Local-Greyscale Vector-Attribute Filters

J.F. de Boer
Department of Mathematics and Computer Science,
Rijks Universiteit Groningen.

Supervisor: M.H.F. Wilkinson
Second reader: S. Achterop

RuG

Abstract

In this paper we will introduce the concept of local greyscale vector attribute filters. Local greyscale vector attribute filters are a variety of vector attribute filters which use greyscale attributes to describe objects, instead of binary attributes used in regular vector attribute filters. An attribute filter is a filter that removes or retains an object based upon that object's properties. A vector attribute filter contains not one attribute but a vector of attributes. A vector attribute filter removes or retains an object based upon the vector attribute (or feature vector) describing that object. We will show how to create local greyscale vector attributes from binary vector attributes and we will test this filter upon a set of images containing traffic signs. The results will be displayed as ROC curves which plots the percentage of detected false positives against the percentage of detected true positives.

Contents

Chapter 1

Introduction

In the modern world everything is digitized, from photographs to paintings to newspapers. This leads to a branch of image analysis, namely shape detection and enhancement. The practical uses of this research include medical applications (e.g. detecting bone structures on X-ray images), commercial applications (e.g. Object Character Recognition software) and government applications (e.g. detecting licence plates on photographs taken by speed cameras). But as the computers become faster and more complex, new paths to research are opened in the image analysis sector. In the past Breen and Jones in [1] describe how a filter can remove or retain objects based upon object attributes. Take for example the attribute area. An attribute filter can remove or retain objects based upon the area attribute, for instance remove all those connected components with an area smaller than a certain value. This is more thoroughly discussed in [2, 19, 18]. Urbach et al. used this concept in [16] to introduce vector-attribute filters. Vector-attribute filters are attribute filters, which use a vector as an attribute instead of a scalar. These vectors contain multiple attributes of the region they describe. Filtering in vector-attribute filters is done by means of a dissimilarity measure, which describes the difference between two vectors of attributes in a single number. In chapter 2 we will discuss the original attribute filters described by Breen and Jones [1] and Urbach et al. vector-attribute filters [16]. In chapter 2 we will also introduce our expansion on vector attributes, local-greyscale vector-attributes. Also a data structure described by Salembier et al. [12] will be discussed, which was used by Urbach et al.[16] and will again be used in this paper. This should give the reader enough information on attribute filters in general and local greyscale vector attribute filters in particular.In chapter 3 we will discuss moments. Moments are used to describe objects or images using mathematical formulas. We will describe how moment invariants (e.g. moment scale invariants, which are moments that are invariant to scale) can be build from geometric moments.In chapter 4 we will describe design and implementation issues which needed resolving during the creation of the filter.Chapter 5 will descirbe how we will test the filter and show the results in an ROC curve. And finally chapter 6 will give our conclusions and recommendations for future research into local greyscale vector attribute filters.

Chapter 2

Attribute Filters

2.1 Introduction

In this chapter we will give a brief introduction into mathematical morphology and morphological operators. Then we will discuss the concept of attribute filters and vector attribute filters. We will introduce the concept of local greyscale vector attribute filters and local greyscale vector attributes and we will discuss the max-tree data structure. which is used to process the images.

2.2 Mathematical Morphology

Using the same introductory structure as used by Sonka et al.[14] we will introduce the concept of mathematical morhpology.

Mathematical morphology, which made its entrance in the field of image analysis during the late 1960s, is a separate part of the image analysis field. Mathematical morphology is based upon non-linear operations operating on object shape. In many ways it yields better results than the linear algebra system of convolution. Mathematical morphology can perform many different tasks, for instance pre-processing, segmentation using object shape and object quantification, which it does better and faster than the standard approaches.

The first people to work in this field were Matheron [8] and Serra [13], whose work was very mathematical in essence. More recent books for example by Heijmans [4] are written in the same spirit. More reference works for mathematical morphology are written by Maragos and Schafer [7] and Roerdink and Heijmans [11].

Mathematical morphology uses point sets, their connectivity and shape, and tools based upon non-linear algebra to simplify images and to preserve and quantify the main shape characteristics of objects. Mathematical morphology works on the initial assumption that real images can be modeled using point sets of any dimension (e.g. N-dimensional Euclidian space), in which the cases of $N = 2$ and $N = 3$ are of particular interest as they represent the images and volumes respectively. The system of subsets inherent to the 2D Euclidian space is a natural domain for planar shape description. The understanding of inclusion ($\subset, \supset$), intersection ($\cap$), union ($\cup$), the empty set $\emptyset$ and set complement (^{c}) is assumed. The set difference is defined as

$$X \backslash Y = X \cap Y^c \tag{2.1}$$

In computer vision real images are discretisized so binary images are represented by a subset of $\mathbb{Z}^2$ and greyscale images by a subset of $\mathbb{Z}^3$. A point set X of an binary image contains the coordinates of pixels that have a value other than zero.

Definition 1 *A morphological transformation Ψ is given by the relation of the image (point set X) with another small point set B called a structuring element. B is expressed with respect to a local origin O (called the representative point).*

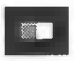

Figure 2.1: Example structuring element. The grey pixel is the representative point

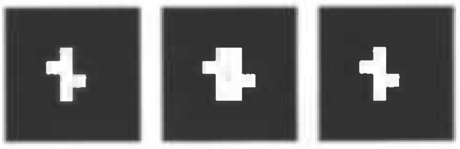

Figure 2.2: Original image (left), dilation (middle) and opening (right)

However this is just one class of many in the field of mathematical morphology. To apply the morphological transformation $\Psi(X)$ to the image X means that the structuring element B is moved systematically across the entire image. Assume that the B is positioned somewhere in the image. The pixel corresponding to the representative point O of the structuring element is called the current pixel. The result of the relation (which can be either zero or one) between the image X and the structuring element B in the current position is stored in the output image in the current image pixel position.

Two of the most commonly used morphological transformations are the Dilation and Erosion.

Definition 2 *Dilation $X \oplus B$ of binary image X by structuring element B is defined as*

$$X \oplus B = \{p \in \varepsilon^2 : p = x + b, x \in X \wedge b \in B\} \tag{2.2}$$

Definition 3 *Erosion $X \ominus B$ of binary image X by structurinf element B is defined as*

$$X \ominus B = \{p \in \varepsilon^2 : p + b \in X, \forall b \in B\} \tag{2.3}$$

Dilation states that if the representative point is a pixel with a value other than zero, then all pixels covered by the structuring element's pixels will be set to one. Erosion states that if all the pixels of the structuring element cover pixels with value one in the image, the pixel corresponding to the origin O of the structuring element will be set to one.

Erosion and dilation are used to create morphological transformations called opening and closing.

Definition 4 *An opening is define as*

$$X \circ B = (X \ominus B) \oplus B \tag{2.4}$$

Definition 5 *A closing is defined as*

$$X \bullet B = (X \oplus B) \ominus B \tag{2.5}$$

The opening is a dilation followed by an erosion. If the image is unaltered after performing an opening the image X is called open with respect to B. The same goes for the closing. If an image X is unaltered after a closing that image is called closed with respect to B.

Example 1 *Figures 2.2 and 2.3 show the opening and closing respectively. For these operations the structuring element in figure 2.1 was used. As can be seen in figure 2.2 the dilation operation thickens the object in the x direction and the erosion removes the thickening again. As the original image and the opening are the same the original image is open with respect to the structuring element in figure 2.1.*
The image however is not closed with respect to the structuring element in figure 2.1. As can be seen in figure 2.3 the erosion removes pixels that are not returned by the subsequent dilation.

Now that the basics of mathematical morphology have been discussed it is time to discuss another morphological transformation.

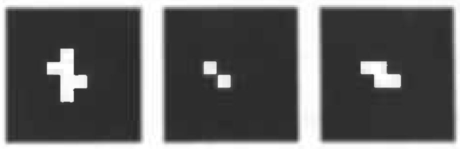

Figure 2.3: Original image (left), erosion (middle) and closing (right)

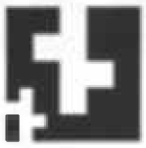

Figure 2.4: Two connected sets

2.3 Attribute Filter Theory

Attribute filters as described in [1] use a criterion to remove or retain connected components (or flat-zones for the greyscale variety) based on their attributes. The concept of trivial thinnings is used. Using a non-increasing criterion it is decided whether a connected component is retained or removed.

Definition 6 *A connected component or connected set is a set of pixels which are connected to each other and all have the same value. Connectivity can be defined by using for example 4-,6- or 8-connectivity as used in image processing*

Definition 7 *A Flat-zone is a set of connected components of the same greyscale value*

Definition 8 *A Peak Component is the set of flat zones which connect to each other and all have a greyscale value between two arbitrary values.*

Definition 9 *The trivial thinning Φ_T of a connected set C with criterion T is just the set C if C satisfies T and empty otherwise. Furthermore $\Phi_T(\emptyset) = \emptyset$.*

Example 2 *Look at figure 2.4. There are two connected sets within this image. One with size* 6 *and the other with size* 24. *We will use the area as the attribute of the connected sets and as a criterion we use 'Area must be bigger than* 10*'. Now if we use the trivial thinnging definition 9 we see that the connected set with area 6 is removed as it does not statisfy our criterion. If the criterion were 'Area bigger than* 5*' both connected sets would have been retained.*

Definition 10 *A criterion T is increasing if the fact that C satisfies T implies that D satisfies T for all $D \supset C$.*

This means that if the set C satisfies T, then all sets D which contain C must also satisfy T. The criterion has to be non-increasing because we want every connected component to be measured on its own merit and do not want to decide removal because its neighbour did not satisfy a criterion. As discussed by Urbach and Wilkinson in [17] scale invariance is not necessarily increasing. For example look at figure 2.4. You can see a set of two connected sets. If we were to use an increasing criterion to remove one of the two connected components, that would automatically mean that we remove the entire set as the connected set is a subset of the set containing the two connected sets.

The binary connected opening $\Gamma_x(X)$ of set X at point $x \in M$ returns the connected component of X which contains x if $x \in X$ and $\emptyset$ otherwise. Or in other words, Γ_x retains the connected component to which x belongs and removes all others.

Definition 11 *The binary attribute thinning Φ^T of set X with criterion T is given by*

$$\Phi^T(X) = \bigcup_{x \in X} \Phi_T(\Gamma_x(X)) \tag{2.6}$$

This formula states that every connected component in the set X is evaluated according to the criterion T and those that are accepted are put into the set $\Phi^T(X)$. It can be shown that this is a thinning because it is idempotent and anti-extensive [1]. If T were increasing, that would mean that the entire filter would be increasing, therefore it is necessary that T in non-increasing..

