor classifier, the complexity also scales with the number of reference images R and becomes $O(GN + RN)$.

Chapter 6

Results

6.1 Introduction

In this chapter, the results that were obtained during testing of the shape filters are presented. We will look at the degree of invariance against scaling, rotation and translation of both the Hu moment invariants, as well as the Krawtchouk moment invariants. We will also take a look at the problems that occur when using these moment invariants. Both small and large input images are used to test the algorithm for its memory usage and speed.

In every test we calculate the Hu moment invariants and the Krawtchouk moment invariants of orders two and three only. This is because the zeroth order moment (area) is normalized by the scale invariance of the system and is therefore constant. The first order moments are also constant due to the invariance and do not contain any information anymore.

The main goal of these tests is to see whether we can get enough information from these two orders of moments to do get reasonable results. The last test that is described in this chapter will give us some insights in how similar shapes are described by both the shape descriptors tested.

Due to some time constraints on the research project we have not tested the nearest neighbor classifier thoroughly. The classifier has been implemented into the algorithm and is ready for testing and usage, but no test results were obtained in this project.

6.2 Test case: Rotated E

In this section we will test Hu moment invariants and Krawtchouk moment invariants against rotation. The image in figure 6.2 is used, which will be rotated step by step. The errors between the original image and the rotated image will be measured after each step. The errors that are calculated with Hu moment invariants and Krawtchouk moment invariants cannot be directly compared, because we had to use a different normalization factor for Krawtchouk moment invariants. This is because Krawtchouk moment invariants are dependent on the size of the image. However, when graphs are drawn of the errors, we might get an impression about the rotation invariance of both shape descriptors.

In figure 6.2 we can see the rotated images which were used in this test. The errors between these images and the image is figure 6.2 is also presented in this image.

When the errors between the reference E and the input E's are calculated and plotted in graphs, the results from figure 6.2 are obtained. We can see that the error only increases fast for small rotations but does not change very much after that. When the rotation angle is set to 45 degrees, the error is smaller again. This is caused by the fact that no interpolation was used during the rotation of the image, which causes the 45 degree rotation of the image being a more precise version of the reference E than the other rotations.

Figure 6.1: A non-rotated image of a letter E. This image will be used as a reference image.

Figure 6.2: The input images used. The E from image 6.2 is rotated by an angle of 2, 5, 10, 20, 30 and 45 degrees respectively.

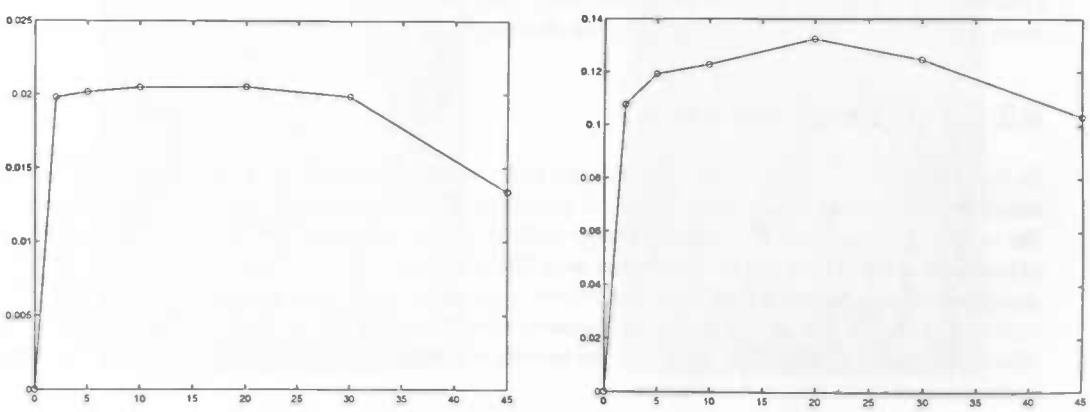


Figure 6.3: Graphs of the errors calculated between a the reference image of figure 6.2 and the rotated input images from figure 6.2. On the horizontal axes are the rotation angles in degrees and on the vertical axes are the errors measured when using Hu moment invariants (left image) and Krawtchouk moment invariants (right image).

