

Extract horizontal and vertical lines by using morphological operations

{#tutorial_morph_lines_detection}

@tableofcontents

@prev_tutorial{tutorial_hitOrMiss} @next_tutorial{tutorial_pyramids}

Original author	Theodore Tsismelis
Compatibility	OpenCV >= 3.0

Goal

In this tutorial you will learn how to:

- Apply two very common morphology operators (i.e. Dilation and Erosion), with the creation of custom kernels, in order to extract straight lines on the horizontal and vertical axes. For this purpose, you will use the following OpenCV functions:

- **erode()**
- **dilate()**
- **getStructuringElement()**

in an example where your goal will be to extract the music notes from a music sheet.


Theory

Morphology Operations

Morphology is a set of image processing operations that process images based on predefined *structuring elements* known also as kernels. The value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the kernel, you can construct a morphological operation that is sensitive to specific shapes regarding the input image.

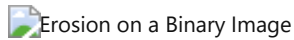
Two of the most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of the object in an image, while erosion does exactly the opposite. The amount of pixels added or removed, respectively depends on the size and shape of the structuring element used to process the image. In general the rules followed from these two operations have as follows:

- **Dilation:** The value of the output pixel is the **maximum** value of all the pixels that fall within the structuring element's size and shape. For example in a binary image, if any of the pixels of the input image falling within the range of the kernel is set to the value 1, the corresponding pixel of the output image will be set to 1 as well. The latter applies to any type of image (e.g. grayscale, bgr, etc).

 Dilation on a Binary Image

 Dilation on a Grayscale Image

- **Erosion:** The vice versa applies for the erosion operation. The value of the output pixel is the *minimum* value of all the pixels that fall within the structuring element's size and shape. Look the at the example figures below:

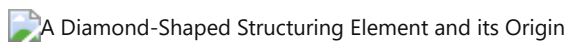


Structuring Elements

As it can be seen above and in general in any morphological operation the structuring element used to probe the input image, is the most important part.

A structuring element is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size. Typically are much smaller than the image being processed, while the pixels with values of 1 define the neighborhood. The center pixel of the structuring element, called the origin, identifies the pixel of interest -- the pixel being processed.

For example, the following illustrates a diamond-shaped structuring element of 7x7 size.



A structuring element can have many common shapes, such as lines, diamonds, disks, periodic lines, and circles and sizes. You typically choose a structuring element the same size and shape as the objects you want to process/extract in the input image. For example, to find lines in an image, create a linear structuring element as you will see later.

Code

This tutorial code's is shown lines below.

@add_toggle_cpp You can also download it from [here](#). @include
samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp @end_toggle

@add_toggle_java You can also download it from [here](#). @include
samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java @end_toggle

@add_toggle_python You can also download it from [here](#). @include
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py @end_toggle

Explanation / Result

Get image from [here](#) .

Load Image

@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp
load_image @end_toggle

@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java
load_image @end_toggle

@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py load_image @end_toggle

Grayscale

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp gray
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java gray
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py gray @end_toggle
```

Grayscale to Binary image

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp bin
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java bin
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py bin @end_toggle
```

Output images

Now we are ready to apply morphological operations in order to extract the horizontal and vertical lines and as a consequence to separate the the music notes from the music sheet, but first let's initialize the output images that we will use for that reason:

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp init
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java init
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py init @end_toggle
```

Structure elements

As we specified in the theory in order to extract the object that we desire, we need to create the corresponding structure element. Since we want to extract the horizontal lines, a corresponding structure element for that purpose will have the following shape: and in the source code this is represented by the following code snippet:

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp horiz
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java horiz
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py horiz @end_toggle
```

The same applies for the vertical lines, with the corresponding structure element: and again this is represented as follows:

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp vert
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java vert
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py vert @end_toggle
```

Refine edges / Result

As you can see we are almost there. However, at that point you will notice that the edges of the notes are a bit rough. For that reason we need to refine the edges in order to obtain a smoother result:

```
@add_toggle_cpp @snippet samples/cpp/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.cpp smooth
@end_toggle
```

```
@add_toggle_java @snippet samples/java/tutorial_code/ImgProc/morph_lines_detection/Morphology_3.java smooth
@end_toggle
```

```
@add_toggle_python @snippet
samples/python/tutorial_code/imgProc/morph_lines_detection/morph_lines_detection.py smooth @end_toggle
```