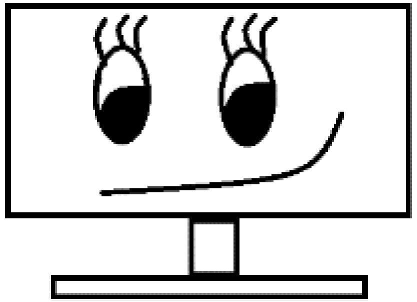
Seneca



CVI620/ DPS920
Introduction to Computer Vision

Object Recognition

Seneca College

Vida Movahedi

Overview

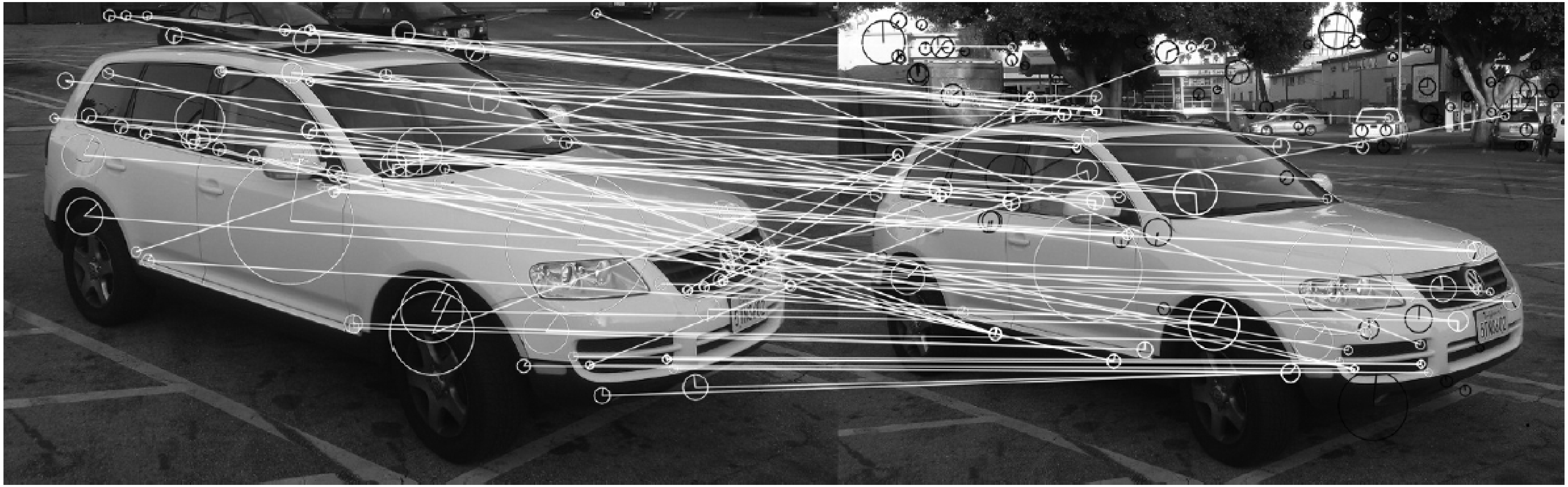
- Object Detection methods
  - Keypoint matching
  - Template matching
  - Chamfer matching

Keypoint Matching

3 main steps [2]

1. Search an image and find all keypoints
    - Different methods
2. Create a descriptor for every keypoint found
3. Find matches in another image or set of images
    - By comparing the descriptors of keypoints
    - Using a matching measure

- Example:
    - Tracking: matching with the next frame
    - Recognition/ Identification: Match with a set of labelled images/ templates



Step 1: Finding keypoints

- **cv::KeyPoint** – the abstract class for finding (the location of) keypoints

```
class cv::KeyPoint {

public:

  cv::Point2f pt;       // coordinates of the keypoint
  float       size;     // diameter of the meaningful keypoint neighborhood
  float       angle;    // computed orientation of the keypoint (-1 if none)
  float       response; // response for which the keypoints was selected
  int         octave;   // octave (pyramid layer) keypoint was extracted from
  int         class_id; // object id, can be used to cluster keypoints by object
```

Step 2: Creating a descriptor

- **cv::Feature2D**
  - The abstract class for finding/ detecting/ computing descriptors
  - Often step 1 & 2 are done simultaneously
  - A descriptor list (A vector)
- Derived classes:
  - Harris detector, Shi-Tomasi detector (GoodFeaturesToTrack)
  - SimpleBlobDetector
  - Also implementations available for FAST, SIFT, SURF, ORB, BRISK, etc.
- Detect/ compute key points:
  - `detect(image, keypoints, mask) ~ detectAndCompute(image, mask, key points, noArray(), false)`
  - `compute(image, keypoints, descriptors) ~ detectAndCompute(image, noArray(), keypoints, descriptors, true)`