2.4 Vector Attribute Filters

Vector attribute filters as described by Urbach et al. in [16] are a continuation of the attribute filter theory described in [1]. Urbach et al. describe how a single attribute can be replaced by a vector of attributes. Normal attribute filters work with trivial thinnings see definition 9 but Urbach et al. expanded that thinning to a vector-attribute thinning. By using a multi-variate attribute thinning $\Phi^{\{T_i\}}(X)$ with scalar attributes $\{\tau_i\}$ and corresponding criteria $\{T^i\}$ with $1 \leq i \leq N$ they can preserve connected components if they satisfy at least one of the criteria $T_i = \tau_i(C) \geq \tau_i$. This multi-variate attribute thinning is defined as

$$\Phi^{\{T_i\}}(X) = \bigcup_{i=1}^{N} \Phi^{T_i}(X) \tag{2.7}$$

They then argue that the set of scalar attributes can also be seen as one vector-attribute $\vec{\tau} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ in which case the vector-attribute thinning becomes

$$T^{\vec{\tau}}_{\vec{r}} = \exists i : \tau_i(C) \geq r_i | 1 \leq i \leq N \tag{2.8}$$

This vector-attribute thinning will still preserve those connected components that satisfy at least one criterion. Next they introduce a dissimilarity measure d which computes the difference between two n dimensional vectors from the space $\Upsilon \in \mathbb{R}^n$. The dissimilarity measure d must satisfy $d : \Upsilon \times \Upsilon \to \mathbb{R}$, which means that it translates the difference between the two vectors to a number, or in other words it quatifies the difference between two vectors. Using this dissimilarity measure they define the vector-attribute thinning with respect to a reference vector as

Definition 12 *The vector-attribute thinning $\Phi^{\vec{\tau}}_{\vec{r},\epsilon}$ of X with respect to a reference vector $\vec{r}$ and using vector-attribute $\vec{\tau}$ and scalar value ϵ is given by*

$$\Phi^{\vec{\tau}}_{\vec{r},\epsilon} = \{x \in X | T^{\vec{\tau}}_{\vec{r},\epsilon}(\Gamma_x(X))\} \tag{2.9}$$

This vector-attribute thinning will preserve all the connected components that have a dissimilarity value smaller than the scalar ϵ. In essence They have translated the difference between two vectors to one scalar, which can be used as a criterion in conjunction with a reference vector. The simplest dissimilarity measure mentioned is the Euclidian distance $d(\vec{u}, \vec{v}) = \|\vec{v} - \vec{u}\|$ and this measure will be used in this paper.

Using this type of filter is particularly useful for shape-based filtering as a shape has to be described by attributes and the more attributes the better a shape can be described.

2.5 Local Greyscale Vector Attribute Filters

In this paper we will expand on the vector-attribute filters as described by Urbach et al. in [16]. Urbach et al. use vector-attributes to store attributes of a connected set. We will use vector-attributes to store attributes of peak components. So in essence the attribute describes the shape of the combined connected sets in the peak component. The principle of the vector-attribute filters remains the same, we will still use a criterion to determine retention or deletion and we still will use the dissimilarity measure used by Urbach et al. in [16] to determine the difference between two vectors. The vector-attributes which will be combined to form the new attribute vector are the attributes of connected sets which have a greyscale level around that of the connected set with the lowest greyscale value in the peak component. When the filter is used it is specified how big the difference in greyscale level can be with respect to the combination of attribute vectors to form a new attribute vector. By combining attribute vectors into a single new attribute vector, that new attribute vector has in fact become an attribute vector that describes multiple connected sets and their difference in greyscale level and thus becomes a local-greyscale vector-attribute. The local part means that the connected set to which the attribute vector is bound is the equivalent of greyscale level 0. All the other attribute vectors combined into that attribute vector are scaled according to that greyscale level. In the next section we will discuss a data structure to store connected sets and which is ideal for using attribute filters.

2.6 Max-tree

The Max-tree data structure as described in [12] is a tree data structure in which nodes with the same value are stored at the same level. When using images the value of a node is the greyscale value of the connected component (or flat zone for the greyscale variety). This means that all connected components (or flatzones) of the same greyscale value will be stored in the tree at the same level. As attribute filters remove or retain connected components (or flatzones) this means that this data structure is ideal for such filters as it stores connected components (or flatzones) based upon their greyscale value.

To create a Max-tree it is necessary to find the pixel with the lowest greyscale value. All connected components which satisfy 'Greyscale value equals lowest value' will be combined to found the rootnode. As can be seen the root node can contain multiple connected components and represents the background of the image. Now the children of the rootnode are defined as the connected components whose greyscale value is strictly higher than the greyscale value of the rootnode. Increase the current greyscale value by one and repeat the following steps on each of the children.

- Determine the connected components with greyscale value equal to the current greyscale value. These connected components will remain in this node.
- Determine the connected components with greyscale value strictly greater than the current greyscale value. These will become the child nodes to the current node.
- Increase the current greyscale value by one and repeat these steps with the child nodes as input. Or if the greyscale value equals 255 stop.

2.7 Max-tree construction example

Consider the image as shown in figure 2.5. It contains seven connected components labeled from A to G. The number next to the label is the greyscale value. As can be seen, this image has 3 greyscale values, 0,1 and 2.

First we need to find the connected component with the lowest greyscale value. In this case A0. This connected component will be the root for our Max-tree.

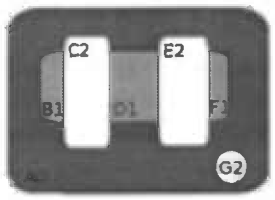

Figure 2.5: Original image

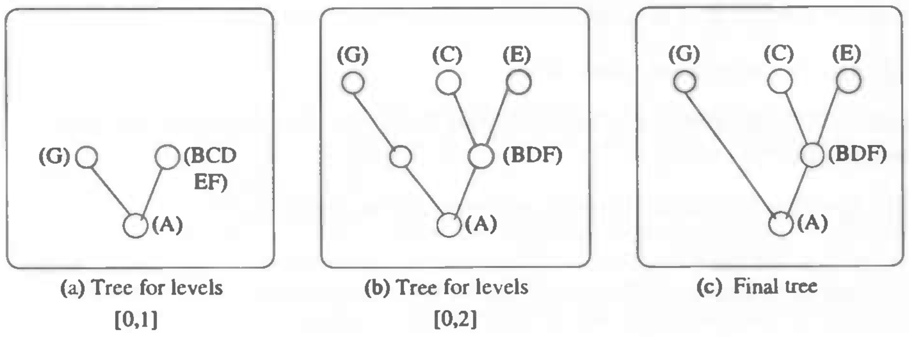

Figure 2.6: Construction of the Max-tree. Example after Salembier et al. [12]

Next the connected components with a greyscale value strictly higher than 0 are determined. As can be seen there will be two connected components strictly higher than 0, BCDEF and G. These will be the children of the rootnode. The result can be seen in 2.6(a).

Now the greyscale value is increased by one to 1 and the same algorithm is applied to the children. Thus for both the connected components strictly higher than 1 are determined. In the case of G it is the entire connected component G. In the case of BCDEF the result will be two connected components, C and E. All those connected components will become the children of the current node, resulting in the tree as shown in figure 2.6(b).

The final step is to remove dummy nodes, the nodes which do not have a connected component tied to them. In this case there is only one. The final tree is shown in figure 2.6(c).

Algorithm 1 shows a flood algorithm which will create a max-tree. A flood algorithm is an algorithm that starts at a single point or instance and then will expands by going into a recursion step for every other point or instance not previously visited.
Explanation of the functions and variables used in algorithm 1

- hqueue-add(h,p) Add the pixel p (of greylevel h) in the queue of priority h
- hqueue-first(h) Extract the first available pixel of queue of priority h
- hqueue-empty(h) Return 'TRUE' if queue of priority h is empty
- number-nodes(h) This defines the number of nodes C_h^k at level h. Initial value is 0

- ORI(p) This denotes the original grey level value of pixel p
- STATUS(p) This stores the information of the pixel status: the pixel can be
  - 'Not-analyzed'
  - 'In-the-queue'
  - assigned to node C_h^k in which case STATUS(p) $== k$

Algorithm 1 works as follows:

- Get a pixel p from the hqueue.(line 4 in algorithm 1)
- Check every neighbour q of p check if it has been analysed. (line 7)
- If not add it to the hqueue. (lines 8-9)
- If q is not analysed check if its value is higher than p's. If so run the flood algorithm on level q. (lines 11-16)
- After all flooding is done the parent pointers are set.(lines 22-33)

As can be seen in the pseudo code the dummy nodes mentioned in the theory are automatically skipped in the actual algorithm. For simplicity attribute management is not shown in algorithm 1.

2.8 Local Greyscale Attributes

The max-tree contains connected sets of specific greyscale values. The attributes stored in the max-nodes are binary attributes, which only apply to the connected set stored in the max-node. To create a local greyscale attribute it is necessary to combine the binary attributes of max-nodes with consecutive greyscale values and are parent and child. Unfortunately not all max-nodes that are parent and child have a greyscale value difference of 1, therefore it is necessary to introduce layernodes. These layernodes will form a chain between a parent and child max-node with a greyscale value difference greater than 1. Layernodes will have the same vector attribute as the max-node with the highest greyscale value (the child max-node). Once the layernodes have been added the vector attributes of parent and child nodes can be combined to form greyscale vector attributes. Essentially we re-invent the dummy nodes.

By combining the vector attributes of a parent and child max-node a local greyscale vector attribute is created in the parent max-node. This local greyscale vector attribute represents the shape of the parent node combined with the shape of the child nodes up to a predetermined level h. This level h is the maximum difference between the greyscale value of the parent max-node and the children of that parent which will be used in the creation of the new max-node.

2.9 Max-tree filtering

During the creation of the Max-tree it is possible to store more than just the connected component in a node. Relevant data of the connected component can also be stored in the node. This data is called an attribute of the connected component. The attributes are used in attribute filters to determine whether a node should be retained or removed. There are multiple node removal schemes, some of these are discussed in [12, 17]. These schemes are

Min A node C_h^k is removed if $T(C_h^k)$ is false, or if one of its ancestors is removed.

Max A node C_h^k is removed if $T(C_h^k)$ is false and all its descendants are removed as well.

