Lab5: Morphological Image Processing

Explore the samples folder and try to solve the following exercises.
Functions documentation

C++: void **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int**iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar&**borderValue**=morphologyDefaultBorderValue())¶

- **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or ``CV_64F.
- **dst** – output image of the same size and type as src.
- **element** – structuring element used for erosion; if element=Mat() , a 3 x 3rectangular structuring element is used.
- **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.
- **iterations** – number of times erosion is applied.
- **borderType** – pixel extrapolation method (see **borderInterpolate()** for details).
- **borderValue** – border value in case of a constant border (see**createMorphologyFilter()** for details).

Python: cv2.**erode**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst¶

C++: void **dilate**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int**iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar&**borderValue**=morphologyDefaultBorderValue())¶

- **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or ``CV_64F.
- **dst** – output image of the same size and type as src.
- **element** – structuring element used for dilation; if element=Mat() , a 3 x 3rectangular structuring element is used.
- **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.
- **iterations** – number of times dilation is applied.
- **borderType** – pixel extrapolation method (see **borderInterpolate()** for details).
- **borderValue** – border value in case of a constant border (see**createMorphologyFilter()** for details).

Python: cv.**Dilate**(src, dst, element=None, iterations=1) → None

Returns a structuring element of the specified size and shape for morphological operations.

Mat **getStructuringElement**(int **shape**, Size **ksize**, Point **anchor**=Point(-1,-1))¶

- **shape** –

Element shape that could be one of the following:

  - **MORPH_RECT** - a rectangular structuring element:
$$E_{ij} = 1$$
  - **MORPH_ELLIPSE** - an elliptic structuring element, that is, a filled ellipse inscribed into the rectangle Rect(0, 0, esize.width, 0.esize.height)
  - **MORPH_CROSS** - a cross-shaped structuring element:
$$E_{ij} = \begin{cases} 1 & \text{if } i=\texttt{anchor.y} \text{ or } j=\texttt{anchor.x} \\ 0 & \text{otherwise} \end{cases}$$
  - **CV_SHAPE_CUSTOM** - custom structuring element (OpenCV 1.x API)
- **ksize** – Size of the structuring element.
- **cols** – Width of the structuring element
- **rows** – Height of the structuring element
- **anchor** – Anchor position within the element. The default value $(-1,-1)$ means that the anchor is at the center. Note that only the shape of a cross-shaped element depends on the anchor position. In other cases the anchor just regulates how much the result of the morphological operation is shifted.
-

C++: void **morphologyEx**(InputArray **src**, OutputArray **dst**, int **op**, InputArray **kernel**, Point**anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar&**borderValue**=morphologyDefaultBorderValue())¶

- **src** – Source image. The number of channels can be arbitrary. The depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
- **dst** – Destination image of the same size and type as src .
- **element** – Structuring element.
- **op** –

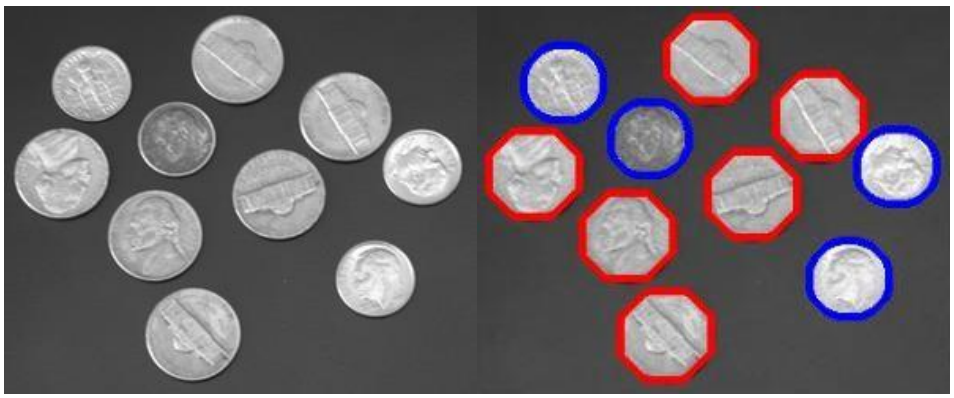
Type of a morphological operation that can be one of the following:

  - **MORPH_OPEN** - an opening operation
  - **MORPH_CLOSE** - a closing operation
  - **MORPH_GRADIENT** - a morphological gradient
  - **MORPH_TOPHAT** - “top hat”
  - **MORPH_BLACKHAT** - “black hat”
  - **MORPH_HITMISS** - “hit and miss”

- **iterations** – Number of times erosion and dilation are applied.
- **borderType** – Pixel extrapolation method. See **borderInterpolate()** for details.
- **borderValue** – Border value in case of a constant border. The default value has a special meaning. See **createMorphologyFilter()** for details.