Step 3: Matching

- **cv::DescriptorMatcher-** abstract class
- Object recognition:
  - Compile keypoints for a variety of objects, save in a database, called a *dictionary* → this stage is called *training*
  - For each new image (*query* image), extract the keypoints
  - Compare these keypoints to the dictionary to find the best match
  - ➢ Compare one single descriptor list to a set of descriptor lists (dictionary)

- Tracking:
  - Find all points in this frame of video
  - Find matches for keypoints in the previous frame
  - Estimate how much and where to each object/ person/ vehicle has moved
  - ➢ Compare two descriptor lists

Matching functions

- A set of descriptors in the form of a cv::Mat object
  - N rows (number of descriptors/ keypoints)
  - D columns (dimensionality of each descriptor, different features describing each keypoint)

- Three functions:
  - match(): each keypoint on the query list will be matched to the “best match” from the train list
  - knnMatch(): find k nearest neighbors, finds the top k matches
  - radiusMatch(): find the best matches within a distance

Table 16-2. Available metrics for the brute force matcher, with their associated formulas; the summations are over the dimensions of the feature vector

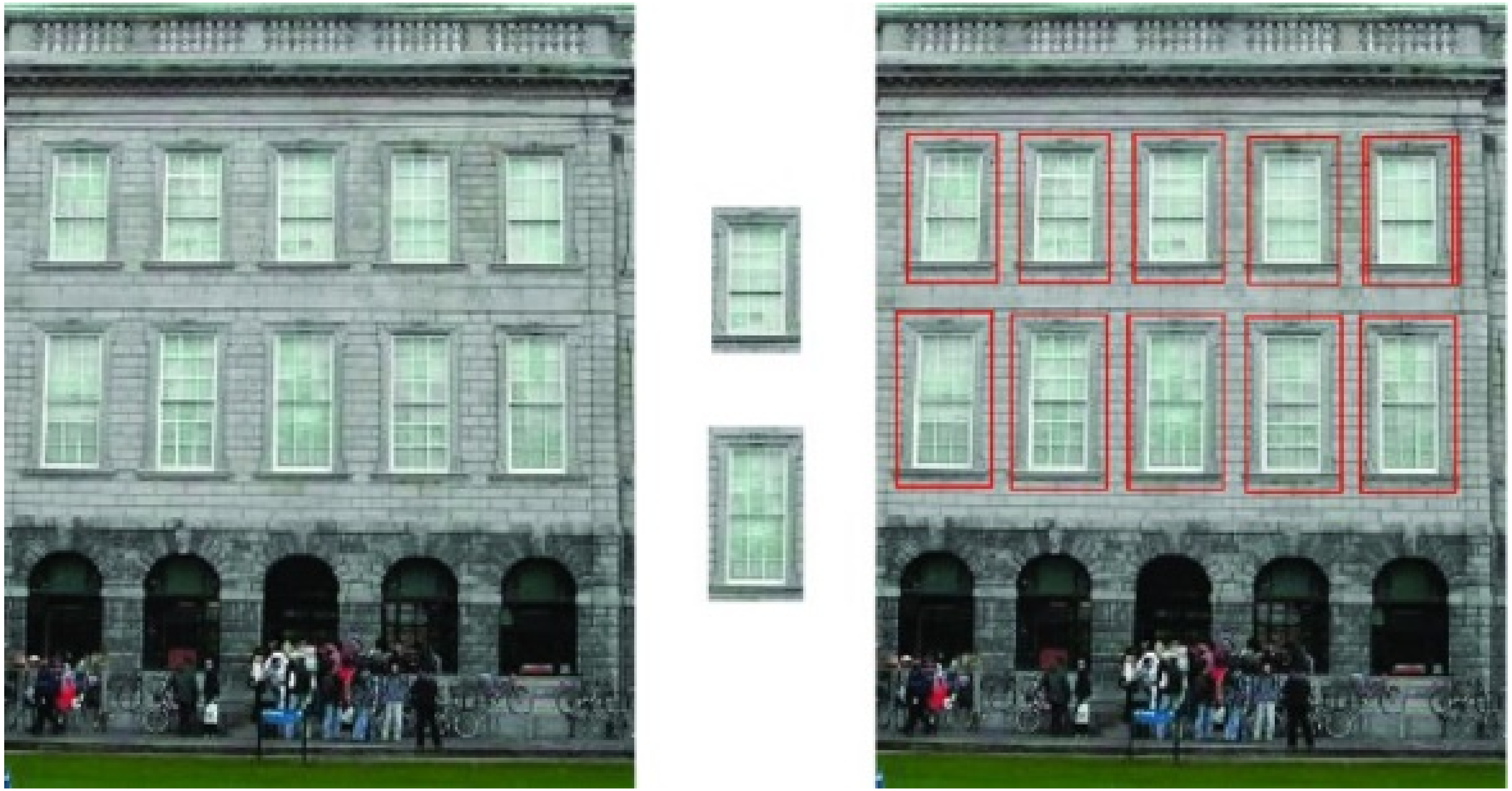
Metric	Function
NORM_L2	$dist(\vec{a}, \vec{b}) = \left[\sum_i (a_i - b_i)^2\right]^{1/2}$
NORM_L2SQR	$dist(\vec{a}, \vec{b}) = \sum_i (a_i - b_i)^2$
NORM_L1	$dist(\vec{a}, \vec{b}) = \sum_i abs(a_i - b_i)$
NORM_HAMMING	$dist(\vec{a}, \vec{b}) = \sum_i (a_i = = b_i)?1:0$
NORM_HAMMING2	$dist(\vec{a}, \vec{b}) = \sum_{i(even)} [(a_i = = b_i)\&\&(a_{i+1} = = b_{i+1}]?1:0$

