UNIVERSITÀ DEGLI STUDI DI PADOVA

Region growing & watershed

Stefano Ghidoni

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS SYSTEMS LAB

Agenda

- Segmentation by similarity
- Region growing: a basic algorithm
  - Seed points
  - Predicate
- Watershed segmentation

Slides by Stefano Ghidoni and Pietro Zanuttigh, figures from Gonzalez & Woods

Segmentation techniques

- Segmentation by thresholding (histogram-based)
- Region growing methods
- Watershed transformation
- Clustering-based methods
- Model-based segmentation
- Edge-based methods
- Graph partitioning methods
- Multi-scale segmentation
- Many others…

Segmentation: principles – recall

- Subdivide an image into n regions $R_1, R_2, \ldots, R_n$ such that
  - $\bigcup_{i=1}^{n} R_i = R$
  - $R_i \cap R_j = \emptyset \quad \forall i, j \ (i \neq j)$
  - Optionally: each region shall be connected
- Two main criteria:
  - Similarity (between pixels in the same region)
  - Discontinuity (between pixels in different regions)
- Region growing: what is the driving criterion?

Region growing

Region growing – idea

- Group pixels or subregions into larger regions
- Based on the concept of connectivity
  - **Local** growing of regions
- Makes use of predefined **merging criteria**
- Starts from **seed points**, to be defined
  - E.g., every pixel, pixels in a neighborhood satisfying merging criteria
- **Stopping rule**

Region growing – preconditions

- Consider a basic region growing algorithm working on:
  - The image $f(x, y)$
  - The seed array $S(x, y)$, containing 1s at the locations of seed points (0 elsewhere)
    - $f(x, y)$ and $S(x, y)$ have same dimensions
  - A predicate Q for controlling growing

Region growing – initialization

- Algorithm initialization: work on the seed points in $S(x, y)$
- Find connected components in $S(x, y)$ and erode each component until one pixel is left
  - Transforms any $S(x, y)$ in a suitable set of seed points

Region growing – growing process

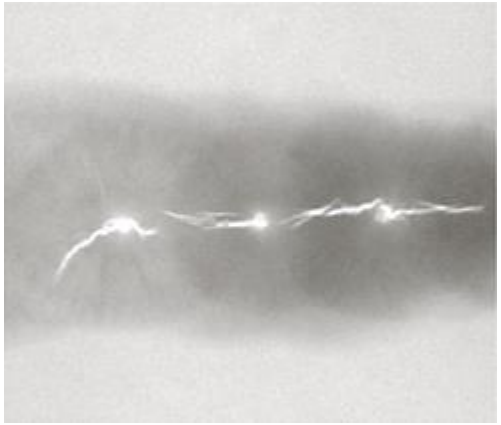
- Growing process: create an image f_Q s.t.

$$f_Q(x_i, y_i) = \begin{cases} 1 \text{ iff } Q(x_i, y_i) \text{ is true} \\ 0 \text{ iff } Q(x_i, y_i) \text{ is false} \end{cases}$$

- Create an image f_G by appending to each seed point in S the 1s in f_Q that are 8-connected to that seed point
- Label each connected component in f_G with a different region label

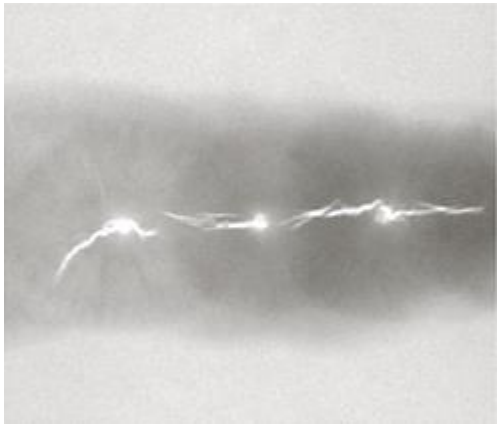
Region growing – example

- Consider an X-ray image of welding
- Goal: find defects in welding
- From the specific case: defects are the brightest areas