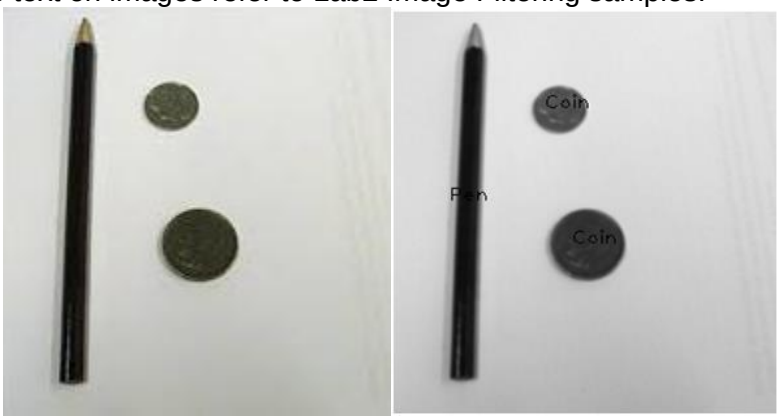
Ex1. Given the image “coins.jpg”, try to use the different morphological operations, to output how much money is in the image -> “coins.jpg”, also highlight different coins with different colors as shown in the output image on the right.
Each big coin -> 100 cent, small coin -> 50 cent.
Hint: you can use the following sequence of operations:

1- You can easily notice that there’s an obvious difference between coins and background colors, so you can use thresholding to differentiate between coins and background, use Otsu method from previous lab to decide the thresholding value.
2- You’ll find some noisy pixels turned white from the background, you can remove it using opening with a small structuring element (small enough in order not to remove the existing coins).
3- You’ll find some black holes inside the coins due to the color variations inside the coin itself (i.e. there’s a coin on the left that has a major color similar to the background), so you’ll need to fill these gaps, you can use closing with a circular structuring element to keep the coins circular shape, or you can use image filling algorithms.
4- Now you have 10 filled circles with different radii (Image A), if you made an opening operation using a relatively high circular structuring element radius, you’ll remove smaller coins and large coins will remain the same (Image B), so you can count the connected components to know how many big coins?
5- Then you can get the image (B-A) to get the smaller coins back again, if you find some parts of larger coins still there, you can use opening to remove them, then you can use connected component analysis again to count how many objects are there (Image C).
6- To highlight different coins with different colors, you can use boundary extraction algorithm to get coins boundaries as follows
    a. From Image B → get the image = dilation(B) – B → Boundary of large coins
    b. From Image C → get the image = dilation(C) – C → Boundary of small coins
7- Back to the original input image, for all pixels lie on boundaries of large coins, color it red, for all pixels lie on boundaries of small coins , color it blue



Ex2. Using similar approach to exercise1 try to make your solution generic to apply on image "Coins2.jpg" as well, note that there're three different coins.

Ex3.Using different morphological operations given the image "Coins&pen.jpg" try to label each object with either coin or pen.
Reminder: to write text on images refer to Lab2 Image Filtering samples.



Ex4.Using simple thresholding and morphological techniques, we need to replace the morning background with a night background of your choice
Hint: use thresholding then some morphological operations then image filling to transform the image into a binary image either part of the background or the foreground, then replace background pixels with any part of the night pixels attached.

One intermediate stage will be the image on the right, but it needs more enhancements to do the required job.

شدوا حيلكم يا جماعة

Ex5. Apply the hit and miss transform in sample #3, to the following image "b.tif", choose a suitable pattern to use to find the "U" letter
Hint:
First you'll need to threshold the image as it's a gray scale image
Then you'll apply hit and miss