Pattern recognition systems – Lab 4

Distance Transform (DT). Pattern Matching using DT

1. Objectives

In this lab session we will study the algorithm that performs a Distance Transformation of a binary image (object and background). Then we want to evaluate the pattern matching score between a known template object (e.g. a pedestrian contour) and an unknown object (e.g. a contour of other object) to decide if the unknown object is or not similar to the template object. The less the pattern matching score is the more similar is the unknown object to the template.

2. Theoretical Background

2.1. Distance Transform

A distance transform, also known as distance map or distance field, is a representation of a digital image. The choice of the term depends on the point of view on the object in question: whether the initial image is transformed into another representation, or it is simply endowed with an additional map or field. The map supplies each pixel of the image with the distance to the nearest obstacle pixel. A most common type obstacle pixel is a boundary pixel in a binary image.

The distance transform is an operator normally only **applied to binary images**. **The result** of the transform **is a grayscale image** that looks similar to the input image, except that the intensities of points are changed to show the distance to the closest boundary from each point.

One way to think about the distance transform is to first imagine that foreground regions in the input binary image are made of some uniform slow burning inflammable material. Then consider simultaneously starting a fire at all points on the boundary of a foreground region and letting the fire burn its way into the interior. If we then label each point in the interior with the amount of time that the fire took to first reach that point, then we have effectively computed the distance transform of that region.

See the next image for an example of a chessboard distance transform on a binary image containing a simple rectangular shape. In the left image the pixels with value “0” represent object pixels (boundary pixels) and those with value “1” background pixels. In the right image is the result of applying the Distance Transformation using the chessboard metric, where each value encodes the minimum distance to an object pixel (boundary pixel):

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Binary Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	2	2	2	1	0
0	1	2	3	2	1	0
0	1	2	2	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Distance transformation

Usually the transform/map is qualified with the chosen metric. For example, one may speak of Manhattan distance transform, if the underlying metric is Manhattan distance. Common metrics are:

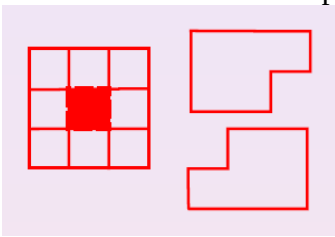
- Euclidean distance;
- Taxicab geometry, also known as *City block distance* or *Manhattan distance*;
- Chessboard distance.

There are several algorithms for implementing DT:

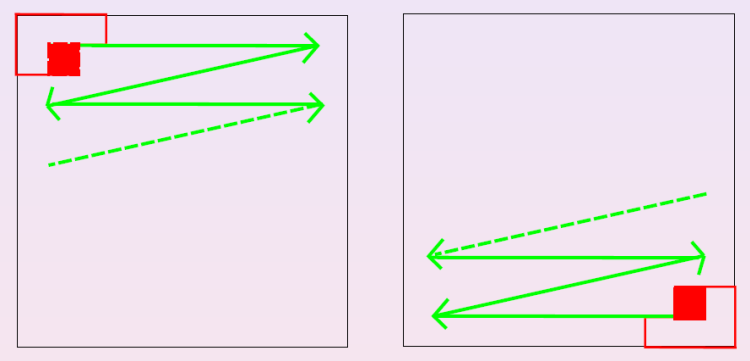
- Chamfer based DT;
- Euclidian DT;
- Voronoi diagram based DT.

We will present the **Chamfer based DT** which is a simple method and it is very fast (it requires only two scans of the binary image). The sketch of the algorithm is:

- A 3x3 weight mask is chosen which is decomposed into two parts:



- A double scan (first top-down, left-right and second bottom-up, right-left) of the image (with the corresponding two parts of the mask – see the figure below) is required to update the minimum distance:



While scanning (forward and backward) the source image, the next update operation should be performed on the DT image:

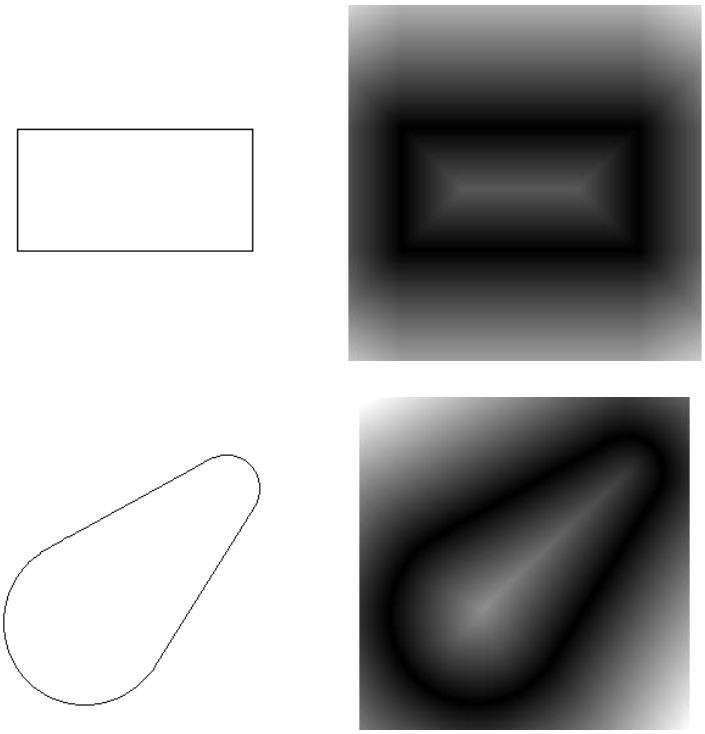
$$DT(i,j) = \min_{(k,l)\in Mask}(DT(i+k, j+l) + weight(k,l))$$

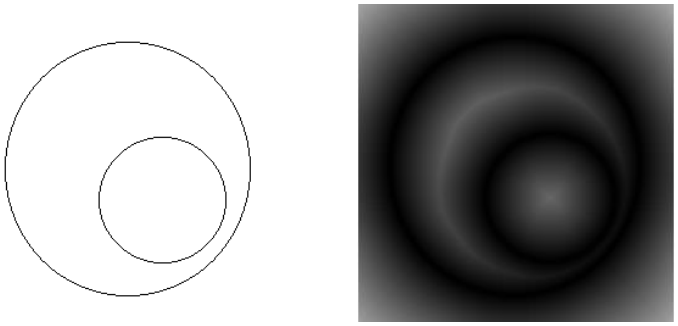
- The initialization step is:

$$DT(i,j) = 0 \quad if \quad (i,j) \in Object$$
$$DT(i,j) = +\infty \quad if \quad (i,j) \notin Object$$

- Remarks:
  - The displacement (0,0) with weight 0 belongs to the mask;
  - When applying DT on an binary image (8 bits/pixel) where the index 0 is encoding the object and index 255 the background and when we want to obtain a grayscale DT image (8 bits/pixel) then the value $+\infty$ from the algorithm should be substituted with the value 255;
  - If we want to obtain a DT image where each pixel approximates the Euclidian distance from the nearest object pixel then the *weight(k, l)* should be set to a value:
    - *wHV* if the direction (*k, l)* is a *horizontal* or *vertical* one starting from the origin of the mask
    - *wD* if the direction (*k, l)* is a *diagonal* one starting from the origin of the mask
    - the relation between *wHV* and *wD* should be $wD \approx wHV \cdot \sqrt{2}$
    - e.g.: a good choice is (*wHV, wD*) = (5, 7) because $7 \approx 5 \cdot \sqrt{2}$

Examples of DT image results using Chamfer method with the setting (*wHV, wD*) = (2, 3). Left is the original image and right is the DT image:





2.2. Pattern Matching using DT

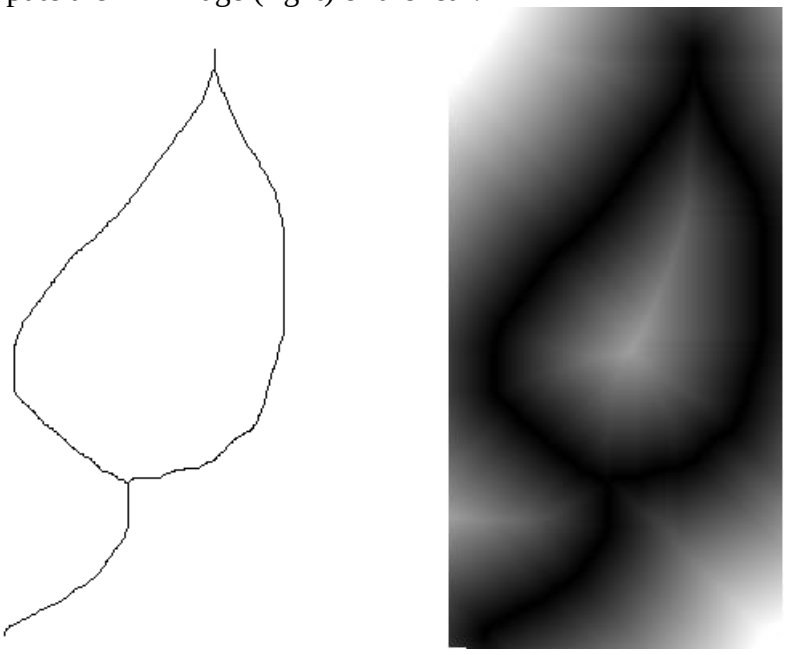
Our goal in this part is to compute the pattern matching score between a template, which represents a known object, and an unknown object. The score can be used to quantify the similarity between the template and the unknown object. We consider that both the template and the unknown object images have the same dimensions.

The steps of computing the pattern matching score:

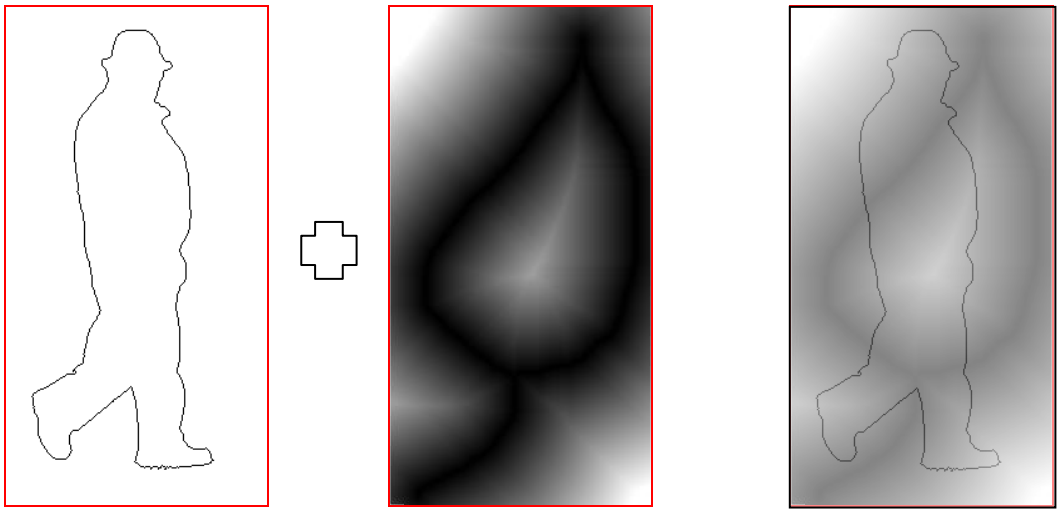
- compute the DT image of the template object;
- superimpose the unknown object image over the DT image of the template;
- the pattern matching score is the average of all the pixel values from the DT image that lie under the unknown object contour.