Region growing – example

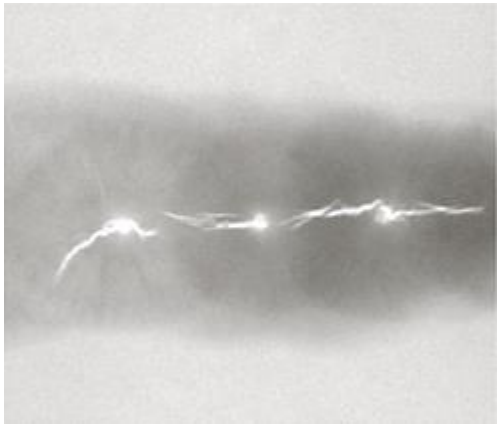
- Initialization: threshold the image to select the bright areas (99.9% in this case)



Region growing – example

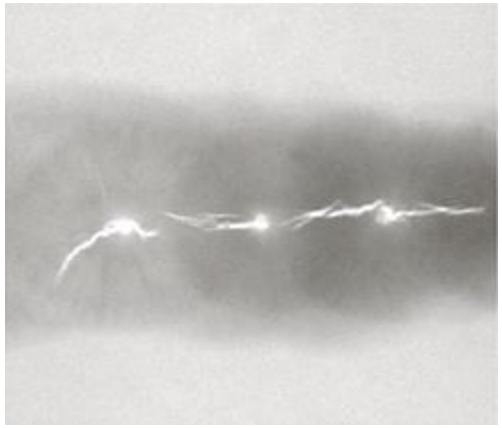
- Initialization: threshold the image to select the bright areas (99.9% in this case)
- Erode each component until 1 point is left

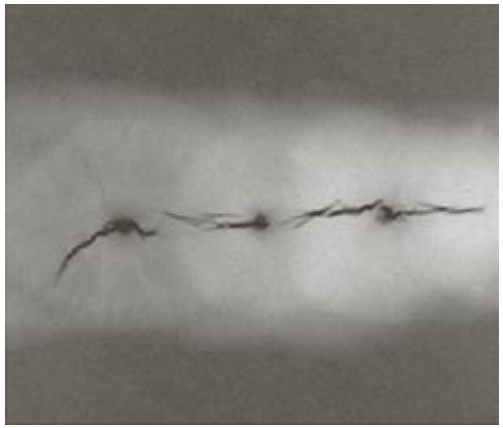
threshold molto alta toglie le regioni chiare sopra e sotto



Region growing – example

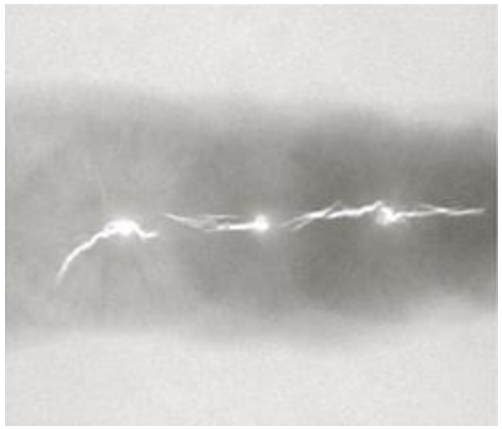
- Specify a predicate to grow the seeds
  - In our case: 8-connected & "similar" (i.e., intensity difference)
  - Given a seed in (x_s, y_s) and a threshold T, $Q(x_i, y_i) =$ true iff $|f(x_i, y_i) - f(x_s, y_s)| \leq T$
- A support image may be created to evaluate the criterion
  - E.g.: map of the distance





Region growing – example

- Final result: each welding defect is a single region
- The output of the algorithm is a region
  - The same type of regions on which morphological operators work
  - They can be applied for disconnecting weakly connected regions etc.