6.3 Test case: ABC

In this second experiment, the image of figure 6.3 is used. In this image, we can see a large letter A, which contains B's and C's in different grey level values. Some letters are rotated or scaled and there are a large number of B's and C's in the image to test translation invariance. The geometry of the image is 2200 × 2000 pixels. This large image is chosen to take a closer look at the running time of the algorithm. The reference images used can be seen in figure 6.3

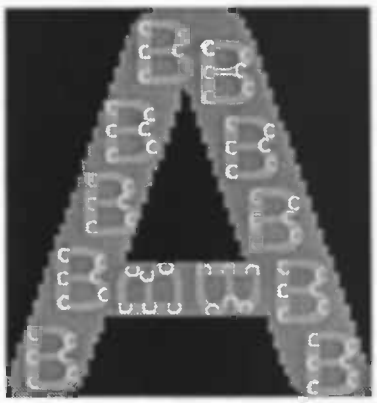


Figure 6.4: Input image of a large A with smaller B's and C's in it. The image is 2200 by 2000 pixels and there are 4 different grey level values in it.

In the first test, the letter 'A' is filtered from the image. We use a small reference image of the letter 'A' which contains 67 × 72 pixels (figure 6.3) so we can test both shape descriptors for scale invariance mainly. In figure 6.3 the results can be observed for both Hu moment invariants and Krawtchouk moment invariants. Both shape descriptors work well enough to get the A out of the image and leaving the B's and C's in place.

In the second test we try to filter the B's from the input image in figure 6.3. The results of this test are in figure 6.3. We can see from this image that when we use Krawtchouk moment invariants, all B's are correctly filtered, but when using Hu moment invariants, the rotated B's in the center are still present in the output image. Apparently the error measured between these B's and the reference image is rather large. If we would slightly increased the tolerance of the filter, such that components with a relatively high error are also filtered, the large A is filtered even before the two B's. The optimal result we can get with Hu moment invariants is therefore the image in figure 6.3.

In this last test case we will try to remove all the C's from the image in figure 6.3. The results of this test can be seen in figure 6.3. Both Hu moment invariants and Krawtchouk moment invariants did not seem to filter all the C's present in the input image. When the maximum error allowed for the filter was set too small, than the rotated C's were not filtered and when

Figure 6.5: The three reference images that are used to filter the image in figure 6.3. All images are 67 × 72 pixels

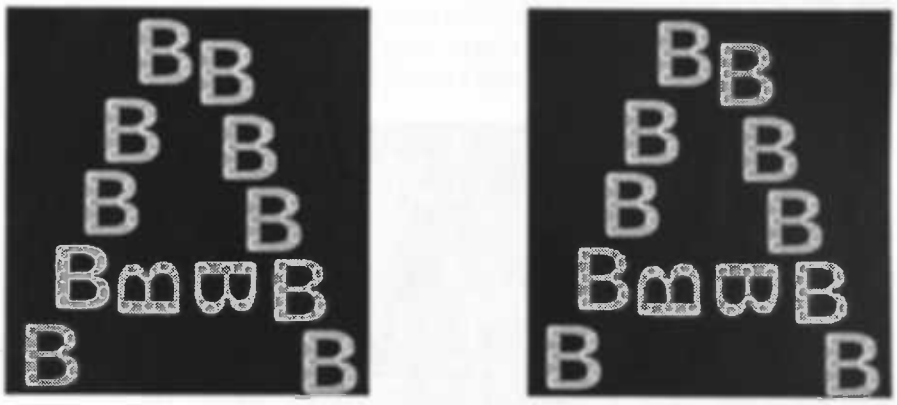


Figure 6.6: Results after filtering with the first reference image in figure 6.3. The left image is the result after using Hu moment invariants as a shape descriptor and the right image is the result after using Krawtchouk moment invariants as a shape descriptor. We can see that in both cases the A is correctly filtered.

