Université de Lille

SHAPE ATTRIBUTES

RECONNAISSANCE DES FORMES

Iacob Sergiu

26th January 2020

CHAPTER 1

INTRODUCTION

1.1 CONTEXT

This report is the result of a university assignment. It aims to prove that the student understood the motivation, goals and means of studying pattern recognition. Therefore, this is a way of summarizing a series of observations and experiments done on images containing shapes.

1.2 MOTIVATION

We want to be able to extract shapes from images and differentiate between them. Using various **shape attributes**, we should be able to conclude which shapes are similar, which are different, which of them are the same but just rotated at different angles, what their main axis of inertia is and so on. Being able to recognise these attributes would allow us to identify and categorise shapes from images. From there on, many posibilities exist: image indexing, searching for certain shapes (e.g. "images with squares") etc.

1.3 GOALS

Given multiple shapes, $S_1, \ldots, S_n$, each shape S_i being represented by its image pixels, find proper **shape indexes** that would allow us to classify these shapes and conclude on a shape's features (e.g. main axis of inertia for S_i). Therefore, we are interested in finding means (that is **shape attributes**, **shape invariants**) to uniquely identify them. The **shape invariants** would help us say that S_i and $S_j, i \neq j$ are the same, even if S_j is rotated at a certain angle, for example.

CHAPTER 2

EXPERIMENTS

2.1 TECHNICAL ACKNOWLEDGEMENTS

The following code presented was written using *Python 3.7*. The images from which the shapes were extracted were used as gray images. 2.1

2.2 LOADING DATA

Images are loaded then converted to grayscale. Right now, we are not interested in their color.

```
def rdfReadGreyImage(nom):
    img = Image.open(f'images/{nom}').convert('L')
    img = np.asarray(img)
    return img
```

LISTING 2.1
Loading images in Python

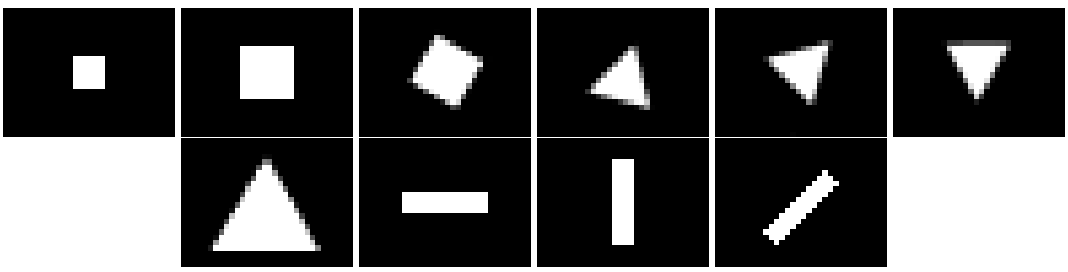

FIGURE 2.1
Shape images

2.3 SHAPE'S MAIN AXIS OF INERTIA

2.3.1 PREREQUISITES

We define the *moment of a shape* as the following:

$$M_{ij} = \sum_x \sum_y x^i y^j A(x, y)$$

where $A(x, y)$ is the value of the pixel (x, y) of the shape's image. One can easily observe that M_{00} is equal to the number of pixels that are not black. We call that the *surface* of the shape.

```
def rdfMoment(img, p, q):
    x = [x ** p for x in range(1, img.shape[0] + 1)]
    y = [y ** q for y in range(1, img.shape[1] + 1)]
    x, y = np.array(x), np.array(y)
    return np.dot(np.dot(x.T, img), y)
```

LISTING 2.2
Calculating the moment of a shape

The *barycentre* is defined as being the following:

$$(\bar{x}, \bar{y}) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}})$$

Having these, we can now make our M_{ij} invariant to the rotation and translation of the shape. We can define the *centered moment* as being:

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y)$$

```
def rdfMomentCentre(img, p, q):
    # Barycentre
    s = rdfSurface(img)
    cx = rdfMoment(img, 1, 0) / s
    cy = rdfMoment(img, 0, 1) / s
    # Initialiser les vecteurs x et y
    x = [(x-cx) ** p for x in range(1, img.shape[0] + 1)]
    y = [(y-cy) ** q for y in range(1, img.shape[1] + 1)]
    # Calcul du moment centre
    return np.dot(np.dot(x, img), y)
```

LISTING 2.3
Calculating centered moments

And for our last trick, we can make μ_{ij} invariant to the scale of the image! We define the *normalized centered moment* as the following:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{1+\frac{i+j}{2}}}$$

```
def rdfMomentCentreNormalise(img, p, q):
    upq = rdfMomentCentre(img, p, q)
    u00 = rdfMomentCentre(img, 0, 0)
    normalised = upq / (u00 ** (1 + (p + q)/2))
    return normalised
```

LISTING 2.4
Calculating normalised centered moments

2.3.2 MAIN AXIS OF INERTIA

Inspiring ourselves from physics, we may use the shape's moment of inertia matrix. With that, we're able to tell which is a figure's main axis of inertia.

```
def mainInertionAxis(img, normalise=True):
    f = rdfMomentCentreNormalise if normalise is True else rdfMomentCentre
    u20 = f(img, 2, 0)
    u11 = f(img, 1, 1)
    u02 = f(img, 0, 2)
    I = np.ndarray(buffer=np.array(
        [u20, u11, u11, u02]), shape=(2, 2), dtype='float')
    _, eigenVectors = np.linalg.eig(I)
    P = eigenVectors.T
    return P
```

LISTING 2.5
Calculating normalised centered moments

Our first experiment is calculating the main axis for the 3 rectangles, in the following order: diagonal, horizontal and vertical. The results clearly show us that we can easily differentiate the fact that the shapes are rotated.

```
Diagonal:
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
Horizontal:
[[1. 0.]
 [0. 1.]]
Vertical:
[[1. 0.]
 [0. 1.]]
```

Let us now analyse the square images. From the results, we can see that the main axis of inertia for the squares that only differ in size is the same. For the square that is rotated, we get different results. Therefore, calculating the main axis of inertia is invariant to the scale of the shape.

```
Small square:
[[1. 0.]
 [0. 1.]]
Big square:
[[1. 0.]
 [0. 1.]]
Big square, rotated:
[[ 0.95469358  0.29759059]
 [-0.29759059  0.95469358]]
```

CHAPTER 3

RESULTS

CHAPTER 4

DISCUSSION

CHAPTER 5

CONCLUSION