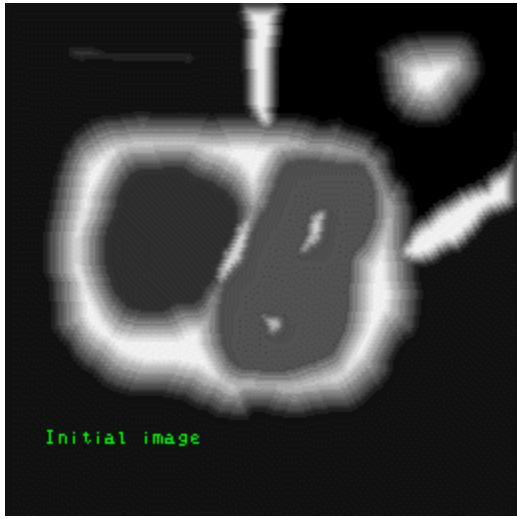
Watershed

Watershed

- Watershed features
  - Provides connected segmentation boundaries
  - Provides connected components

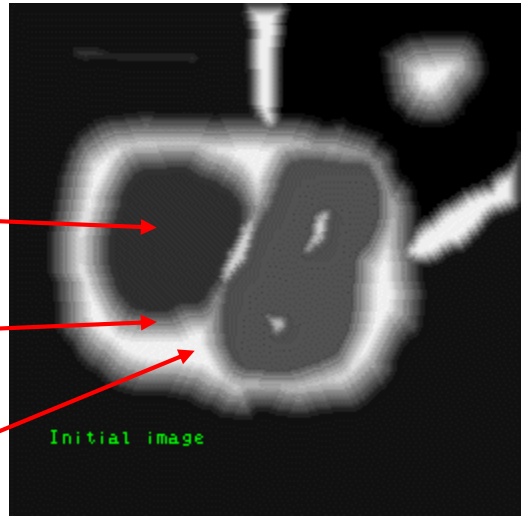
Watershed

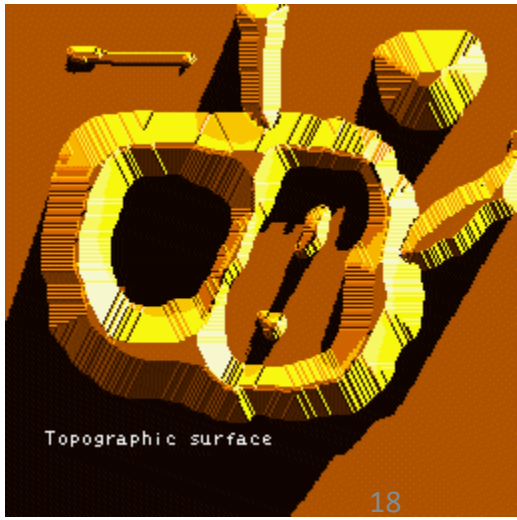
- Watershed features
  - Provides connected segmentation boundaries
  - Provides connected components
- Ground idea
  - A grayscale image can be seen as a topographic surface
    - Intensity as a height value



Watershed

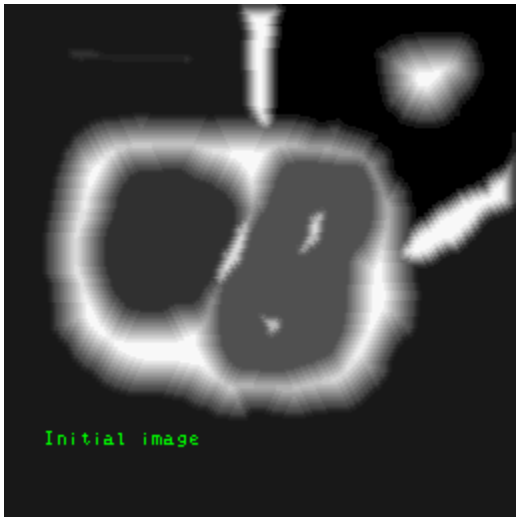
- Three types of points:
  - Local minima
  - Points at which a drop of water would fall to a given minima
  - Points at which a drop of water could fall into two (or more) different minima
    - Watershed lines
- **Goal of the algorithm: find watershed lines**