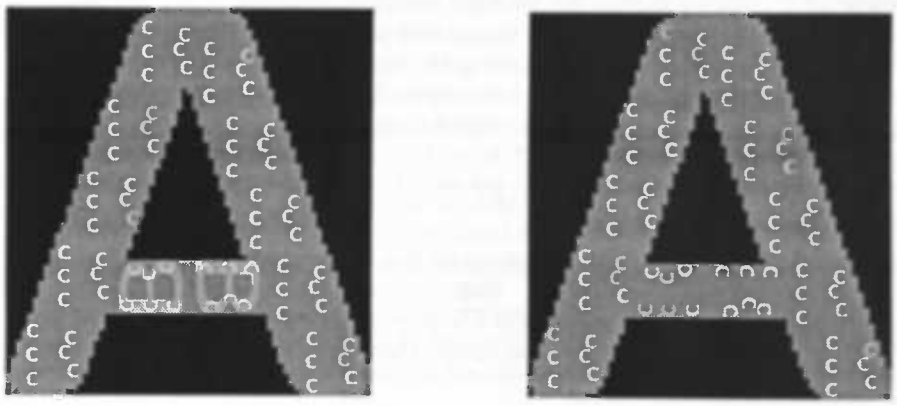


Figure 6.7: Results after filtering with the second reference image in figure 6.3. The left image is the output image when using Hu moment invariants as a shape descriptor and the right image is the output image when using Krawtchouk moment invariants as a shape descriptor. We can see that in the left image, not all the B's are filtered, where in the right image no B's are left.

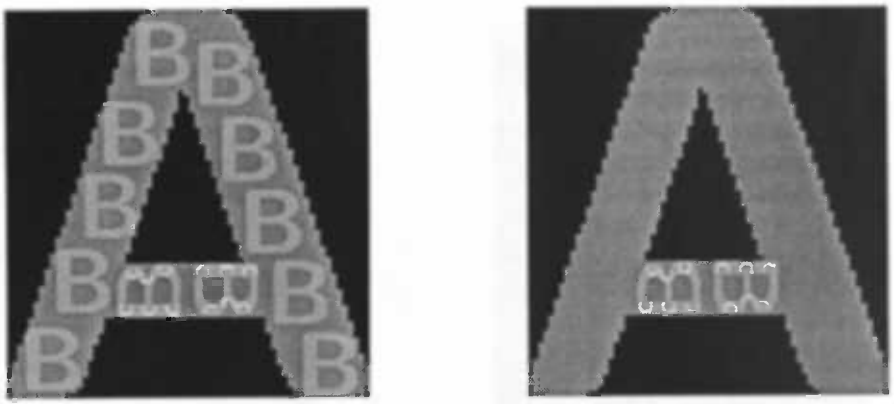


Figure 6.8: Results after filtering with the third reference image in figure 6.3. Both images are can be obtained by using Hu moment invariants as well as Krawtchouk moment invariants as the shape descriptor in the filter. When the maximum error allowed is set too low, the first image is obtained where there are still some C's left. When the maximum error allowed is set too high, the right image is obtained, where the B's have disappeared and where there are still C's left.

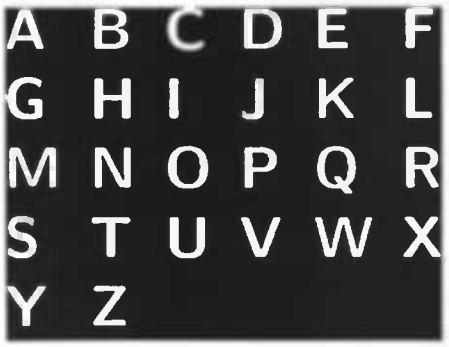


Figure 6.9: An input image with every letter from the alphabet.

the maximum error allowed was slightly increased, the B's would be filtered rather than the C's. This could mean that there is a problem with the rotation invariance of the C's, or that the shape of a C is difficult to determine. Please keep in mind that for all tests only second order and third order moments were used. There is a relatively large chance that when using a higher order of moments, the filter would remove all the C's from the input image. However, it is not within the scope of the research project to try that. The most important thing is that we can make the observation that for recognition of the C's in the image in figure 6.3 a higher order of moments is needed.