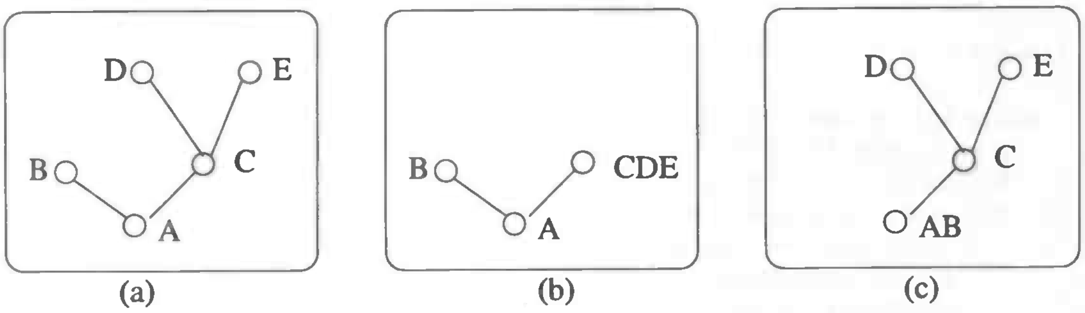

Figure 2.7: Example of the Direct filter rule. (a) Original max-tree. (b) Nodes D and E removed. (c) Node B removed

Viterbi The removal or retainal of a node is considered to be an optimization problem. For each leaf in the tree the path with the lowest cost to the root is taken, where cost is assigned to each transition. For details see [12].

Direct A node C_h^k is removed if $T(C_h^k)$ is false; its pixels are lowered in greylevel to the highest ancestor which meets T, its descendants are unaffected.

Substract As above, but the descendants are lowered by rhe same amount as C_h^k itself.

Using a criterion and a removal scheme one traverses the Max-tree and per node determines whether it should be retained or removed. Once every node has been analysed the resulting Max-tree represents the filtered image. Now one only has to recreate the image, which is nothing more than writing back the connected components with their new greyscale value.

For example take a look at figure 2.7(a). This shows a max-tree with 5 nodes. Node A is the root node and nodes B through E are normal max-nodes. Figure 2.7(b) shows the max-tree after it has been filtered with the direct filter rule. Nodes D and E did not meet the criterion, nodes B and C did however. Thus according to the direct filter rule the nodes D and E are lowered in greylevel to the highest ancestor which did meet the criterion. In this case node C. Thus node C now contains the connected components of nodes C,D and E. Another example in figure 2.7(c) we see the max-tree after another filter run. Here node B was the only node that did not meet the criterion. As per the direct filter rule node B's greylevel was lowered to the highest ancestor, in this case node A, the root node. So node A now contains the nodes A and B, but as node A was the root node, node B is now part of the background.

Algorithm 2 shows the algorithm for the direct filter rule. The following variables are used:

- NUMLEVELS, this is a constant which equals the number of greyscale levels in an 8 bit greyscale image, this is 256.
- NumNodesAtLevel, this is an array which contains the number of max-nodes at a certain greylevel.
- nodes, this is a 2 dimensional array containing the max-tree. The first index is the greyscale value of the nodes and the second index is the n-th node at that greyscale level.
- criterion, this is a procedure with a max-node as input. It will return `true` if the criterion is met and `false` if not.

As can be seen in this algorithm every node is evaluated and its greyscale value changed accordingly. The node however is not moved from its place in the array. And because we work from the root upward in the greylevels the parent of any node being evaluated has already been evaluated, thus if a node's parent was rejected and the current node is rejected, the current node's greylevel will be adjusted to the parent's adjusted greyscale value.

```
flood(h)

  while not hqueue->empty(h)
    p := hqueue-first(h)
    STATUS(p) := number-nodes(h)
    for every neighbour q of p
      if STATUS(q) == "Not-analyzed"
        hqueue-add(ORI(q),q)
        STATUS(q) := "In-the-queue"
        node-at-level(ORI(p)) := true
        if (ORI(q) > ORI(p))
          m := ORI(q)
          repeat
            m := flood(m)
          until m == h
        end
      end
    end
  end
  number-nodes(h) := number-nodes(h) + 1

  m := h - 1
  while m >= 0 and node-at-level(m) == false
    m := m - 1
  end

  if m >= 0
    i := number-nodes(h) - 1
    j := number-nodes(m)
    father of C^i_h := C^i_m
  else
    C^i_h has no father (C^i_h is root node)
  end

  node-at-level(h) := false
  return m
end
```

Algorithm 1: Flood algorithm for the creation of a max-tree

```
for i=0;i<NUMLEVELS;i++
  for j=0;j<NumNodesAtLevel[i];j++
    node = nodes[i][j];
    parent = node->parent;
    if (node != parent AND criterion(node))
      node->NewLevel = node->Level;
    else
      node->NewLevel = parent->NewLevel;
    end
  end
end
```

Algorithm 2: Pseudo code for the direct filter rule

Chapter 3

Moments

3.1 Introduction

In this chapter we will describe what moments are and how to create them. We will also show how to create moment invariants and finally discuss Hu's moment invariants [5]. Hu's moment invariants will be contained within the vector attributes described in the previous chapter.

3.2 Introduction to moments

An image can be interpreted as a 2D function $f(x, y)$, where x and y are the coordinates of the pixel within the image. This function returns a certain value ($[0, 1]$ for binary images, $[0, 255]$ for greyscale images), which is the value of pixel (x, y). There are other ways to represent an image, one of those is the frequency domain, which is the result of a Fourier Transform. The frequency domain represents the image as a 1D function $g(\omega)$, in which ω represents a frequency ($\{0, 1\}$ for binary images and $\{0, \ldots, 255\}$ for greyscale images) and returns the intensity of that specific frequency in the image. In this section yet another representation will be described and why it is of particular use within this thesis.

3.3 Geometric Moments

The most basic of the different moments available are the geometric moments. They are in fact bases for the creation of the other moments. The mathematical formula for a geometric moment in a digitised greyscale image is shown below.

Definition 13 *The geometric moment m_{pq} can be created using the following formula.*

$$m_{pq} = \sum_x \sum_y x^p y^q \cdot f(x, y) \tag{3.1}$$

In which $f(x, y)$ is as described above.

Which basically means that for every pixel you multiply the x and y coordinates (to a certain power, depending on which moment you are looking for) with the greyscale value of the pixel. Thus in the end you get a number which represents the region you summed over. A special moment is m_{00}, which represents the area of the region the moment represents in a binary image. Why are moments so useful? That is because if you have two different objects within an image they will both have their own geometric moment. But the useful part is that if you want to calculate the moment of the two objects combined you can just add the two separate moments together to form the moment describing the two objects. As the moment invariants described in this chapter are

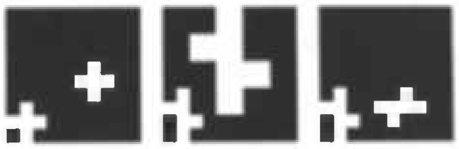

Figure 3.1: Example images for moments

all created from the geometric moments this ability holds true for the moment invariants too.

As the geometric moments form the basis for all other moments we store the geometric moments in our attribute vectors and in the filtering step we compute the desired moments from the geometric moments.

Looking at the formula for geometric moments it is clear that the value of the moment is sensitive to where the region of interest is within an image. If in an image the top left corner is the origin and the bottom right corner (x_{max}, y_{max}) then the further the region of interest is placed near the bottom right corner, the higher the moment will become. This is not very useful when detecting shapes within images as the shapes can be all over the image and the geometric moments of those shapes will vary greatly even if the shapes are the same, yet not in the same place.

Example 3 *In figure 3.1(left) 2 crosses are drawn. One near the origin and the other further to the right. The origin is in the lower left pixel. The cross near the origin consists of the pixels* $\{(0,1),(1,0),(1,1),(1,2),(2,1)\}$ *and the other cross consists of the pixels* $\{(5,4),(6,3),(6,4),(6,5),(7,4)\}$. *If we now calculate the geometric moments for both these crosses we see that the result for the cross near the origin is* $m_{11} = 5$ *and that for the other cross is* $m_{11} = 110$.

Seeing the difference between those two values it is clear that geometric moments are very sensitive to the location of the shape within an image. Seeing this it is clear that some improvement is needed before moments are useful in this paper.

3.4 Centralised Geometric Moments

The first improvement is to make the geometric moments translation invariant, because this allows us to describe objects anywhere in an image. The result of a translation invariant moment will be the same regardless of where the object is located in the image. Translation invariance is obtained by translating the center of mass for every region of interest to the origin before calculatin its moment. Which results in the following moments

Definition 14 *Centralised moments are described by the following mathematical formula*

$$\mu_{pq} = \sum_x \sum_y (x - x_c)^p (y - y_c)^q \cdot f(x,y) \tag{3.2}$$

In which x_c, y_c *are the coordinates for the center of mass for the region of interest and can be calculated using the following equations*

$$x_c = \frac{m_{10}}{m_{00}} \tag{3.3}$$

$$y_c = \frac{m_{01}}{m_{00}} \tag{3.4}$$

By translating the center of mass for the region of interest to the origin it no longer matters where in an image the region is located. Thus a shape somewhere in an image is first translated to the origin and then its geometric moment is computed, which means that wherever the shape is it will result in the same centralised moment if it has the same orientation and size.

Example 4 *Looking once again to figure 3.1(left) we can calculate that for the cross near the origin* $x_c = \frac{5}{5} = 1$ *and* $y_c = \frac{5}{5} = 1$ *and for the other cross* $x_c = \frac{30}{5} = 6$ *and* $y_c = \frac{20}{5} = 4$*. If we now calculate* μ_{11} *for both crosses we get for the cross near the origin* $\mu_{11} = 0$ *and the other cross* $\mu_{11} = 0$*. As can be seen the centralised geometric moment is the same for both crosses, thus the geometric centralised moments are indeed translation invariant.*

Looking at the notation in definition 14 it is clear that two passes have to be made to calculate the centralised moments for a certain image. For efficiency it is possible to calculate the centralised moments of an image in one pass, this is done by rewriting the definition. As an example of how this is done we will rewrite the definition for μ_{20} and μ_{02}.

$$\begin{aligned}
\mu_{20} &= \sum_{\#x}\sum_{\#y}(x - x_c)^2(y - y_c)^0 && (3.5)\\
&= \sum_{\#x}(x - x_c)^2 && (3.6)\\
&= \sum_{\#x}(x^2 - 2xx_c + x_c^2) && (3.7)\\
&= \sum_{\#x}x^2 - \sum_{\#x}2xx_c + \#xx_c^2 && (3.8)\\
&= m_{20} - \frac{2m_{10}^2}{m_{00}} + m_{00}\frac{m_{10}^2}{m_{00}^2} && (3.9)\\
&= m_{20} - \frac{m_{10}^2}{m_{00}} && (3.10)
\end{aligned}$$

For μ_{02} this is exactly the same, only this time $x = y$ and $x_c = y_c$, which will result in

$$\mu_{02} = m_{02} - \frac{m_{01}^2}{m_{00}} \qquad (3.11)$$

As these notations do only contain geometric moments, the centralised moments can be instantly derived using these equations.