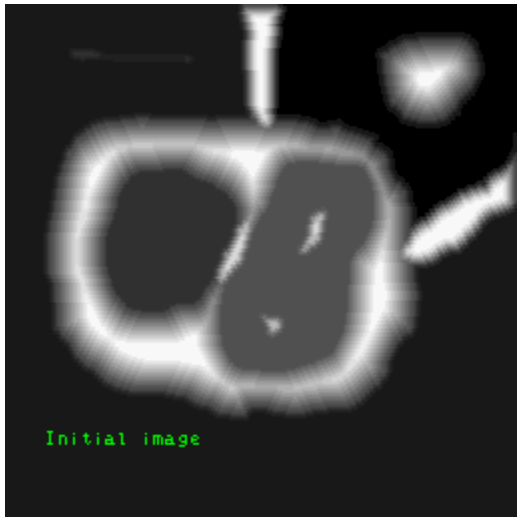
Filling the dam

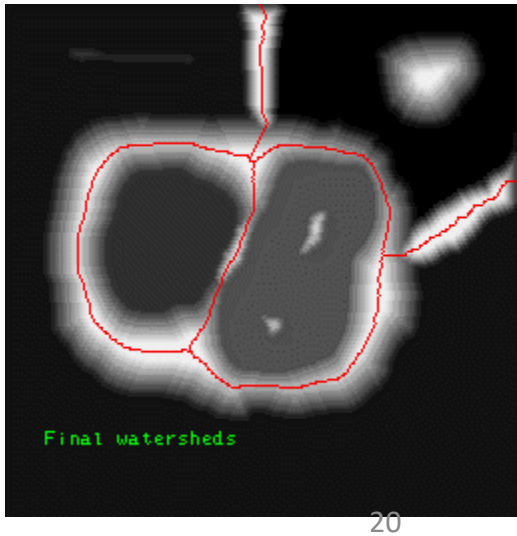
- Flood the surface from the minima
  - Each flooded region is a catchment basin



Filling the dam

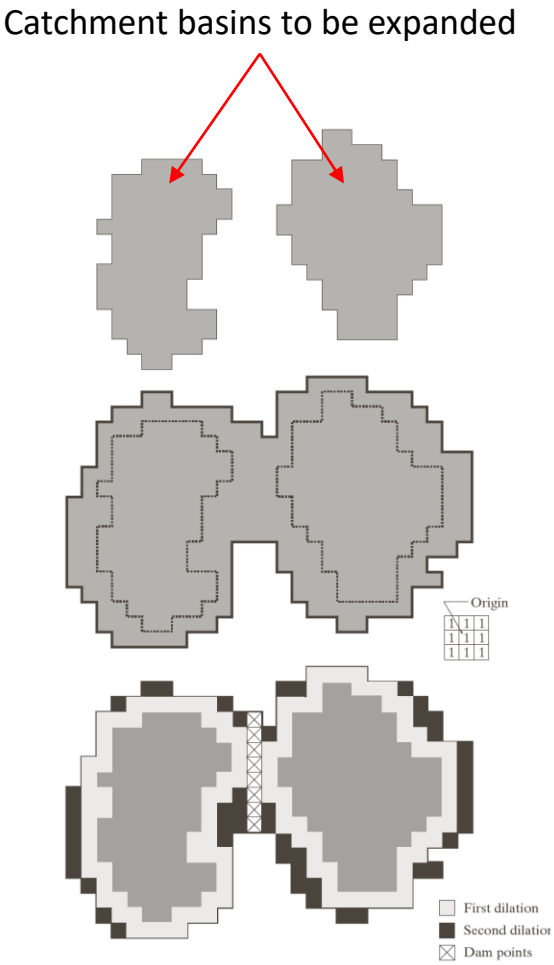
- Prevent merging of water from different basins
  - When two catchment basins merge, build a dam between them
  - The dam is taller than any pixel in the image
  - Dam points define the watershed lines
    - Such lines are the boundary of the segments