Template Matching

Template Matching [3]

- Searching within an image for a match with a template (often a smaller image)
- Applications:
  - Object recognition
  - Face identification
  - Sign detection
  - Robotics
  - Tracking
  - Augmented reality

Example



Example

95-D-37825
05-D-26633
LOCH GARMAN
03-WX-4000
97-D-50260
0123456789

Template Matching algorithm

- For every possible position (i, j) of the object in the image
  - Evaluate a matching metric and store it in a matching space: $M(i, j)$
  - Search in the matching space for
    - A position with $M(i,j) > T$ (a threshold)
    - Where also $M(i,j)$ is a local maxima (more than neighboring values)
- If using a distance measure, look for minima

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

Image

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

Template

11	13	14	11	14
10	14	16	8	14
9	12	12	1	10
10	14	15	7	15

Matching Space

[3] A binary image and a binary template compared using the sum of the squared differences. The best match between the template and the image is towards the bottom right of the matching space where a matching score of 1 was obtained. Note that the comparison is only made where the template fits completely within the image and hence the matching space is smaller than the image.

Matching metrics

- Minimize total differences

$$D_{\text{SquareDifferences}}(i,j) = \sum_{(m,n)} (f(i+m, j+n) - t(m,n))^2$$

$$D_{\text{NormalisedSquareDifferences}}(i,j) = \frac{\sum_{(m,n)} (f(i+m, j+n) - t(m,n))^2}{\sqrt{\sum_{(\mathrm{m,n})} f(i+m, j+n)^2 \sum_{(\mathrm{m,n})} t(m,n)^2}}$$

- Maximize similarity

$$D_{\text{CrossCorrelation}}(i,j) = \sum_{(m,n)} f(i+m, j+n) \cdot t(m,n)$$

$$\mathrm{D}_{\text{NormalisedCrossCorrelation}}(i,j) = \frac{\sum_{(\mathrm{m,n})} f(i+m, j+n) \cdot t(m,n)}{\sqrt{\sum_{(\mathrm{m,n})} f(i+m, j+n)^2 \sum_{(\mathrm{m,n})} t(m,n)^2}}$$

In OpenCV, template matching is supported using normalised cross-correlation **CV_TM_CCORR_NORMED** *(although other measures are also supported, such as cross-correlation* **CV_TM_CCORR**, *sum of the squared differences* **CV_TM_SQDIFF** *and the normalised sum of the squared differences* **CV_TM_SQDIFF_NORMED**):

```
Mat matching_space;
matching_space.create(
        search_image.cols-template_image.cols+1,
        search_image.rows-template_image.rows+1,
CV_32FC1);
matchTemplate (search_image, template_image,
                    matching_space,
CV_TM_CCORR_NORMED);
```

Variations in the images

- When matching:
  - For every possible size
    - For every possible rotation
      - For every possible position (*i*, *j*)
        - Calculate the matching measure between image and template
  - Very expensive computation!