6.4 Test case: Alphabet

In this final test case we will take a look at how moment invariants define similarity of shapes. For this test we will use the input image from figure 6.4 and the middle reference image from figure 6.3. We will set-up the shape filter in a way that it removes about five or six letters from the input image. In this way we will get a feeling about how the algorithm determines whether shapes are similar. The results of this test can be seen in figure 6.4

From these results we can see that when Hu moment invariants are used, the letters A, B, K, R and X are removed from the image. Since the letter B was used as a reference image we are not

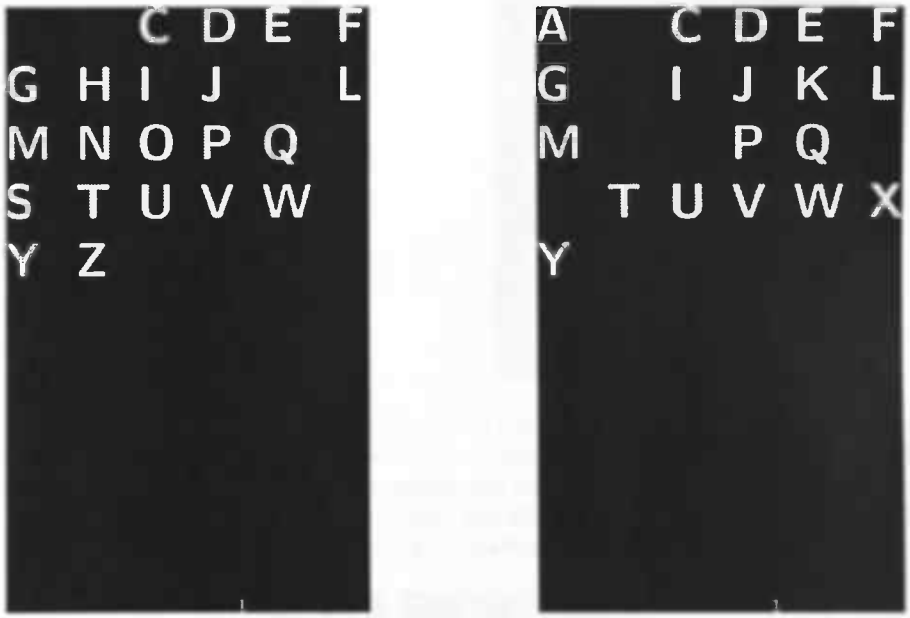


Figure 6.10: Results after filtering the image of figure 6.4 with the middle reference image of figure 6.3. The maximum error was set in a way that when using both shape descriptors, a number of other letters would be filtered also. In the left image Hu moment invariants were used as a shape descriptor and in the right image Krawtchouk moment invariants were used.

surprised that this letter has been removed from the input image. For the human eye, the letter R is very similar to the letter B, because one of the two roundish shapes of the B is replaces by a straight line. This could explain why the R is deleted from the image.

When this comparison is taken one step further, the letter K is nothing more than a letter B where both roundish shapes of the B are replaced by straight lines. The letter X however does not look like a B at all, so we have to look at common features between the letter B and the letter X. When we analyze the letter B, we can observe that it is a very symmetrical letter, that is, the foreground pixels around the center of the B are very evenly distributed. This is also the case with the letters X and K.

When using Krawtchouk moment invariants we observe that the letters B, H, N, O, R and S are removed from the image. Every letter that is removed here has very similar symmetry properties to those of the letter B, so we are not very much surprised with this result.

We can now make the conclusion that both Hu moment invariants as well as Krawtchouk moment invariants describe the distribution of foreground pixels around the center of the image. This is hardly surprising when we look back to the mathematical definitions of both the moment invariants.

Chapter 7

Conclusions

In most tests, reasonable results were obtained when using Hu moment invariants as well as Krawtchouk moment invariants. Second and third order moment invariants seem to define a shape vector that is unique enough to distinguish between the reference shape and other components in the input image. However, the claims of Yap *et al.* [10] and their results could not be reproduced.