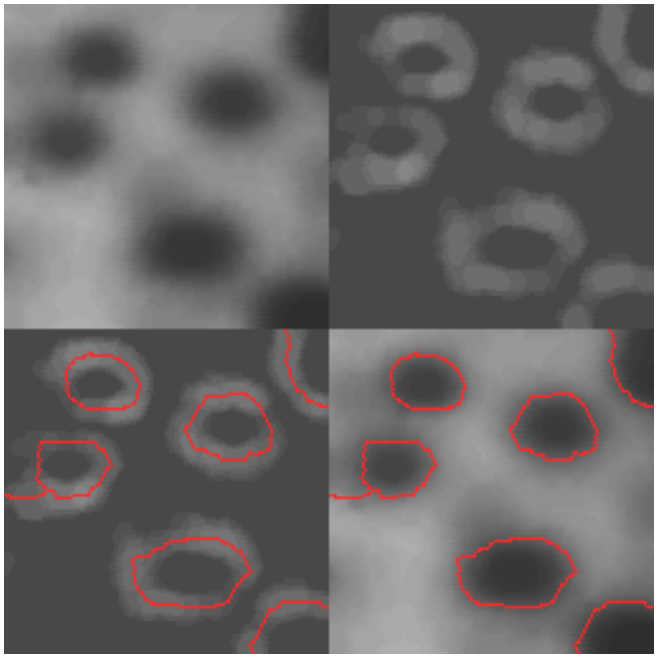
Dam construction

- Recall: when two components merge, we need to build a dam
- **Morphological interpretation**: this can be seen as a set of subsequent dilations
  - Dams created when dilation merges two components



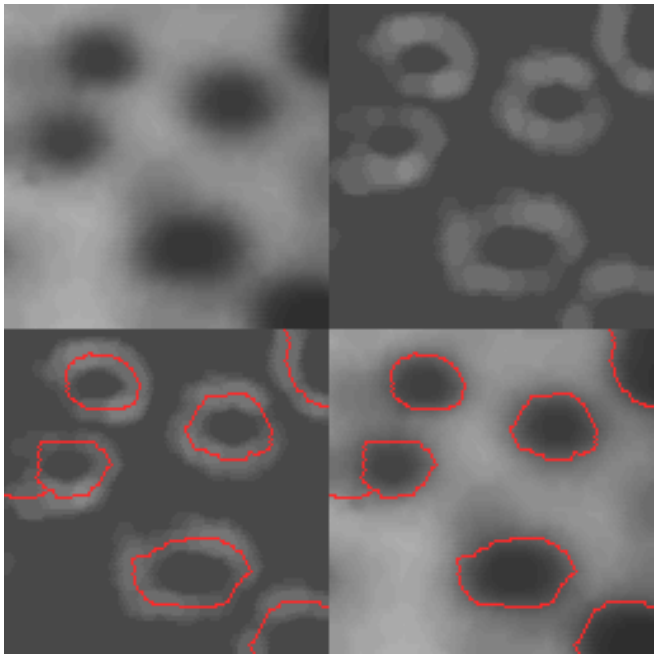
Watershed on gradients

- The watershed algo can be used on the gradient
- Goal: extract uniform regions



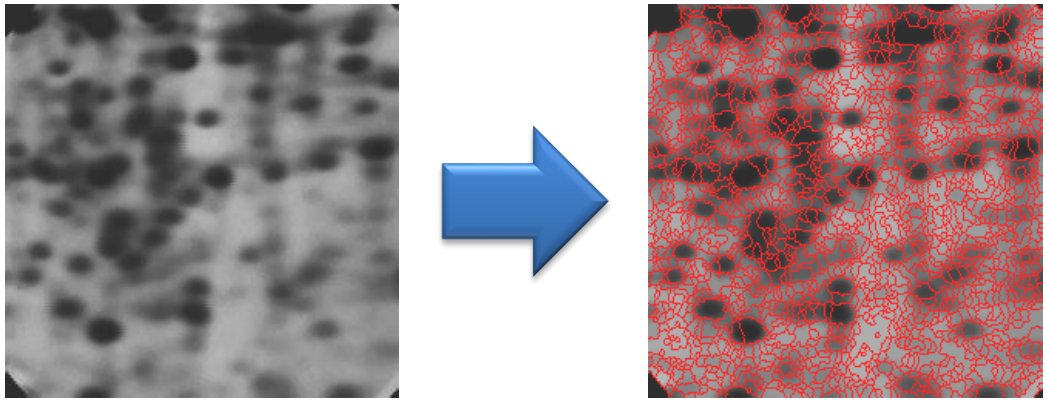
Watershed on gradients

- This is useful when dealing with small gradients
- Small gradients: thick edges
- Catchment basins correspond to **homogeneous graylevel regions**



Oversegmentation

- Direct application often leads to oversegmentation due to noise and irregularities of the image
- Enhancement: flood the topographic surface from a previously defined set of markers