If we were to make a shape bigger the centralised moment of that shape will get bigger too, which leads to yet another improvement which is needed.

3.5 Scaled Centralised Moments

If two shapes within an image are the same but unequal in size their centralised moments will never equal eachother. Thus it is necessary to introduce scale-invariance.

Definition 15 *Scale invariance can be found within scaled centralised moments (scale change* $x' = \alpha x, y' = \alpha y$*).*

$$\eta_{pq} = \frac{\mu'_{pq}}{(\mu'_{00})^{\gamma}} \qquad (3.12)$$

$$\gamma = \frac{p+q}{2} + 1 \qquad (3.13)$$

$$\mu'_{pq} = \frac{\mu_{pq}}{\alpha^{(p+q+2)}} \tag{3.14}$$

and normalised centralised moments v_{pq}

$$v_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{3.15}$$

In which it states that an increase in the size of the image does not affect its moment as the normalised centralised moments are still computed by dividing by μ_{00}, which has increased as much as μ_{pq}.

Example 5 *Look at figure 3.1(middle). Two shapes can be seen, one twice as big as the other. The pixels of the small shape are $\{(0,2),(1,0),(1,1),(1,2),(1,3),(2,1)\}$. The pixels of the big shape are $\{(2,6),(2,7),(3,6),(3,7),(4,2),(4,3),(4,4),(4,5),(4,6),(4,7),(4,8),(4,9),(5,2),(5,3),(5,4),(5,5),(5,6),(5,7),(5,8),(5,9),(6,4),(6,5),(7,4),(7,5)\}$. For both shapes the γ for μ_{11} will be the same. $\gamma = \frac{1+1}{2} + 1 = 2$. Now we have to determine the values of μ_{11} for both shapes, to do that we need to determine x_c and y_c. For the smaller shape $x_c = \frac{6}{6} = 1$ and $y_c = \frac{9}{6} = 1\frac{1}{2}$. Now that we have calculated these values we only have to insert them into the formula for Centralised Geometric moments. Which leads to the following value for $\mu_{11} = -1$. With this we can calculate $v_{11} = \frac{-1}{6^2} = \frac{-1}{36}$. For the big shape these values can be calculated $m_{10} = 108$, $m_{01} = 132$, $m_{00} = 24$. Which leads to $x_c = \frac{108}{24} = 4\frac{1}{2}$ and $y_c = \frac{132}{24} = 5\frac{1}{2}$. Now we can calculate $\mu_{11} = -16$, which leads to $v_{11} = \frac{-16}{24^2} = \frac{-16}{576} = \frac{-1}{36}$, which equals the values of v_{11} for the small shape. So we have established that size does not matter in these moments as both the small and big shape give the same result.*

Now that we've negated the increased size, using these normalised centralised moments. We have shown that location and size do not matter anymore, the only thing that still matters is orientation.

3.6 Hu's moment invariants

We cannot assume that a shape will have the same orientation within an image as it has in the reference image. Therefore rotation invariance has to be added to the normalised un-scaled centralised moments. Using these properties Hu introduced his seven moment invariants in [5] in which he showed after lots of complicated calculations that those moment invariants are invariant to scale, rotation and translation which makes them extremely useful for shape detection within images. Hu's moment invariants are listed below.

Definition 16

$$\begin{aligned}
\phi_1 &= v_{20} + v_{02} && (3.16)\\
\phi_2 &= (v_{20} - v_{02})^2 + 4v_{11}^2 && (3.17)\\
\phi_3 &= (v_{30} - 3v_{12})^2 + (3v_{21} - v_{03})^2 && (3.18)\\
\phi_4 &= (v_{30} + v_{12})^2 + (v_{21} + v_{03})^2 && (3.19)\\
\phi_5 &= (v_{30} - 3v_{12})(v_{30} + v_{12})[(v_{30} + v_{12})^2 - 3(v_{21} + v_{03})^2] && (3.20)\\
&\quad +(3v_{21} - v_{03})(v_{21} + v_{03})[3(v_{30} + v_{12})^2 - (v_{21} + v_{03})^2] && (3.21)\\
\phi_6 &= (v_{20} - v_{02})[(v_{30} + v_{12})^2 - (v_{21} + v_{03})^2] + 4v_{11}(v_{30} + v12)(v_{21} + v_{03}) && (3.22)\\
\phi_7 &= (3v_{21} - v_{03})(v_{30} + v_{12})[(v_{30} + v_{12})^2 - 3(v_{21} + v_{03})^2] && (3.23)\\
&\quad -(v_{30} - 3v_{12})(v_{21} + v_{03})[3(v_{30} + v_{12})^2 - (v_{21} + v_{03})^2] && (3.24)
\end{aligned}$$

In which v_{pq} is as described above.

Example 6 *Now take a look at figure 3.1(right) which shows two identical shapes, only one is turned 90 degrees clockwise. The left shape contains the pixels $\{(0,2),(1,0),(1,1),(1,2),(1,3),(2,1)\}$*

	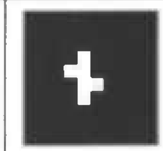	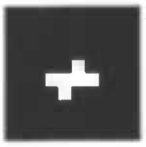	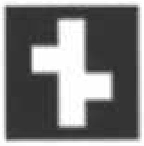
ϕ_1	0.208333	0.208333	0.229167
ϕ_2	0.012539	0.012539	0.012539
ϕ_3	0	0	0
ϕ_4	0	0	0
ϕ_5	0	0	0
ϕ_6	0	0	0
ϕ_7	0	0	0

Figure 3.2: Some binary images and Hu's moment invariants

the shape on the right contains the pixels $\{(4,2),(5,1),(5,2),(6,2),(6,3),(7,4)\}$. x_c *and* y_c *for the left shape have already been given in the previous example. They are* $x_c = \frac{6}{6} = 1$ *and* $y_c = \frac{9}{6} = 1\frac{1}{2}$. *Now to calculate the first of Hu's moment invariants we need to calculate* v_{20} *and* v_{02}. *For both these normalised central moments the* γ *will be* $\gamma = \frac{2}{2} + 1 = 2$. *Which leads to the following values for* v_{20} *and* v_{02}. $v_{20} = \frac{2}{6^2} = \frac{1}{18}$ *and* $v_{02} = \frac{(5\frac{1}{2})}{6^2} = \frac{5\frac{1}{2}}{36} = \frac{\frac{11}{2}}{36} = \frac{11}{72}$ *so the first of Hu's moment invariants for the left shape will be* $\phi_1 = v_{20} + v_{02} = \frac{1}{18} + \frac{11}{72} = \frac{4}{72} + \frac{11}{72} = \frac{15}{72}$.

Now to calculate the same moment invariant for the right shape. First we need to determine x_c *and* y_c. $x_c = \frac{33}{6} = 5\frac{1}{2}$ *and* $y_c = \frac{12}{6} = 2$. *Which leads to* $v_{20} = \frac{5\frac{1}{2}}{6^2} = \frac{\frac{11}{2}}{36} = \frac{11}{72}$ *and* $v_{02} = \frac{2}{6^2} = \frac{2}{36} = \frac{1}{18}$. *Which results in the following moment invariant for this shape* $\phi_1 = v_{20} + v_{02} = \frac{11}{72} + \frac{1}{18} = \frac{11}{72} + \frac{4}{72} = \frac{15}{72}$. *As can be seen the moment invariant* ϕ_1 *for both shapes is the same, therefore this moment invariant is rotation invariant.*

Since Hu's first article much research has been done in moment invariants and other moment invariants were introduced. We will now list a few.

- Krawtchouk moment invariants [21]
- Zernike and Chebychev moment invariants [6, 9]
- Affine and blur moment invariants [3, 15]

In table 3.2 some images and Hu's moment invariants corresponding to those images can be found. The only anomaly I can discern is that for one image ϕ_1 should be the same as the others, yet it is not. Wilkinson and Westenberg describe in [20] how the moment invariants described by Hu work fine for the continuous case, but for the discretisation a correction has to be added to ϕ_1. In this paper we will not elaborate on that, but it explains the discrepancy found in figure 3.2.

Chapter 4

Implementation and Design

4.1 Introduction

In this chapter we will discuss the design and implementation issues that arose whilst creating the filter. First we will discuss how to add layernodes to the data structure used by Urbach [16] as the flood algorithm 1 does not add the dummy nodes described in the max-tree theory in chapter 2. Next we will motivate why we have made the max-tree bidirectional instead of the uni-directionality described in algorithm 1. After that we will discuss our implementation for testing the filter by means of a template. Then we will describe the new parameter `windowsize` which is an intergral part of the filter and finally we will discuss the filtering procedure.

4.2 Layernodes

The vector-attribute filter program as used in [16] has a memory saving datastructure. The max-tree was stored in an array with size equal to the image width times the image height. Or in other words the array had a size equal to the number of pixels in the image. This has a reason as the worst case scenario for the max-tree is that every pixel in an image is a non-dummy node in the max-tree, thus in this case there would be a number of connected components equal to the number of pixels in the image. So there is no possibility that there can be more connected components than there are pixels in the image.

While the max-tree is created the area of the connected components is stored in the max-nodes. Using this area an indexing scheme can be derived for determining where in the array the next max-node should be entered. As the Max-tree is created by increasing the pixel value the number of pixels below a certain level equals the area of all max-nodes with level lower than the current pixel value. This fact is used to determine the offset index for max-nodes at a certain level. So to access max-node C_h^k we first have to determine the offset for level h which is equal to `NumPixelsBelowLevel[h]` and next we have to determine the k-th max-node, which is just adding k to the offset.

In figure 4.1 is an example max-tree. It contains 4 max-nodes (A, B, C, D). max-node A is

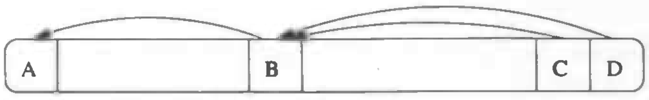

Figure 4.1: Example of the datastructure

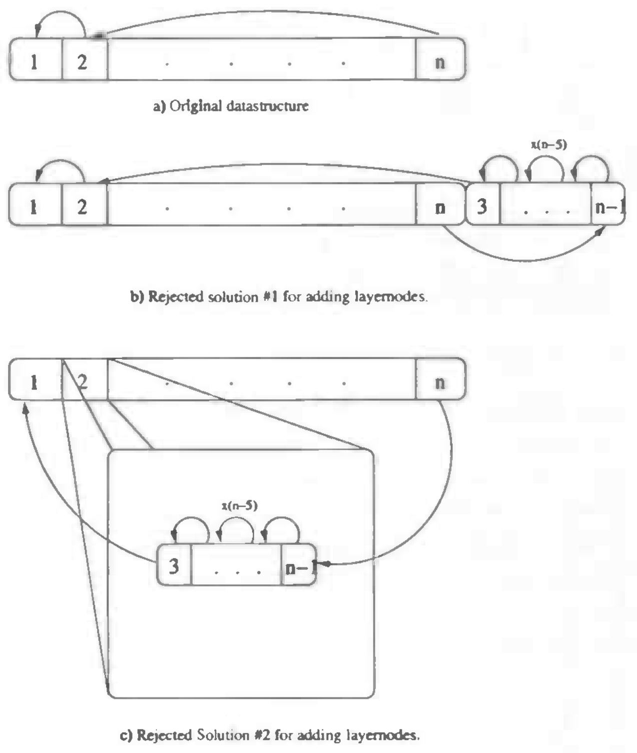
1
2
n
a) Original datastructure
x(n-5)
3
n-1
b) Rejected solution #1 for adding layernodes.
c) Rejected Solution #2 for adding layernodes.