Krawtchouk moment invariants have the disturbing property that they are dependent of the size of the image used. A good comparison can only be made if we use the same value for the image size of the reference image as the image size of the input image. Consider the example where moments calculation is done as a pre-processing step, so that the shape vector of an image will be stored with the image. When the shape vector will later be tested against various different-sized reference images, the stored moment invariants cannot be used, and have to be re-calculated (or the shape vectors of the reference image have to be re-calculated). Hu moment invariants do not have this problem.

Another problem with this dependency on the image-size is that setting an error threshold becomes very hard, since the moment invariants change when using another normalization factor and therefore the error also changes. Once again, this is really not what we want.

There are still some problems with the rotation invariance of the moments. This is caused by a misalignment of the two images. Two images are aligned by aligning their longest axes. There is a possibility that those axis are aligned, but have a 180 degrees rotation with respect to each other. When this occurs a large error will be the result, when comparing the two images.

Moment invariants describe the shape of an object by looking at how the foreground pixels are distributed from the center of the image. When the letter B is taken as a reference image, all letters where the foreground pixels are distributed equally around the center are considered to be similar-shaped letters. Krawtchouk moment invariants seem to be based on this property more strongly than Hu moment invariants.

With Hu moment invariants, better results were obtained when testing for rotation invariance. When using Krawtchouk moment invariants, the error between the reference image and the rotated components in the input image became larger than the error between the reference image and some different-shaped components in the input image. This in contrast to the claims of Yap *et al.* [10].

The *Max-Tree* algorithm is very suitable to work with vector attributes such as moment invariants. The complexity of the connected filter is linear in the number of pixels which is a good result.

Chapter 8

Future work

There are still a number of optimizations that can be applied to the *Max-Tree* algorithm. For example, the calculation of Krawtchouk moment invariants or Hu moment invariants can be done with only one step, when the geometric moments of an image are given. This involves some difficult mathematics but will speed up computational time.

Other (shape) filters can be used with the vector attribute *Max-Tree* algorithm. Now that a standard *Max-Tree* algorithm has been expanded to support vector attributes, new shape descriptors can be tested with the algorithm.

An extension from $2D$ to $3D$ is also be possible. The use of Krawtchouk moment invariants can be extended to $3D$ very easily and intuitively. In this way the shape of a volume in $3D$ can be described by a set of moments.

The nearest neighbor algorithm still has to be tested and can be, for example, extended to a neural network classifier.

Bibliography

[1] E. J. Breen and R. Jones. Attribute openings, thinnings and granulometries. *Comp. Vis. Image Understand.*, 64(3):377–389, 1996.

[2] R.C. Gonzales and R.E.Woods. *Digital Image processing*. Prentice Hall, second edtion, 2002.

[3] M. K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, IT-8:179–187, 1962.

[4] X. Y. Jiang and H. Bunke. Simple and fast computation of moments. *Pattern Recognition*, 24:801–806, 1991.

[5] R. Koekoek and R.F. Swarttouw. The Askey-scheme of hypergeometric orthogonal polynomials and its q-analogue. Technical Report 98-17, Faculty of Technical Mathematics and Informatics, Deft University of Technology, 1998.

[6] A. Meijster and M. H. F. Wilkinson. A comparison of algorithms for connected set openings and closings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):484–494, 2002.

[7] P. Salembier, A. Oliveras, and L. Garrido. Anti-extensive connected operators for image and sequence processing. *IEEE Trans. Image Proc.*, 7:555–570, 1998.

[8] E. R. Urbach and M. H. F. Wilkinson. Shape-only granulometries and grey-scale shape filters. In *Proc. Int. Symp. Math. Morphology (ISMM) 2002*, pages 305–314, 2002.

[9] M. H. F. Wilkinson and M. A. Westenberg. Shape preserving filament enhancement filtering. In W. J. Niessen and M. A. Viergever, editors, *Proc. MICCAI'2001*, volume 2208 of *Lecture Notes in Computer Science*, pages 770–777, 2001.

[10] P. T. Yap, R. Paramesran, and S. H. Ong. Image analysis by krawtchouk moments. *IEEE Trans. Image Proc.*, 12:1367–1377, 2003.