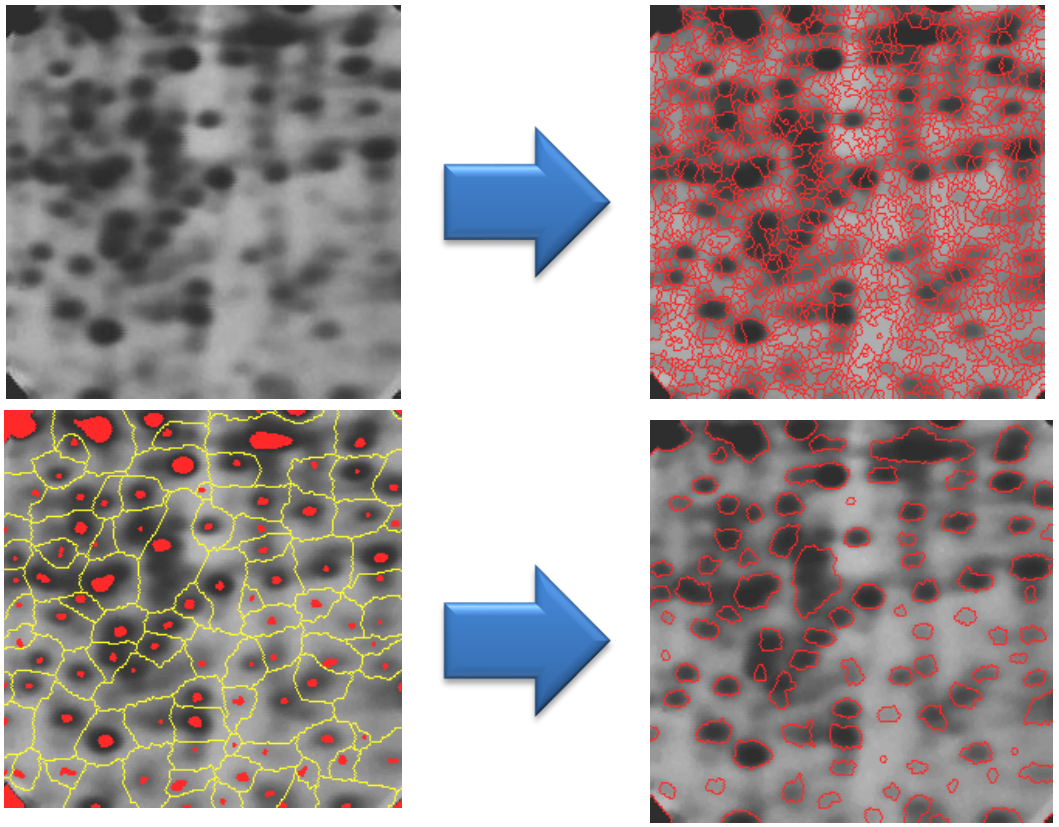


Markers

- Markers can drive the segmentation
- Internal markers: associated with objects of interest
- External markers: background
- Marker selection
  - Preprocessing
  - Criteria definition

Example

- Preprocessing: smoothing
- Internal markers: local minima that are surrounded by lighter points



Watershed in OpenCV

- OpenCV implements a marker-based watershed algorithm
- The valley points to be merged or not are manually selected (i.e., the markers)
- watershed(img,markers)

Implementing a segmentation system

Image segmentation with distance transform and watershed in OpenCV

1. Load the source image and check if it is loaded without any problem, then show it:

```
// Load the image
Mat src = imread(argv[1]);

// Check if everything was fine
if (!src.data)
    return -1;

// Show source image
imshow("Source Image", src);
```

2. Then if we have an image with white background, it is good to tranform it black. This will help us to desciminate the foreground objects easier when we will apply the Distance Transform:

```
// Change the background from white to black, since that will help later to extract
// better results during the use of Distance Transform
for( int x = 0; x < src.rows; x++ ) {
  for( int y = 0; y < src.cols; y++ ) {
      if ( src.at<Vec3b>(x, y) == Vec3b(255,255,255) ) {
        src.at<Vec3b>(x, y)[0] = 0;
        src.at<Vec3b>(x, y)[1] = 0;
        src.at<Vec3b>(x, y)[2] = 0;
      }
    }
}

// Show output image
imshow("Black Background Image", src);
```