Figure 4.2: Rejected solutions to the layernode addition problem

the rootnode of the Max-tree, it has 1 child B. max-node B has two children (C, D). To access max-node B we first have to determine the offset for level B, which is $Area(A)$ and as max-node B is the only max-node of its level the offset already points to it.
If we want to acces max-node D, we first have to determine the offset for level D, which is equal to level C. Which equals $Area(A) + Area(B)$. As max-node D is the second node of its level we need to add 1 to the offset to point to the correct max-node.

For the filter described in this paper it is necessary to add so-called 'layernodes' to the max-tree. These layernodes are the dummy nodes, which were mentioned in [12], but were not present in the flood algorithm described in chapter 2. By creating layernodes we fill up the difference in greyscale value between a parent and child node. Each layernode represents a greyscale level between the parent and child node.
Look at figure 4.3. The left figure shows a cross-setion of figure 2.5 and it can be seen that node G has a greyscale value 2, whereas its parent has greyscale value 0, which means that the difference is greater than 1 so a layernode has to be added. In the right figure the added layernode X can be seen. By adding layernode X the greylevel difference between parents and children is reduced to one.

When the layernodes are added it is easy to combine the vector attributes of the current node with those of n greyscale levels higher. For example if we pick $n = 1$ and figure 4.3(right) the result of combining will be that node A will contain the combined vector attributes of node A,X and BDF. Node BDF will contain the combined vector attributes of node BDF,C and E. Node X will contain the combined vector attributes of node X and node G. Nodes C,E and G will remain unaltered as they have no children. After this combining of vector attributes the current max-node

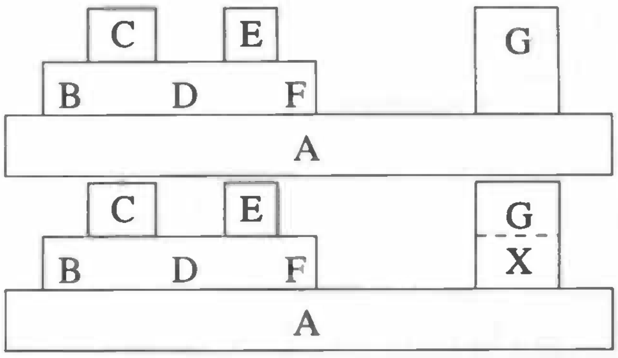

Figure 4.3: Cross-section of figure 2.5 without layernode (top) and with layernode (bottom)

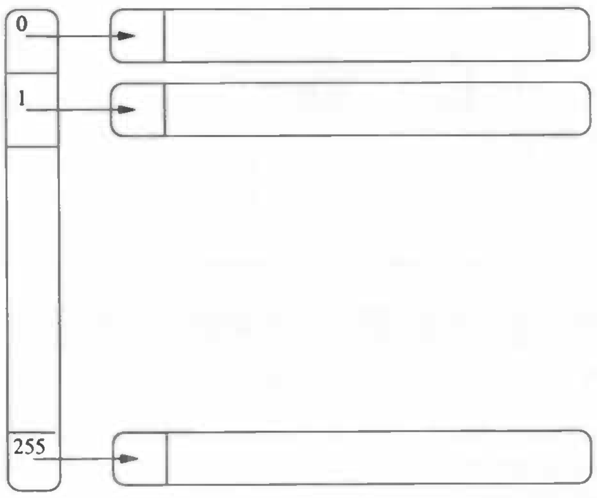

Figure 4.4: Overhauled data structure

contains a local greyscale vector attribute of level n.

To add the layernodes to the existing data structure several possibilities were reviewed.The first possibility reviewed was extending the array containing the max-tree by pasting the layernodes behind the existing max-tree. This was to be done on a max-node by max-node basis. As can be seen in figure 4.2(b) this leads to chaos as the layernodes are not directly accessable as they are not part of the indexing scheme.

The second possibility reviewed was to create a list of max-nodes inside the current max-nodes which would represent the layernodes between the max-node and its parent. This is shown in figure 4.2(c). This is still not a viable option as the layernodes are still not directly accessible.

Therefore we opted for a complete overhaul of the data structure. Instead of a one-dimensional array it was decided to use a two-dimensional array in which the first index corresponds to the level of the max-node and the second index to the node number at that level. This datastructure is less memory efficient, but is more flexible to additions. This data structure is shown in figure 4.4. Using the data structure as shown in figure 4.4 all max-nodes and layernodes are directly accessible.

4.3 Bidirectionality

Another change is making the Max-tree a bi-directional tree, instead of the former uni-directional tree. This means that in the original program the max-nodes only knew their parents, not their children. In our program we added pointers to the children as well, thus making searching or traversing the tree much easier. This is necessary as adding a number of nodes together is best done by traversing the tree with a recursive algorithm starting at the root.

4.4 Windowsize

A new parameter was added to the program. This parameter called `windowsize` reprents the number of layers to combine to form a new max-node. A windowsize of 0 means that no layers will be combined and thus results in the original vector attribute program by Urbach et al. This parameter also determines the scalar with which the binary reference attribute vector is multiplied to create a greyscale attribute vector of the same local greyscale as the local greyscale vector attributes in the max-nodes. The scalar used for this multiplication is `windowsize+1`, this as `windowsize`=0 means that no max-nodes will be combined, therefore the reference attribute vector should be multiplied with 1.

4.5 Filter

The filtering procedure used in [16] uses the direct filter rule described in Chapter 2 section 8. As we now have to consider multiple nodes while filtering, the filter algorithm has been slightly altered.

```
for i=0;i<NUMLEVELS;i++
  for j=0;j<NumNodesAtLevel[i];j++
    node = nodes[i][j];
    parent = node->parent;
    if (node != parent AND criterion(node))
      node->NewLevel = node->Level;
      SetChildren(node,windowsize);
    else
      node->NewLevel = parent->NewLevel;
    end
  end
end
```
Algorithm 3: Pseudocode for the direct filter rule with local greyscale attributes

Algorithm 3 describes the direct filter rule for local greyscale attributes. The variables and procedures used are the same as in algorithm 2 except for

- SetChildren, this is a procedure with as parameters a max-node and windowsize. It will set the greyscale levels for all children of the max-node uptil a depth of windowsize.

What this altered filtering procedure does is check every node against the criterion and if the node matches the criterion, it will set all greyscale values of that node and its children upto depth windowsize. This is because the node being examined contains the attributes of its children upto depth windowsize, thus all nodes contributing to the retention should also be retained. Because all the nodes that contributed to a retention are retained as well the filter becomes a filter in mathematical morphological sense as this implies idempotence.

Chapter 5

Experiments and Results

5.1 Introduction

In this chapter we will discuss the data set used to run our experiments on, how the experiments are set up and how we will validate the results. The results themselves are also be part of this chapter.

5.2 Data set

The filter will be tested on a data set comprising of real-life images of traffic signs. The specific traffic signs are shown in figure 5.1 on the top. The bottom consists of the reference shape on which the filter will try to detect the traffic sign in an image. As can be seen this reference shape consists of only one connected component, and is therefore suitable as a reference shape.

The images for the test-set were acquired by walking through the city of Groningen and the town of Drachten, both in The Netherlands, with a Sony Cybershot DSC-S50 2.1 Megapixel digital camera.

An example of an input image is shown in figure 5.2 together with the template for determining 'True Positive' nodes.

The reference images were acquired from the following site: `http://proto.thinkquest.nl/~klb019/borden.htm` which shows the Dutch traffic signs by category and which are synthetic, thus devoid of any noise. As these images are noiseless they are excellent to construct reference images, just by using an imaging program like The Gimp or Adobe Photoshop.

This data set has been chosen as traffic signs are bound by law to be the same all over the country and as such are uniform. Furthermore traffic signs have to be easily viewable by drivers. The only problem with detecting traffic signs using a filter is that usually you don't see them from right upfront, but more from side angles. Also lighting plays a crucial role in shape distortion when applied to traffic signs. This is because traffic signs are coated in a special coating which reflects light to facilitate night visibility. And the most common problem is scale, as you can view traffic signs from different distances there has to be scale invariance.

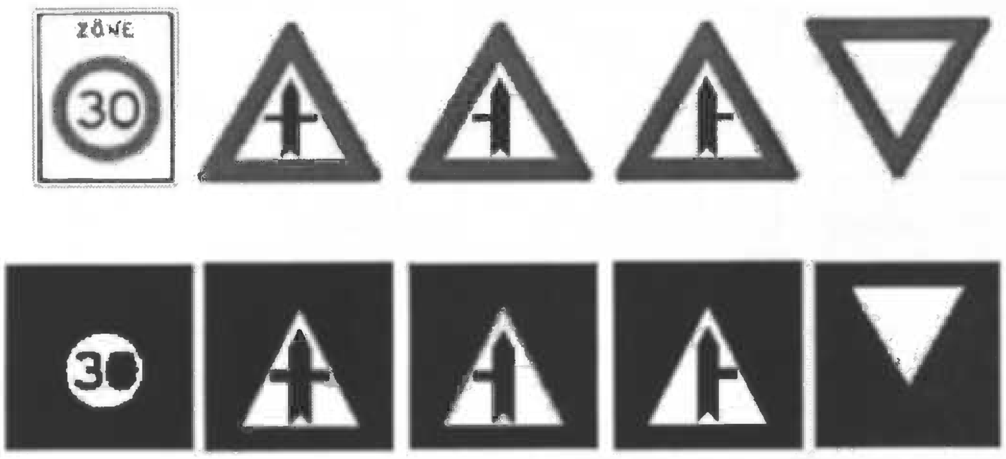

Figure 5.1: Traffic signs and their reference shapes

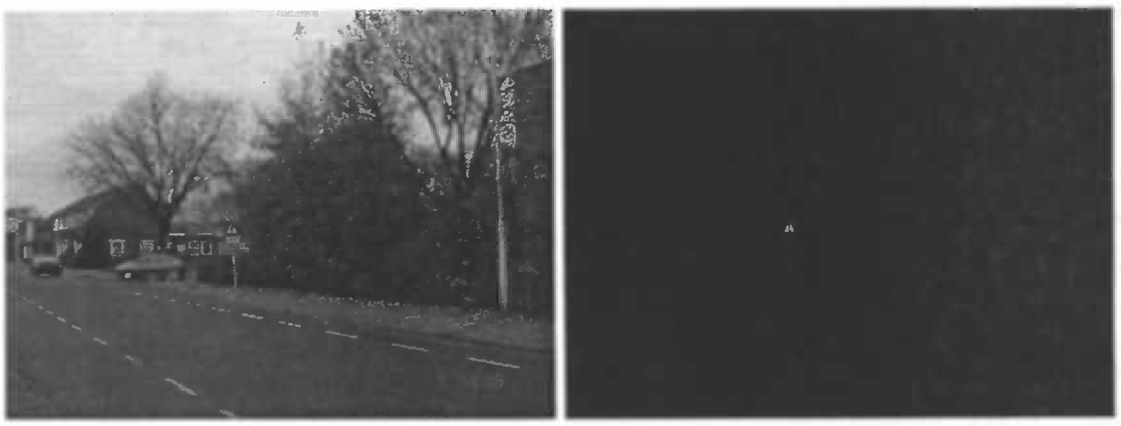

Figure 5.2: Sample input image (left) and template (right)

