Computer Vision

Hough Transform
Morphological Operations

Fátima Leal

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UNIVERSIDADE PORTUCALENSE

What we have learnt

- Gradient Detection
  - Sobel Filter
  - Prewitt
  - Laplacian Filter
- Edge detection
  - Noise reduction
  - Detection od edge points
- Counters

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Content

- Hough Transform
- Morphological operations

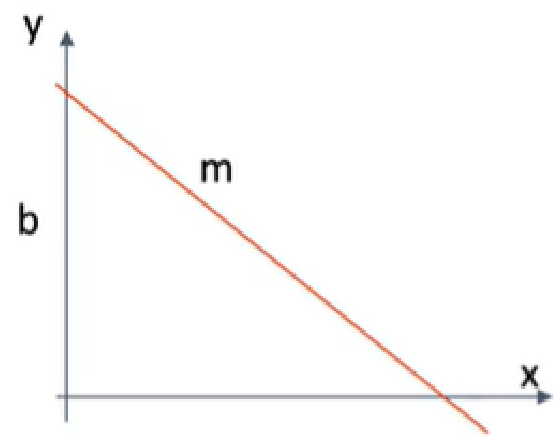
Hough Transform

- Hough transform is a popular technique to detect straight lines

- Normally a straight line is defined as:

$$y = mx + b$$

- Where m is the slope and b is the intercept



UPT DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

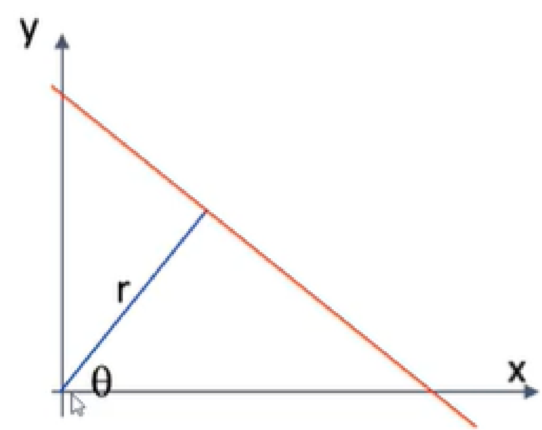
Hough Transform

- The line can also be represented as:

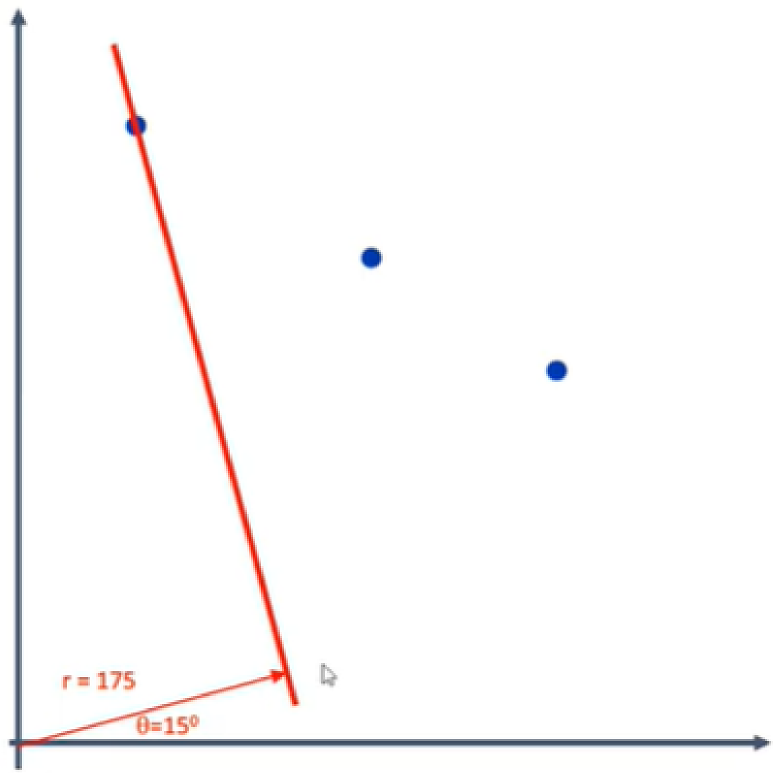
$$r = x\ \cos\theta + y\sin\theta$$

- Where r is the distance from the origin to the closest point on the straight line

- (r, θ) corresponds to the Hough space representation of a line

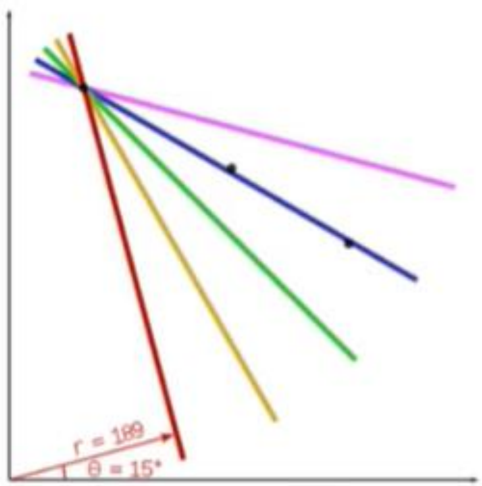


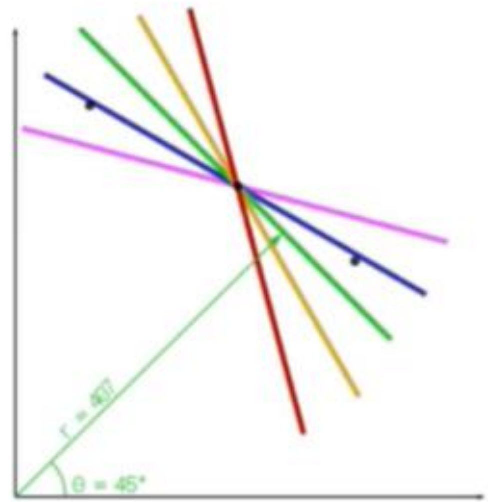
Hough Transform

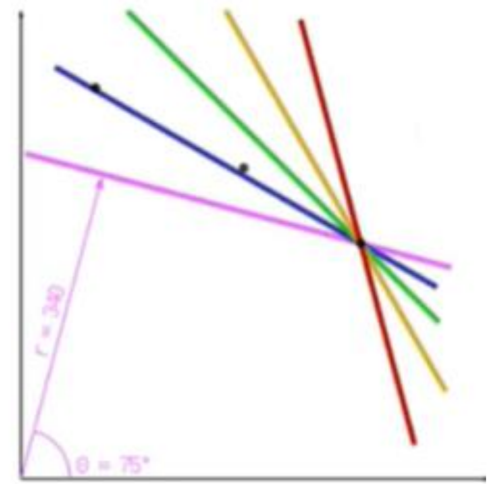


DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Hough Transform





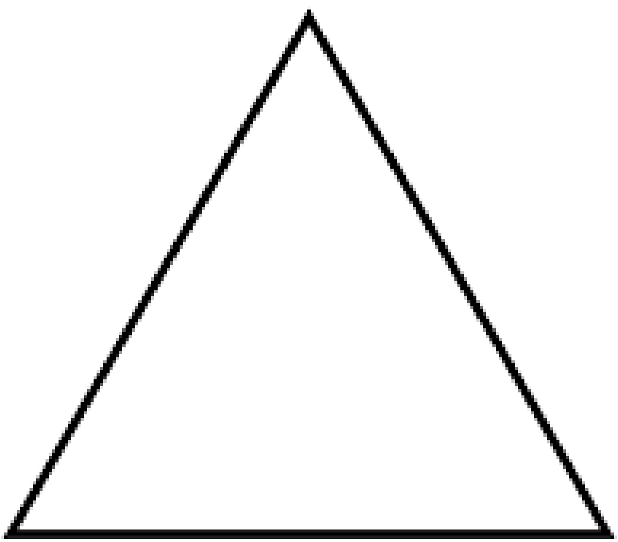


θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4

θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3

θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

Hough Transform



- Using the image triangle.png apply:
  - Gray conversion
  - Gaussian blur
  - Canny to binarize and detect edges
  - Detect lines with Hough transform using HoughLinesP() function. This function returns the coordinates of the points

```
HoughLinesP(image, r, theta,
threshold)
```

  - Draw in the original image in blue colour the detected lines

DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Hough Transform

```
import cv2
import numpy as np

image = cv2.imread("triangle.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
cv2.imshow("Blurred image", blurred)
edged = cv2.Canny(blurred, 50, 150)
cv2.imshow("Edges", edged)


# Detect points that form a line
max_slider = 50
lines = cv2.HoughLinesP(edged, 1, np.pi/180, max_slider)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 1)
# Show result
cv2.imshow("Result Image", image)
cv2.waitKey(0)
```


DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Hough Transform: Use case

Morphological Operations

- Simple operations applied to binary or grayscale images
- Morphological operations are applied to shapes and structures inside of images
  - Increase the size of objects
  - Decrease the size of objects
  - Close gaps or open

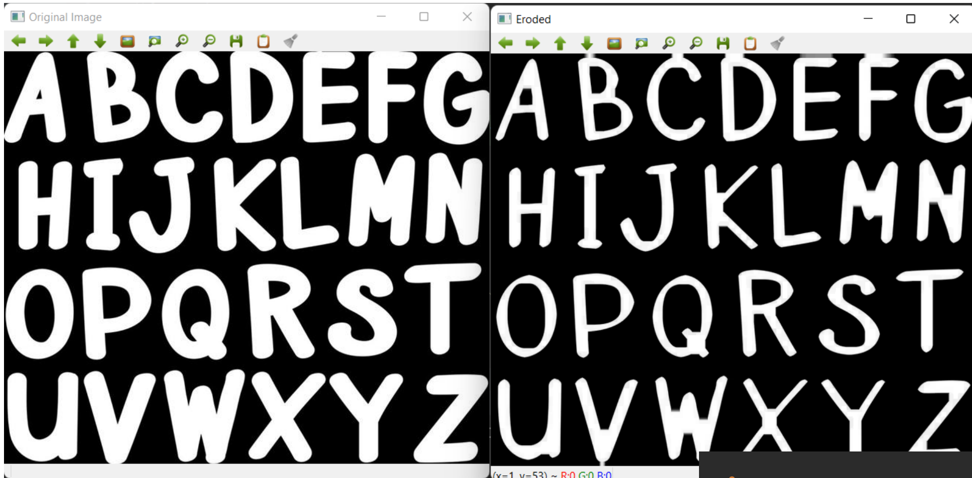
Morphological Operations

- Erosion
- Dilatation
- Opening
- Closing

Morphological Operations: Erosion

- Import the necessary packages
- Read the image
- Binarize the image.
- As it is advised to keep the foreground in white, invert operation on the binarized image to make the foreground as white.
- We are defining a 5×5 kernel filled with ones
- Then we can make use of Opencv erode() function to erode the boundaries of the image.
- All the pixels near boundary will be discarded depending upon the size of kernel.

Morphological Operations: Erosion



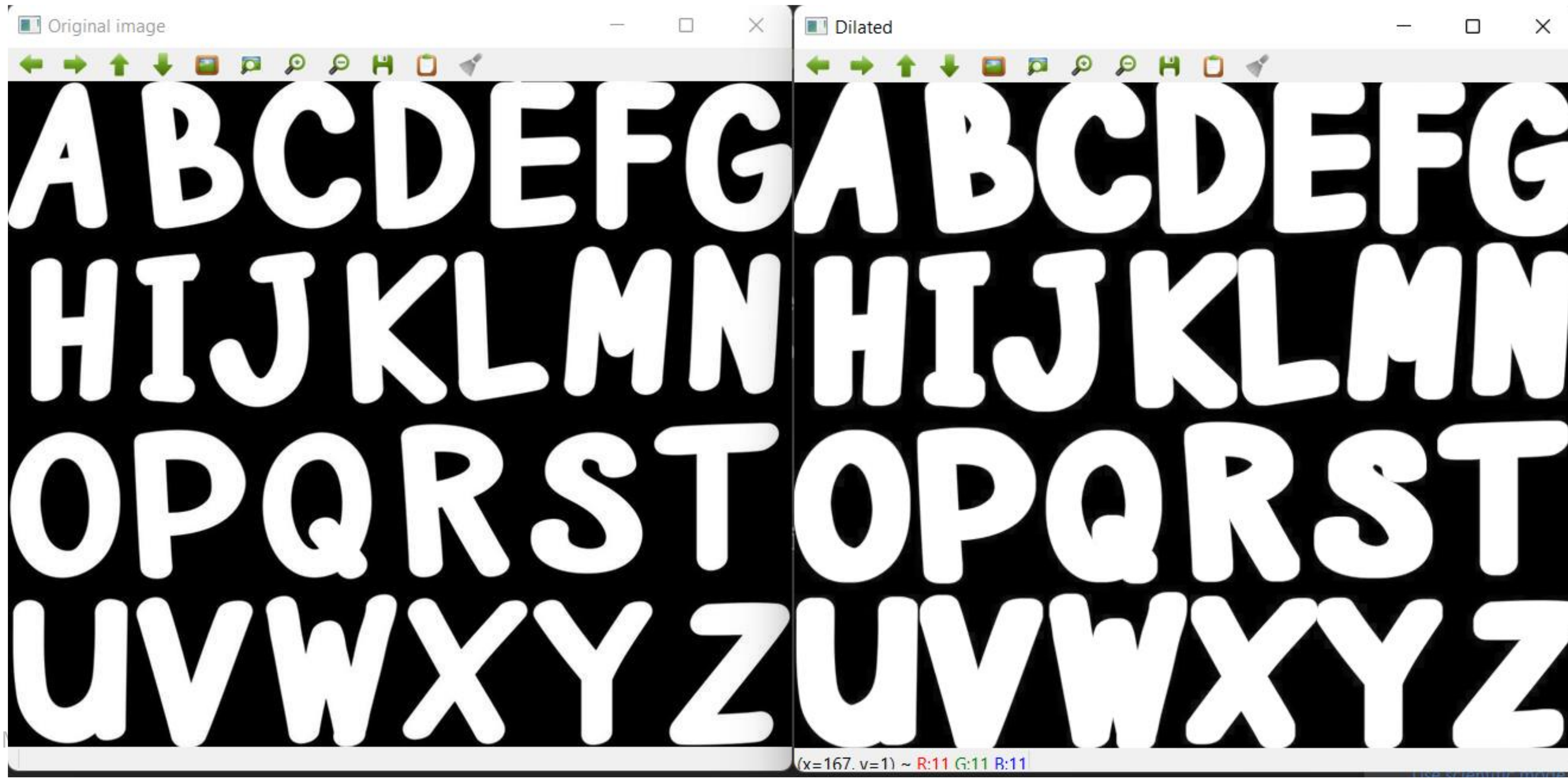

```python
import cv2
import numpy as np

image = cv2.imread("morpho.jpg")
invert = cv2.bitwise_not(image)
# define the kernel
kernel = np.ones((3, 3), np.uint8)

erosion = cv2.erode(invert.copy(), kernel,
iterations=1)
cv2.imshow("Eroded", erosion)
cv2.waitKey(0)
```

Morphological Operations: Dilatation

```
dilatation = cv2.dilate(invert.copy(), kernel, iterations=1)
```



Morphological Operations:opening

- Erosion followed by dilatation
- Useful to remove noise

```
opening = cv2.morphologyEx(invert.copy(), cv2.MORPH_OPEN, kernel)
```

Morphological Operations:closing

- Dilatation followed by erosion
- Useful in closing small holes inside the foreground objects, or small black points on the object.

```
closing = cv2.morphologyEx(invert.copy(), cv2.MORPH_CLOSE, kernel)
```

Morphological Operations

- Morphological gradient: difference between dilatation and erosion

```
gradient = cv2.morphologyEx(invert.copy(), cv2.MORPH_GRADIENT, kernel)
```

- Top Hat: difference between the input image and the opening

```
tophat = cv2.morphologyEx(invert.copy(), cv2.MORPH_TOPHAT, kernel)
```

- Black hat: difference between the input image and the closing

```
blackhat = cv2.morphologyEx(invert.copy(), cv2.MORPH_BLACKHAT, kernel)
```


DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Morphological Operations: Structuring Element

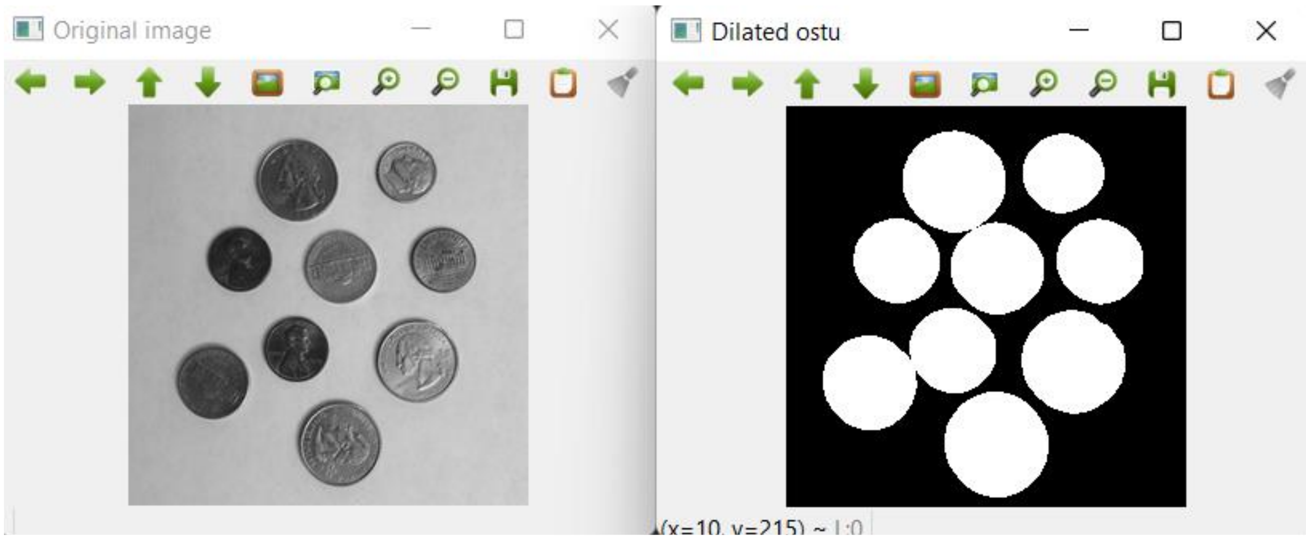
- The previous examples use rectangular kernels with the help of numpy
- Some use cases may need elliptical/circular shaped kernels.
- OpenCV has a function, cv.getStructuringElement(). You just pass the shape and size of the kernel, you get the desired kernel.

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(13,13))
```


DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Morphological Operations: Structuring Element

- Increase the coins using morphological operations with an ellipse kernel



Morphological Operations: Use Case

UPT DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Let’s play with images!

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.