- Including more than one template for each object
  - For example if the goal is to locate cars in images, use a dictionary consisting of different car models, colors, perspectives, etc.
  - May contain different sizes, rotations

Chamfer Matching

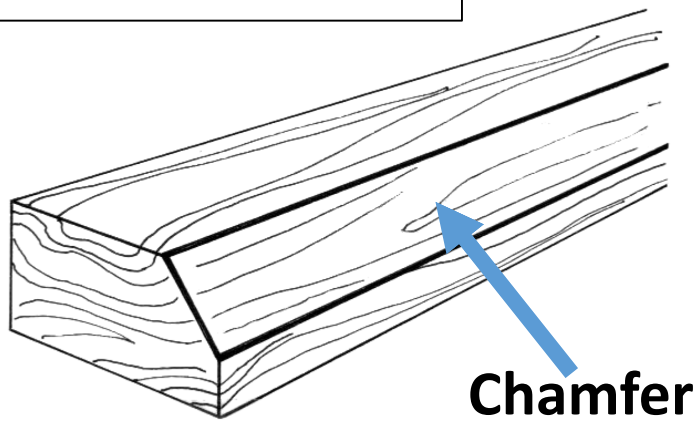
Chamfer matching

- Designed to lower dependence on the appearance of object of interest

image template

For example, looking for a square in the image; if they are not the exact same size, the template matching method will fail. Chamfer matching, however, will detect it.

- Based on distance, instead of perfect match



Chamfering algorithm

- The chamfer value for a pixel is the distance from that pixel to the nearest object or **edge** point

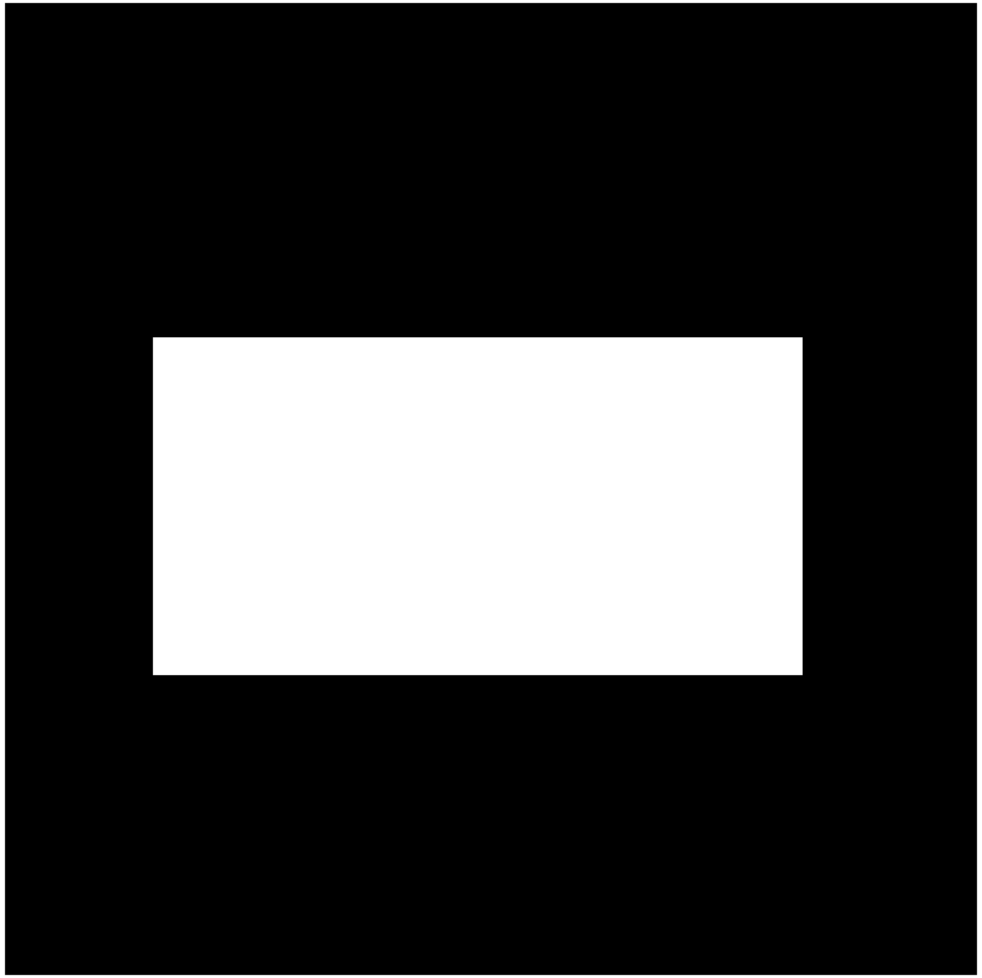
In OpenCV, to compute a chamfer image from a binary edge image we must invert the edge image so that edges are zero points (for the sake of the **distanceTransform** *routine):*