5.3 Set-up

For the experiments to run parameters have to be defined. The most important parameter to be defined is the error parameter (ϵ). We will run the filter on the data set with a range of error parameters as to create enough data points to make a plot. We will be using Hu's moment invariants as attributes for this experiment as they are scale, translation and rotation invariant. All these invariances are needed, because the traffic signs in the data set will not all be in the same place in the image, nor will they be shot from the same distance in every image and finally there can be a slight shift in the angle from which the photograph has been taken which results in a slight rotation. So that means that translation, scale and rotation invariance is needed. We will be running the same test on the data set for different values of the parameter `windowsize`. The paramter `windowsize` will be in the range $0 \leq n \leq 9 : n \in \mathbb{N}$.

5.4 Template and True Positive Nodes

A new command-line parameter was added, the parameter template. This parameter is an image of equal size as the input image. This image contains mostly black pixels. Only the pixels that

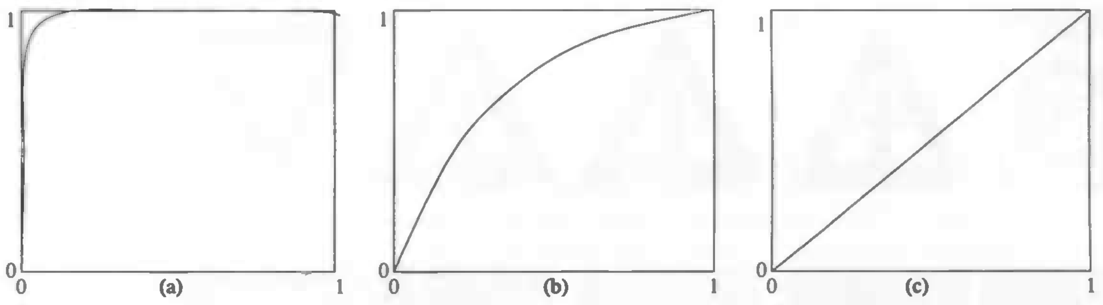

Figure 5.3: Sample ROC curves

correspond to the object to be detected in the corresponding input image are set to white. While flooding the max-tree every pixels is scanned and if a pixel corresponds to a white pixel in the template then a counter, which counts 'groundtruth pixels' or pixels that are white in the template, is increased by one. This counter will be used to determine if a node should be labeled a 'True Positive' node. A 'True Positive' node is a node from whose pixels at least 66.67% is a groundtruth pixel. Thus a 'True Positive' node is a node that should be retained as it contains (part of) the shape that is to be detected.

5.5 Validation

For each value of `windowsize` we will make a ROC curve. An ROC curve is a graph which shows the percentage of false positives detected against the percentage of true positives detected. A few examples of ROC curves can be seen in figure 5.3.

Example (a) in figure 5.3 shows a good to very good filter. As can be seen in the first 10% of false positives the percentage of true positives shoots towards 90-95%. Which implies that for a 10% of false positives, 90-95% of true positives are gained. At about 20% of false positives the filter has 99% true positive detection. The last 1% of true positives is gained with the remaining 40% of false positives. Example (b) in figure 5.3 shows a bad-average filter. It gains somewhat more true positives than false positives, but not that many. Example (c) in figure 5.3 Shows the random choice. This is what occurs when just randomly chosing nodes to retain. The amount of true positives gained is equal to the amount of false positives gained.

As described in the previous section some max-nodes in the max-tree have been labeled 'True Positive'. After the max-tree has been filtered, the max-tree is again traversed and the following counters are used:

- *Number of nodes.* After the traversal of the max-tree this counter will return the total number of max-nodes in the max-tree.
- *Number of True Positive nodes* After the complete traversal of the max-tree this counter will return the total number of nodes that have been labeled 'True Positive'.
- *Number of True Positive nodes detected* When the max-tree has been traversed this counter will return the number of nodes that have been labeled 'True Positive' and are retained by the filter.
- *Number of False Positive nodes detected.* When the tree has been traversed this counter returns the number of max-nodes that has been retained by the filter, but is not labeled 'True Positive'.

With these counters it is easy to determine the percentage of true and false positives. The percentage of true positives can be calculated by dividing *Number of True Positve nodes detected* by *Number of True Positive nodes.* This will give a value between 0 and 1, if you want the actual

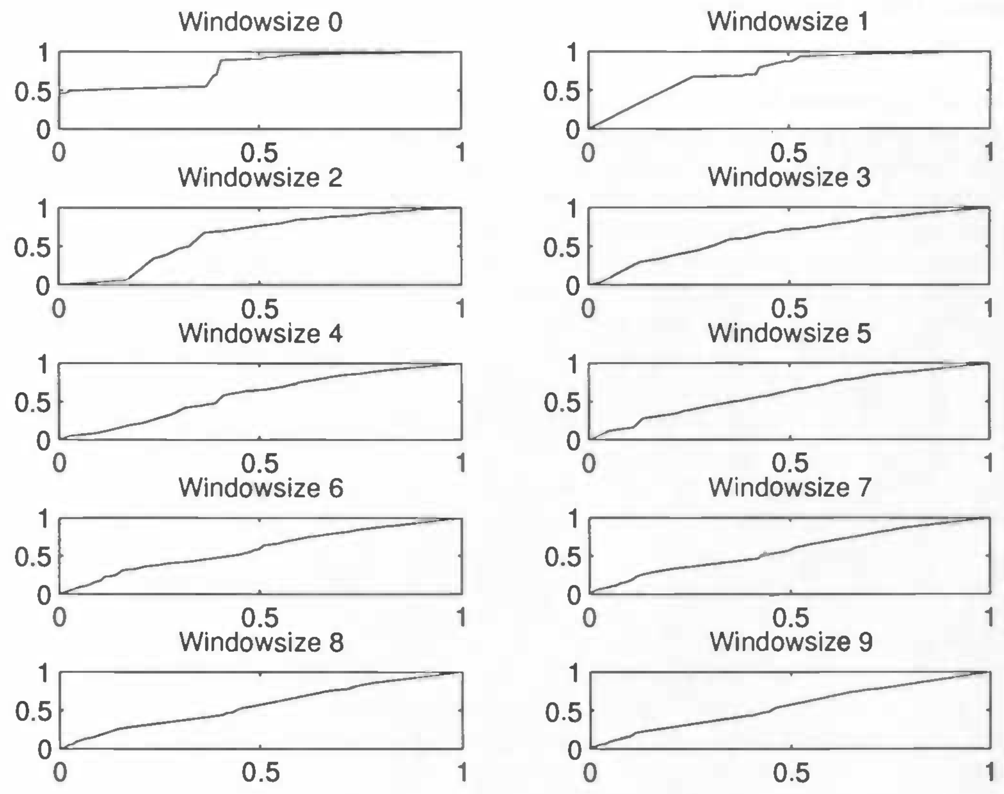

Figure 5.4: ROC curves for windowsizes 1 through 9 for the entire data set

percentage you have to multiply this value with 100%, but for plotting graphs that is not necessary. The percentage false positives is calculated by dividing *Number of False Positive nodes detected* by (*Number of Nodes - Number of True Positive nodes*), which will also give a value between 0 and 1. But as the number of false positive nodes by far outnumbers the number of true positive nodes it is exceptionally hard to find only true positive nodes in the abundance of false positive nodes. It is literally searching for a needle in a haystack.

5.6 Results

Now that we've discussed the setup and validation of experiments it's time to run some experiments. We will create an ROC curve for the windowsizes 0 through 9. To do this we need to apply the filter to our data set multiple times per windowsize value. Eacu of these applications of the filter will have a different value for ϵ (Error margin or dissimilarity between two vectors based upon Euclidian distance as described in [16]). For these experiments the filter will be applied to the data set for the values of 0.001 through 0.040 with increments of 0.001. This will result in 40 lines as a result of the filter per windowsize value. These lines contain the total amount of nodes in the max-tree, the amount of true positive nodes to be detected in the max-tree, the amount of true positive nodes detected and the number of false positive nodes detected. As described in the previous section we can create a point in $\mathbb{R}^2$ using these lines. Thus in fact we get 40 points in $\mathbb{R}^2$, which when plotted will result in an ROC curve. Figure 5.4 shows the ROC curves for the entire data set. The top left reprents windowsize 0 and the bottom right windowsize 9. As can be seen, the higher the windowsize the worse the filter will become. In figures 5.5 and 5.6 are some examples of what result images look like.

One of the reasons for the poor performance is that Hu's moment invariants are not scaled. By this we mean that one of Hu's moment invariants can have a value between 0 and 1, whereas another moment invarient's value will be in the order of magnitude of 10^6. Thus if we add the vector attributes of different nodes together, the latter moment invariant will increase by a value near 10^6 and the former moment invariant by a value of about 1. When calculating the Euclidian distance the moment invariant of 10^6 will have considerable more influence than the moment invariant of 1. Thus we only calculate the dissimilarity between the moment invariant of size 10^6, because its sheer magnitude makes the other moment invariant negligible small. By combining vector attributes which contain Hu's moment invariants we make the bigger moment invariants bigger and we keep the smaller moment invariants small. So when calculating the dissimilarity by means of Euclidian distance only the bigger moment invariants will be of consequence, whereas they in no way describe the entire shape. Thus this will result in more false positive detections and less true positive detections as shown in figure 5.4.
Another reason could be that we are using a binary reference image which we interpret as a greyscale reference image instead of using real greyscale reference images. Yet another reason may be that we are using an artificial perfect reference image instead of a couple of real-life images which we could use as references. Maybe Hu's moment invariants are not the ideal attribute for this kind of filter, and some other moment invariants might be better. All in all there may be many reasons for the poor performance of the filter for windowsize greater than 0.