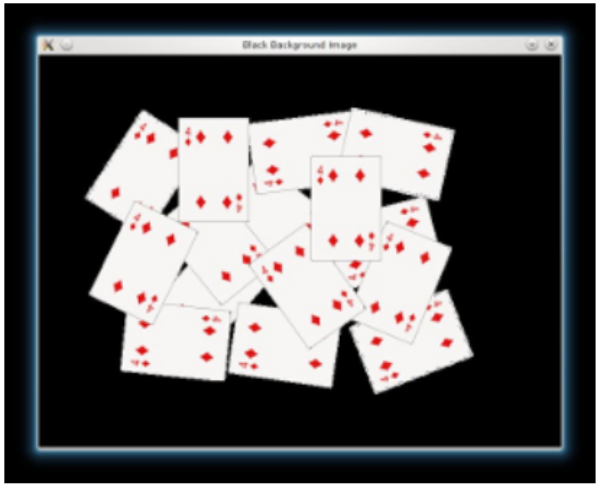


Image segmentation with distance transform and watershed in OpenCV

3. Afterwards we will sharp our image in order to acute the edges of the foreground objects. We will apply a laplacian filter with a quite strong filter (an approximation of second derivative):

```
// Create a kernel that we will use for accuting/sharpening our image
Mat kernel = (Mat_<float>(3,3) <<
        1,  1, 1,
        1, -8, 1,
        1,  1, 1); // an approximation of second derivative, a quite strong kernel

// do the laplacian filtering as it is
// well, we need to convert everything in something more deeper then CV_8U
// because the kernel has some negative values,
// and we can expect in general to have a Laplacian image with negative values
// BUT a 8bits unsigned int (the one we are working with) can contain values from 0 to 255
// so the possible negative number will be truncated
Mat imgLaplacian;
Mat sharp = src; // copy source image to another temporary one
filter2D(sharp, imgLaplacian, CV_32F, kernel);
src.convertTo(sharp, CV_32F);
Mat imgResult = sharp - imgLaplacian;

// convert back to 8bits gray scale
imgResult.convertTo(imgResult, CV_8UC3);
imgLaplacian.convertTo(imgLaplacian, CV_8UC3);

// imshow( "Laplace Filtered Image", imgLaplacian );
imshow( "New Sharped Image", imgResult );
```

Image segmentation with distance transform and watershed in OpenCV

4. Now we tranfrom our new sharped source image to a grayscale and a binary one, respectively:

```
// Create binary image from source image
Mat bw;
cvtColor(src, bw, CV_BGR2GRAY);
threshold(bw, bw, 40, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
imshow("Binary Image", bw);
```

Image segmentation with distance transform and watershed in OpenCV

5. We are ready now to apply the Distance Tranform on the binary image. Moreover, we normalize the output image in order to be able visualize and threshold the result:

```
// Perform the distance transform algorithm
Mat dist;
distanceTransform(bw, dist, CV_DIST_L2, 3);

// Normalize the distance image for range = {0.0, 1.0}
// so we can visualize and threshold it
normalize(dist, dist, 0, 1., NORM_MINMAX);
imshow("Distance Transform Image", dist);
```

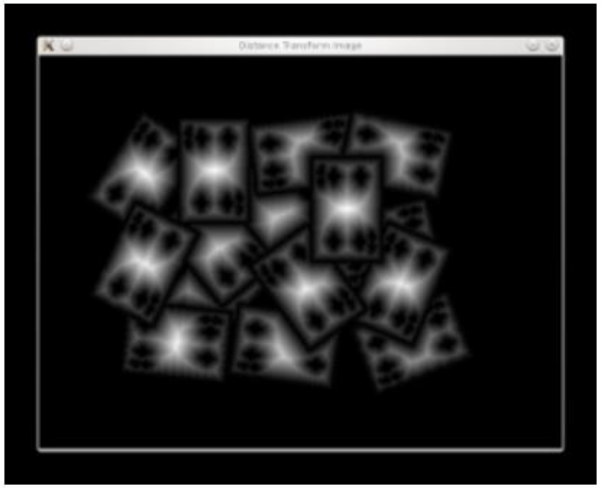


Image segmentation with distance transform and watershed in OpenCV

6. We threshold the *dist* image and then perform some morphology operation (i.e. dilation) in order to extract the peaks from the above image:

```
// Threshold to obtain the peaks
// This will be the markers for the foreground objects
threshold(dist, dist, .4, 1., CV_THRESH_BINARY);

// Dilate a bit the dist image
Mat kernel1 = Mat::ones(3, 3, CV_8UC1);
dilate(dist, dist, kernel1);
imshow("Peaks", dist);
```

Image segmentation with distance transform and watershed in OpenCV

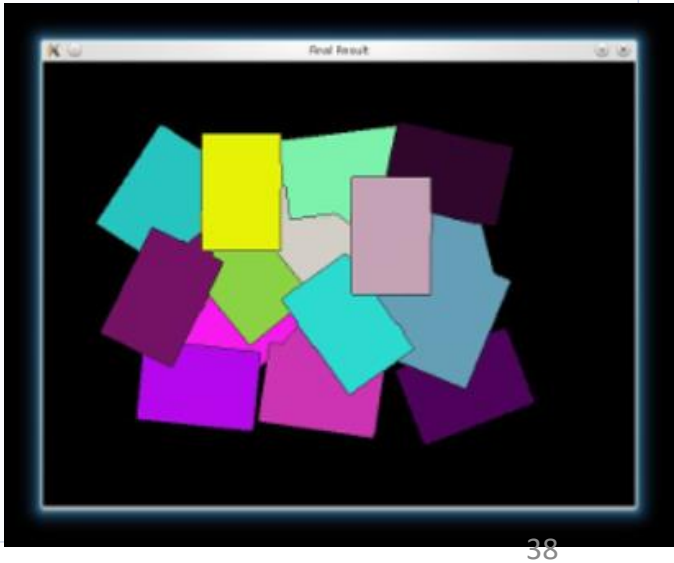
7. From each blob then we create a seed/marker for the watershed algorithm with the help of the **cv::findContours** function:

```
// Create the CV_8U version of the distance image
// It is needed for findContours()
Mat dist_8u;
dist.convertTo(dist_8u, CV_8U);

// Find total markers
vector<vector<Point> > contours;
findContours(dist_8u, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

// Create the marker image for the watershed algorithm
Mat markers = Mat::zeros(dist.size(), CV_32SC1);

// Draw the foreground markers
for (size_t i = 0; i < contours.size(); i++)
    drawContours(markers, contours, static_cast<int>(i), Scalar::all(static_cast<int>(i)+1), -1);

// Draw the background marker
circle(markers, Point(5,5), 3, CV_RGB(255,255,255), -1);
imshow("Markers", markers*10000);
```

Image segmentation with distance transform and watershed in OpenCV

8. Finally, we can apply the watershed algorithm, and visualize the result:

```
    // Perform the watershed algorithm
    watershed(src, markers);

    Mat mark = Mat::zeros(markers.size(), CV_8UC1);
    markers.convertTo(mark, CV_8UC1);
    bitwise_not(mark, mark);
//    imshow("Markers_v2", mark); // uncomment this if you want to see how the mark
                                  // image looks like at that point

    // Generate random colors
    vector<Vec3b> colors;
    for (size_t i = 0; i < contours.size(); i++)
    {
        int b = theRNG().uniform(0, 255);
        int g = theRNG().uniform(0, 255);
        int r = theRNG().uniform(0, 255);

        colors.push_back(Vec3b((uchar)b, (uchar)g, (uchar)r));
    }

    // Create the result image
    Mat dst = Mat::zeros(markers.size(), CV_8UC3);

    // Fill labeled objects with random colors
    for (int i = 0; i < markers.rows; i++)
    {
        for (int j = 0; j < markers.cols; j++)
        {
            int index = markers.at<int>(i,j);
            if (index > 0 && index <= static_cast<int>(contours.size()))
                dst.at<Vec3b>(i,j) = colors[index-1];
            else
                dst.at<Vec3b>(i,j) = Vec3b(0,0,0);
        }
    }

    // Visualize the final image
    imshow("Final Result", dst);
```



UNIVERSITÀ DEGLI STUDI DI PADOVA

Region growing & watershed

Stefano Ghidoni

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS SYSTEMS LAB