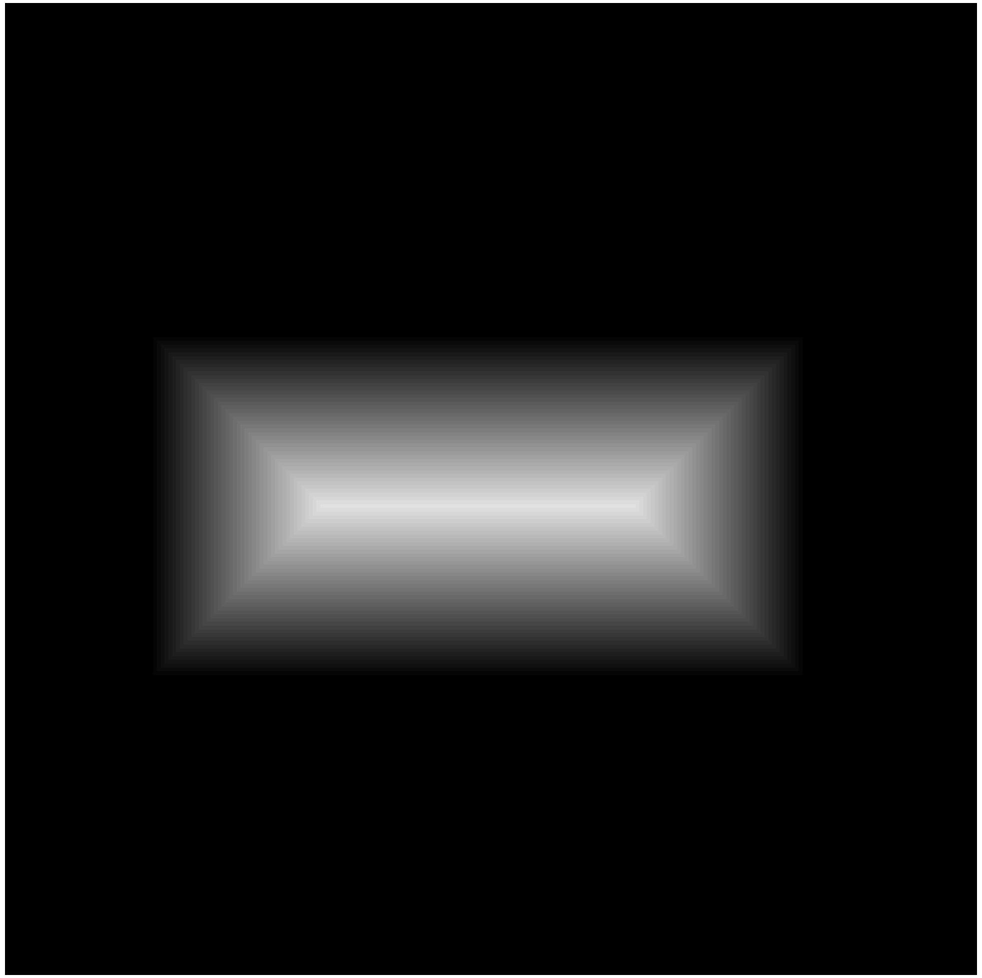
```
Canny (gray_image, edge_image, 100, 200, 3);
threshold (edge_image, edge_image, 127, 255,
THRESH_BINARY_INV);
distanceTransform (edge_image, chamfer_image,
CV_DIST_L2, 3);
```

Distance Transform

- The distance transform of an image is defined as a new image in which every output pixel is set to a value equal to the distance to the nearest zero pixel in the input image.

```
void cv::distanceTransform(
  cv::InputArray  src,                    // Input image
  cv::OutputArray dst,                    // Result image
  int             distanceType,           // Distance metric to use
  int             maskSize                // Mask to use (3, 5, or see below)
);
```





3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

1	1	1	1
1			1
1			1
1			1
1	1	1	1

Template

27.4	19.6	12.8	8	27.4
23.2	16.8	10.4	5	9.4
21	14	8	0	6
23.2	16.8	10.4	5	9.4

Matching Space

Comparing binary edge images (or any binary images)
Matching score = sum of chamfer values for nonzero pixels of template
Minimum score indicates best match
See [3], page 139 for code

References

[1] Computer Vision: Algorithms and Applications, R. Szeliski (http://szeliski.org/Book)

[2] Learning OpenCV 3, A. Kaehler & G. Bradski

- Available online through Safari Books, Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA5153244920003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US

[3] Practical introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe

- Available through Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA5142810950003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US