Of course the type of traffic sign can also influence results, therefore figures 5.7, 5.8 and 5.9 show the ROC curves for the separate traffic signs. Please bear in mind that figure 5.8 represents three traffic signs, but due to their similarity we have counted them as one set. A peculiar point of note would be that for windowsize 1 in figure 5.7 the filter seems to do better.

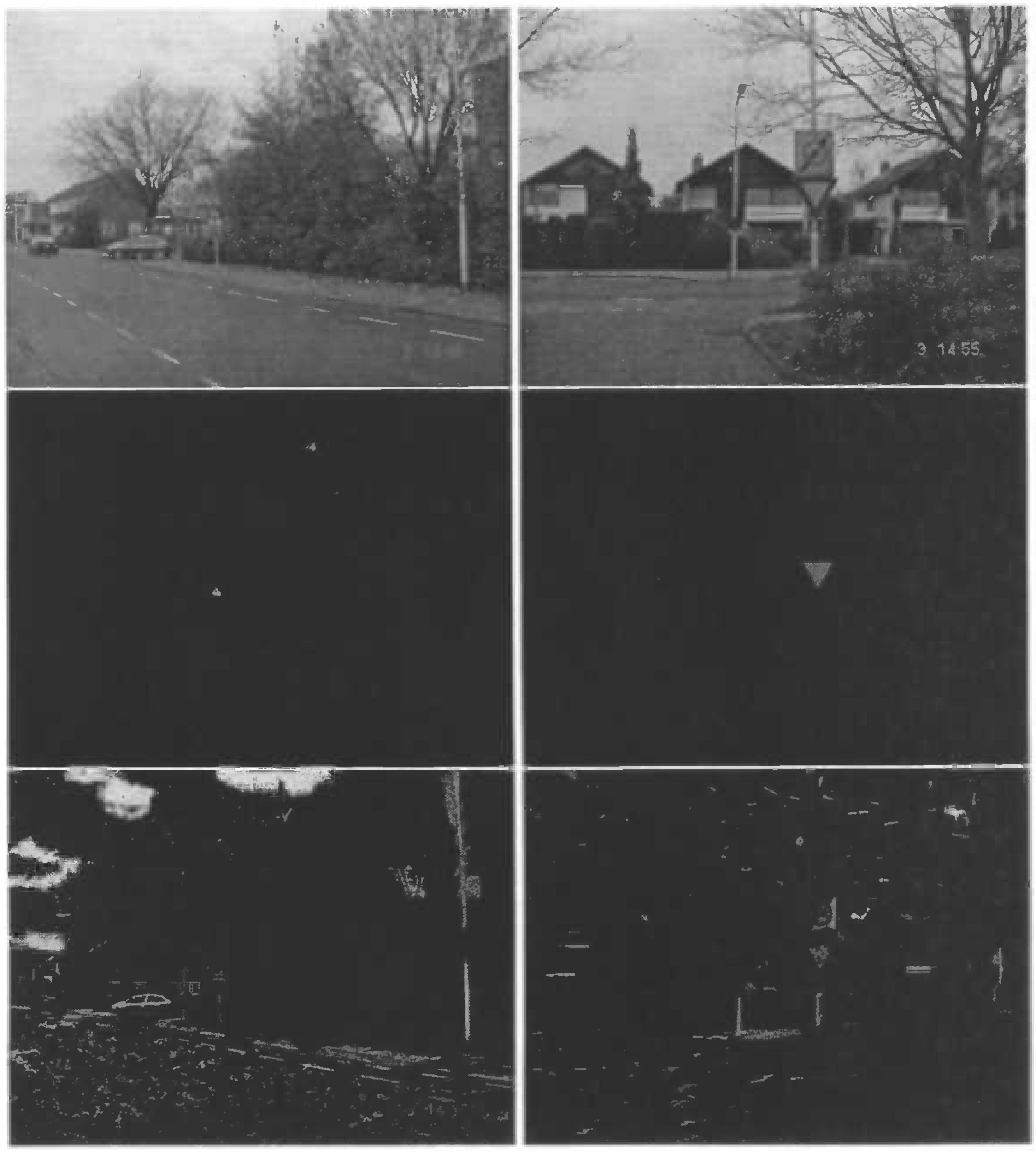

Figure 5.5: Original image (top), filter results with windowsize 0 and (middle) and windowsize 5 (bottom). For the left column $\epsilon = 0.013$ was used and for the right column $\epsilon = 0.001$

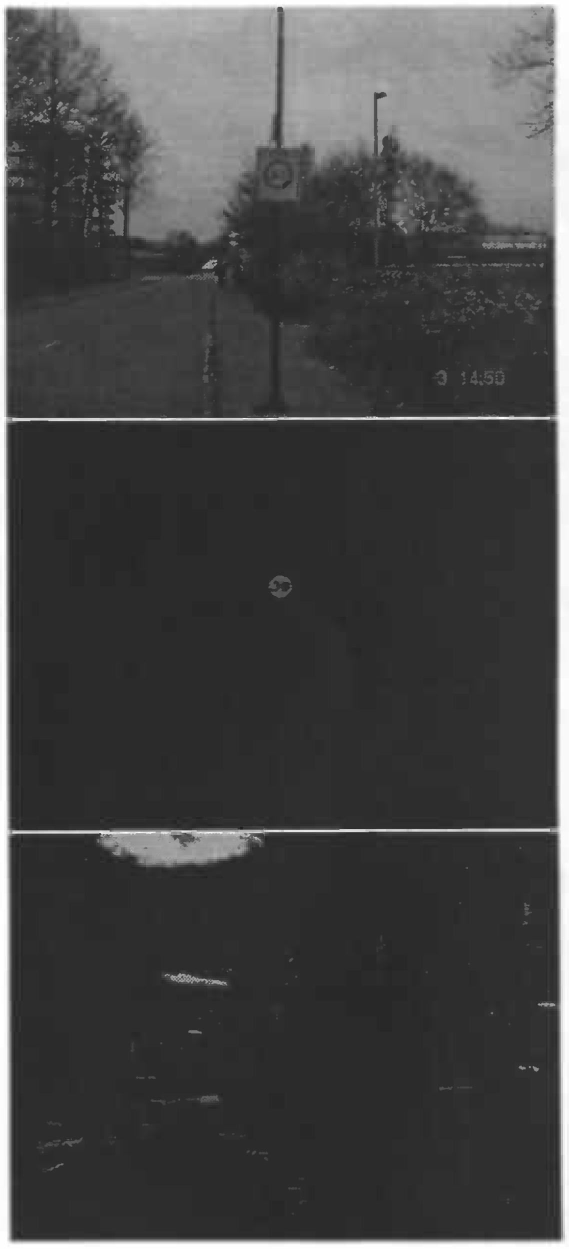

Figure 5.6: Original image (top), filter results with windowsize 0 (middle) and windowsize 5 (bottom). Here $\epsilon = 0.002$

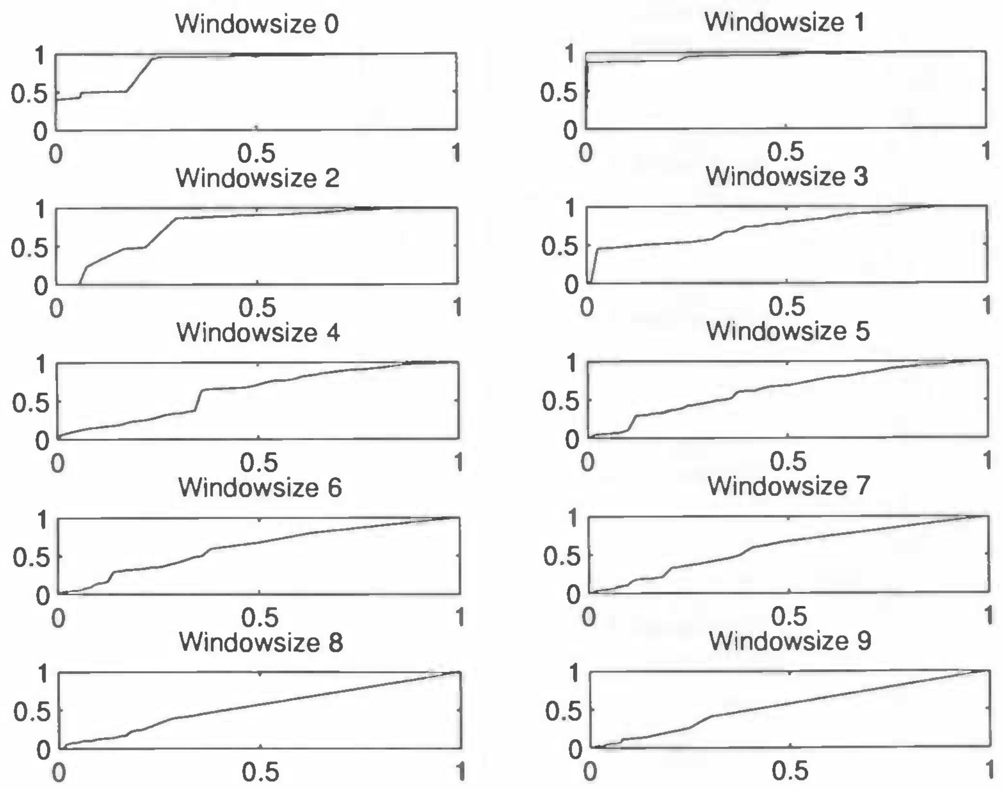

Figure 5.7: ROC curves with windowsizes 0 through 9 for the traffic sign shown in figure 5.1(left)

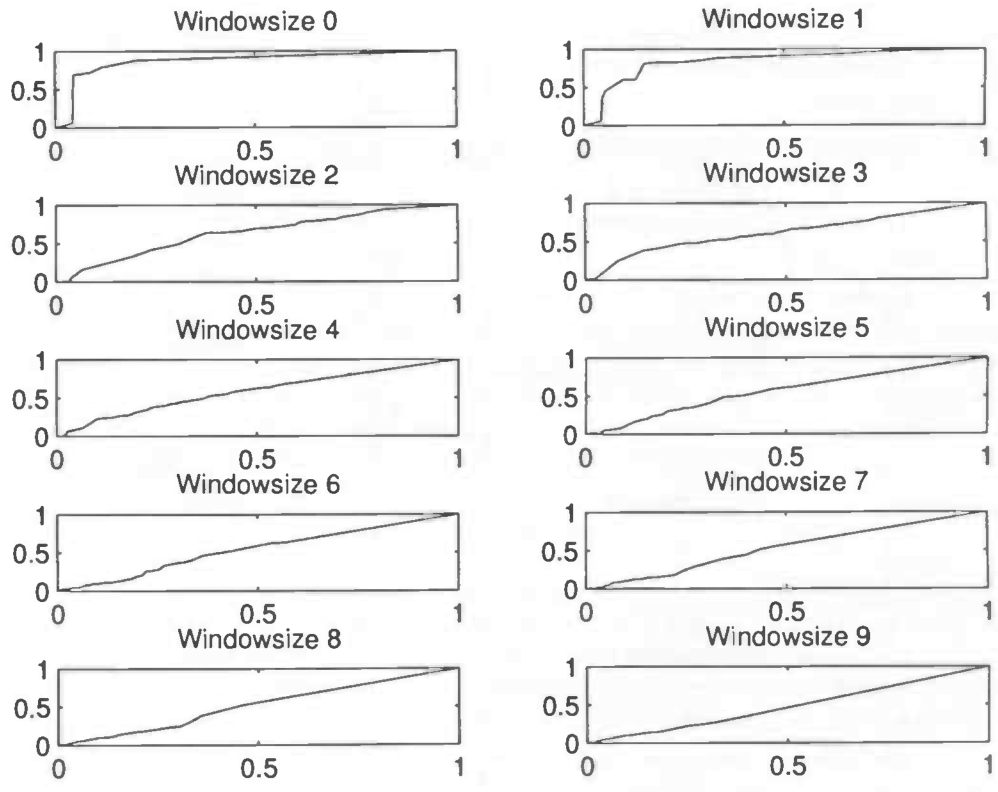

Figure 5.8: ROC curves with windowsizes 0 through 9 for the traffic sign shown in figure 5.1(middle three)

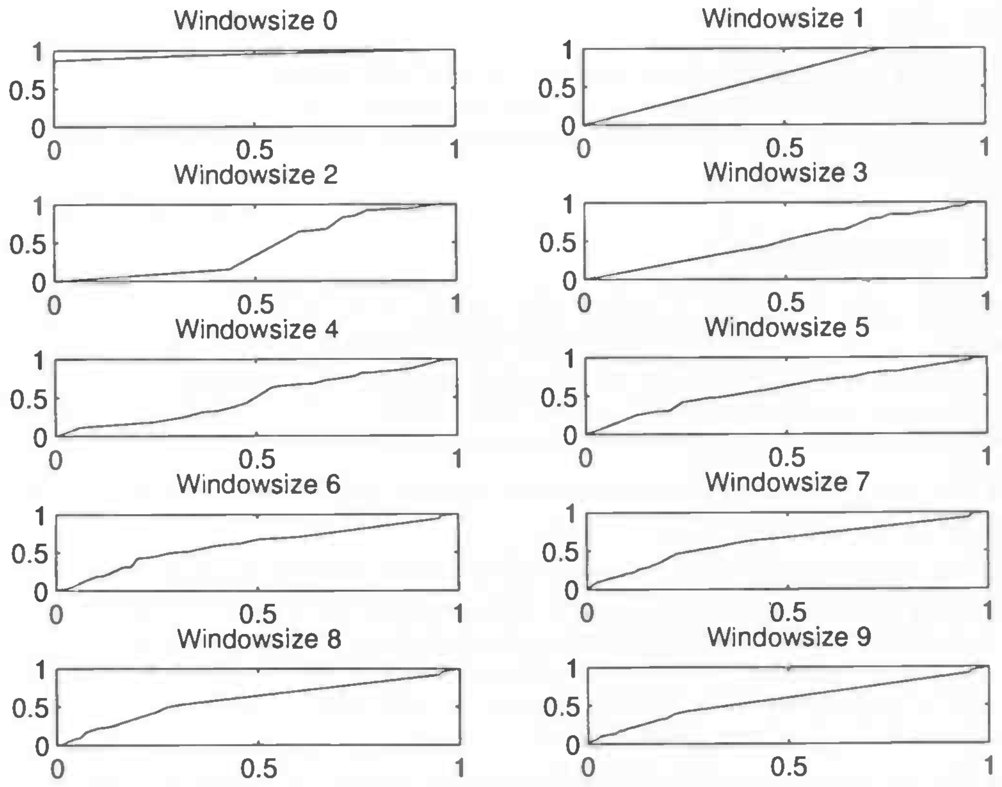

Figure 5.9: ROC curves with windowsizes 0 through 9 for the traffic sign shown in figure 5.1(right)

Chapter 6

Conclusions and Future Work

6.1 Introduction

In this chapter we will give our conclusions with regard to the experiments run and discuss options for future research.

6.2 Conclusions

Seeing the results it can be concluded that using a vector attribute containing Hu's moment invariants on this data set using the current reference images in local-greyscale vector-attribute filters leads to worse results if the number of greyscale levels is increased. The best results are acquired using windowsize 0 which means using binary vector atrributes. As discussed in the previous chapter the following reasons can result in the poor performance:

- The use of Hu's moment invariants as the separate moment invariants do not have the same range.
- The use of a binary reference image and interpreting it as a greyscale reference image. By doing this we take the perfect binary reference image and multiply its attribute vector with the scalar `windowsize` which leads to the perfect greyscale reference image. As no object in real life will be as perfect as the reference image in the binary case, it definitely will not be in the greyscale case. We only increase the error found and thus more false positives will occur before we detect a true positive.
- The use of Euclidian distance as a dissimilarity measure. You might not want to give all the attributes in a vector attribute the same weight, therefore you might want to use another dissimilarity measure.

Hu's moment invariants are, as stated, translation-, scale- and rotation invariant. But do we actually want to have rotation invariance? We would argue we need limited rotation invariance, but no total rotation invariance because the orientation of a traffic sign does indeed matter. Look for example at figure 5.1(right). If that sign was turned 180 degrees it would look very similar to figure 5.1(middle three), though they have distinctly different meanings. So we would argue that total rotation invariance is not needed. Instead it is better to use moment invariants with limited rotation invariance as to compensate for camera skew. Another option would be to use affine moment invariants [3] which are invariant to rotation in the third dimension, instead of the two displayed in an image. This might increase performance as you hardly ever look straight at a traffic sign, instead you look at it from an angle. The affine moment invariants compensate for this angle. Yet another option would be blur invariant moments [15] as they compensate for blur. Real-life images almost always contain some degree of blur, these moment invariants will compensate for that and could lead to better results. Using other dissimilarity measures instead of Euclidian distance to measure the difference between two vectors may lead to better results e.g. nearest neighbour, support vector machines or neural networks. measure for instance. A nearest neighbour scheme checks the input vector versus a set of reference vectors and classifies the input as belonging to the category to which it is nearest. A support vector machine is a supervised learning method. When used it creates a hyperplane which separates the data into two classes 'yes' and 'no'. The decision for classification is made by training the SVM by giving it examples and their answers. After training the machine it will calculate the closest distance between an example and the input and then determine 'yes' or 'no'. A Neural Network is a combination of elementary processing units (neurons). Each of these neurons has a number of input and generates one output. Each input has a weight attached to it and the output is generated by a function of the inputs and their weights.

The use of real-life reference images could lead to better results as no image will contain the perfect reference image, but most images may contain one of the offered real-life reference images. Thus you will decrease the error found.

Another conclusion is that the current filter uses a staggering amount of memory. This is due to the fact that we do not dynamically allocate memory, we allocate the theoretical maximum of memory the filter might use.

6.3 Future Work

As mentioned in the conclusions the following items can be considered for future research:

- Other attributes, for instance affine and blur moment invariants.
- Real-life reference images instead of one perfect artificial reference image.
- Other dissimilarity measures for example nearest neighbour, neural networks or support vector machines.
- Dynamic memory allocation as to reduce the amount of memory the filter uses and thus making it more useful for commercial applications.
- Different notions of connectivity can be used to increase the robustness of the filter. For example what Ouzounis and Wilkinson describe in [10]

Bibliography

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

[2] F. Cheng and A. N. Venetsanopoulos. An adaptive morphological filter for image processing. *IEEE Trans. Image Proc.*, 1:533–539, 1992.

[3] J. Flusser and T. Suk. Pattern recognition by affine moment invariants. *Pattern Recognition*, 26:167–174, 1993.

[4] H.J.A.M. Heijmans. *Morphological Image Operators*. Academic Press, Boston, 1994.

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

[6] A. Khotanzad. Invariant image recognition using Zernike moments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:489–497, 1990.

[7] P. Maragos and R.W. Schafer. Morphological filters part i and ii. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35:1153–1184, 1987.

[8] G. Matheron. *Eléments pour une Théorie des Milieux Poreux*. Masson, Paris, 1967.

[9] R. Mukundan, S. H. Ong, and P. A. Lee. Image analysis by Tchebichef moments. *IEEE Trans. Image Proc.*, 10:1357–1364, 2001.

[10] G. K. Ouzounis and M. H. F. Wilkinson. Countering oversegmentation in partitioning-based connectivities. In *Proc. Int. Conf. Image Proc. 2005*, pages 844–847, Genova, Italy, September 11–14 2005.

[11] J.B.T.M. Roerding and H.J.A.M. Heijmans. Mathematical morphology for structures without translation symmetry. *Signal Processing*, 15:271–277, 1988.

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

[13] J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 2 edition, 1982.

[14] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. PWS Publishing, 1999.

[15] T. Suk and J. Flusser. Blur and affine moment invariants. In *Proc. 16th Int. Conf. Pattern Rec.*, volume 4, page 40339, 2002.

[16] E. R. Urbach, N. J. Boersma, and M. H. F. Wilkinson. Vector-attribute filters. In *Mathematical Morphology: 40 Years On, Proc. Int. Symp. Math. Morphology (ISMM) 2005*, pages 95–104, Paris, 18-20 April 2005.

[17] 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.

[18] L. Vincent. Grayscale area openings and closings, their efficient implementation and applications. In *Proc. EURASIP Workshop on Mathematical Morphology and its Application to Signal Processing*, pages 22–27, Barcelona, Spain, 1993.

[19] L. Vincent. Morphological area openings and closings for grey-scale images. In Y.-L. O, A. Toet, D. Foster, H. J. A. M. Heijmans, and P. Meer, editors, *Shape in Picture: Mathematical Description of Shape in Grey-level Images*, pages 197–208. NATO, 1993.

[20] 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.

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