Example: Consider that the template object is a leaf (contour) and the unknown object is a pedestrian (contour):

- Compute the DT image (right) of the leaf:



- Superimpose the pedestrian image over the DT image of the leaf; compute the pattern matching score that is the average of all the pixel values from the leaf DT image that lie under the pixels of the pedestrian contour:



3. Implementation details

Opening the source image as a grayscale image:

```
Mat img = imread("filename", IMREAD_GRAYSCALE);
```

Initializing the DT image as the source image:

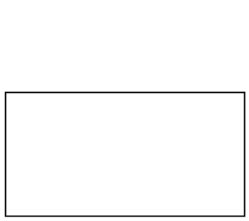
```
Mat dt = src.clone();
```

Convenient 8-neighborhood access at position *(i,j)*:

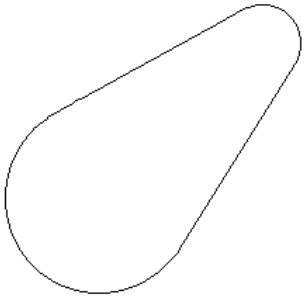
```
int di[8] = {-1,-1,-1,0,0,1,1,1};
int dj[8] = {-1,0,1,-1,1,-1,0,1};
int weight[8] = {0,1,0,1,1,0,1,0};
for(int k=0; k<8; k++)
     uchar pixel = img.at<uchar>(i+di[k], j+dj[k]);
```

4. Practical work

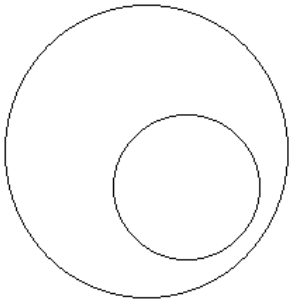
1. Implement the Chamfer Distance Transform algorithm. Compute and visualize the DT images for: *contour1.bmp, contour2.bmp, contour3.bmp*. Results should coincide with ones presented in the text. Object pixels are black and background pixels are white.



contour1.bmp

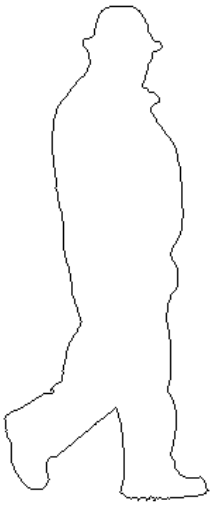


contour2.bmp

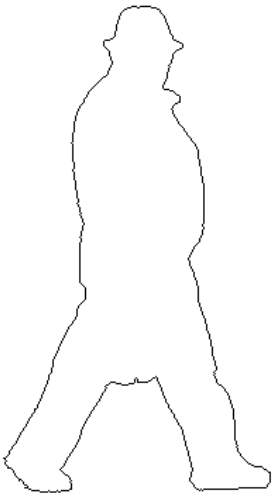


contour3.bmp

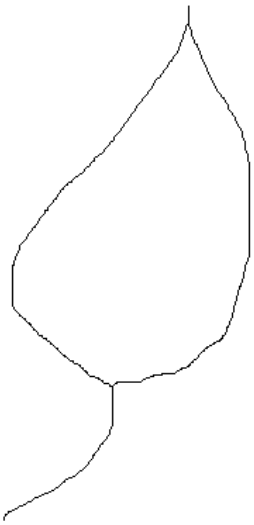
2. Compute the DT image for *template.bmp*. Evaluate the matching score between the template and each of the two unknown objects (*unknown_object1.bmp* – pedestrian contour, *unknown_object2.bmp* – leaf contour). The matching score is given as the average of the DT image values that lie under the contour points from the object image.



template.bmp



unknown_object1.bmp



unknown_object2.bmp

3. Compute the matching score by switching the roles of template and unknown object. Are the scores the same?

4. Before performing the matching score translate the unknown object such that its center of mass coincides with template's center of mass. Estimate the center of mass based on the contour points only.

5. Optionally, implement the true Euclidean Distance Transform. Why is the Chamfer Distance Transform different?

5. References

[1] Wikipedia The Free Encyclopedia – *Distance Transform*,
http://en.wikipedia.org/wiki/Distance_transform
[2] D. Coeurjolly – *Distance Transform*, University Claude Bernard Lyon
http://www.cb.uu.se/~tc18/subfields/distance_skeletons/DistanceTransform.pdf
[3] Compendium of Computer Vision – *Distance Transform*,
http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm