Shape Matching using Hu Moments (C++/Python)

Satya Mallick Kritika Rapat

DECEMBER 10, 2018 — 15 COMMENTS

how-to OpenCV 3 OpenCV 4 Shape Analysis Tutorial

In this post, we will show how to use Hu Moments for shape matching. You will learn the following

1. What are image moments?
2. How are image moments calculated?
3. What are Hu moment invariants (or Hu Moments)?
4. How to calculate Hu Moments for an image using OpenCV?
5. How can Hu Moments be used for finding similarity between two shapes.

Let's dive into the details

1. What are Image Moments?

Image moments are a weighted average of image pixel intensities. Let's pick a simple example to understand the previous statement.

For simplicity, let us consider a single channel binary image I. The pixel intensity at location (x, y) is given by $I(x, y)$. Note for a binary image $I(x, y)$ can take a value of 0 or 1.

The simplest kind of moment we can define is given below

$$M = \sum_x \sum_y I(x, y) \tag{1}$$

All we are doing in the above equation is calculating the sum of all pixel intensities. In other words, all pixel intensities are weighted only based on their intensity, but not based on their location in the image.

For a binary image, the above moment can be interpreted in a few different ways

1. It is the number of white pixels (i.e. intensity = 1).
2. It is area of white region in the image.

So far you may not be impressed with image moments, but here is something interesting. Figure 1 contains three binary images — S (S0.png), rotated S (S5.png), and K (K0.png).

This image moment for S and rotated S will be very close, and the moment for K will be different.

For two shapes to be the same, the above image moment will necessarily be the same, but it is not a sufficient condition. We can easily construct two images where the above moment is the same, but they look very different.

2. How are image moments calculated?

Let's look at some more complex moments.

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y) \tag{2}$$

where i and j are integers (e.g. 0, 1, 2 ....). These moments are often referred to as **raw moments** to distinguish them from **central moments** mentioned later in this article.

Note the above moments depend on the intensity of pixels and their location in the image. So intuitively these moments are capturing some notion of shape.

TL;DR : Image moments capture information about the shape of a blob in a binary image because they contain information about the intensity $I(x, y)$, as well as position x and y of the pixels.

Centroid using Image Moments

The centroid of a binary blob is simply its center of mass. The centroid $(\bar{x}, \bar{y})$ is calculated using the following formula.

$$\begin{aligned} \bar{x} &= \frac{M_{10}}{M_{00}} \\ \bar{y} &= \frac{M_{01}}{M_{00}} \end{aligned} \tag{3}$$

We have explained this in a greater detail in our previous post.

2.1 Central Moments

Central moments are very similar to the raw image moments we saw earlier, except that we subtract off the centroid from the x and y in the moment formula.

$$\mu_{ij} = \sum_{x} \sum_{y} (x - \bar{x})^{i} (y - \bar{y})^{j} I(x, y) \tag{4}$$

Notice that the above central moments are **translation invariant**. In other words, no matter where the blob is in the image, if the shape is the same, the moments will be the same.

Won't it be cool if we could also make the moment invariant to **scale**? Well, for that we need **normalized central moments** as shown below.

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{(i+j)/2+1}} \tag{5}$$

TL;DR : **Central moments** are translations invariant, and **normalized central moments** are both translation and scale invariant.

3. What are Hu Moments?

It is great that central moments are translation invariant. But that is not enough for shape matching. We would like to calculate moments that are invariant to translation, scale, and rotation as shown in the Figure below.

Fortunately, we can in fact calculate such moments and they are called Hu Moments.

Definition

Hu Moments (or rather Hu moment invariants) are a set of 7 numbers calculated using central moments that are invariant to image transformations. The first 6 moments have been proved to be invariant to **translation**, **scale**, and **rotation**, and **reflection**. While the 7th moment's sign changes for image reflection.

The 7 moments are calculated using the following formulae :

$$\begin{aligned} h_0 &= \eta_{20} + \eta_{02} \\ h_1 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\ h_2 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\ h_3 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\ h_4 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ h_5 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})] \\ h_6 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} \end{aligned} \tag{6}$$

Please refer to this paper if you are interested in understanding the theoretical foundation of Hu Moments.

4. How to calculate Hu Moments in OpenCV?

Next, we will show how to use OpenCV's built-in functions

Fortunately, we don't need to do all the calculations in OpenCV as we have a utility function for Hu Moments. In OpenCV, we use `HuMoments()` to calculate the Hu Moments of the shapes present in the input image.

Let us discuss step by step approach for calculation of Hu Moments in OpenCV.

Read in image as Grayscale

First, we read an image as a grayscale image. This can be done in a single line in Python or C++.

Python

```
1 | # Read image as grayscale image
```

```
im = cv2.imread(filename,cv2.IMREAD_GRAYSCALE
```

C++

```
// Read image as grayscale image
Mat im = imread(filename,IMREAD_GRAYSCALE);
```

Binarize the image using thresholding : Since our data is simply white characters on a black background, we threshold the grayscale image to binary:

Python

```
# Threshold image
_,im = cv2.threshold(im, 128, 255, cv2.THRESH_BINARY)
```

C++

```
// Threshold image
threshold(im, im, 128, 255, THRESH_BINARY);
```

Calculate Hu Moments

OpenCV has a built-in function for calculating Hu Moments. Not surprisingly it is called **HuMoments**. It takes as input the central moments of the image which can be calculated using the function **moments**

Python

```
# Calculate Moments
moments = cv2.moments(im)
# Calculate Hu Moments
huMoments = cv2.HuMoments(moments)
```

C++

```
// Calculate Moments
Moments moments = moments(im, false);
// Calculate Hu Moments
double huMoments[7];
HuMoments(moments, huMoments);
```

Log Transform

The Hu Moments obtained in the previous step have a large range. For example, the 7 Hu Moments of K (K0.png) shown above

h[0] = 0.00162663
h[1] = 3.11619e-07
h[2] = 3.61005e-10
h[3] = 1.44485e-10
h[4] = -2.55279e-20
h[5] = -7.57625e-14
h[6] = 2.09098e-20

Note that hu[0] is not comparable in magnitude as hu[6]. We can use use a log transform given below to bring them in the same range

$$H_i = -\text{sign}(h_i) \log |h_i| \tag{7}$$

After the above transformation, the moments are of comparable scale

H[0] = 2.78871
H[1] = 6.50638
H[2] = 9.44249
H[3] = 9.84018
H[4] = -19.593
H[5] = -13.1205
H[6] = 19.6797

The code for log scale transform is shown below.

Python

```
# Log scale hu moments
for i in range(0,7):
    huMoments[i] = -1* copysign(1.0, huMoments[i]) * log10(abs(huMoments[i]))
```

C++

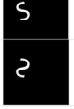
```
// Log scale hu moments
for(int i = 0; i < 7; i++) {
    huMoments[i] = -1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i]));
}
```

5. Shape Matching using Hu Moments

As mentioned earlier, all 7 Hu Moments are invariant under translations (move in x or y direction), scale and rotation. If one shape is the mirror image of the other, the seventh Hu Moment flips in sign. Isn't that beautiful?

Let's look at an example. In the table below we have 6 images and their Hu Moments.

id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797

S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

As you can see, the image K0.png is simply the letter K, and S0.png is the letter S. Next, we have moved the letter S in S1.png, and moved + scaled it in S2.png. We added some rotation to make S3.png and further flipped the image to make S4.png.

Notice that all the Hu Moments for S0, S1, S2, S3, and S4 are close to each other in value except the sign of last Hu moment of S4 is flipped. Also, note that they are all very different from K0.

5.1 Distance between two shapes using matchShapes

In this section, we will learn how to use Hu Moments to find the distance between two shapes. If the distance is small, the shapes are close in appearance and if the distance is large, the shapes are farther apart in appearance.

OpenCV provides an easy to use a utility function called **matchShapes** that takes in two images (or contours) and finds the distance between them using Hu Moments. So, you do not have to explicitly calculate the Hu Moments. Simply binarize the images and use matchShapes.

The usage is shown below.

Python

```
d1 = cv2.matchShapes(im1,im2,cv2.CONTOURS_MATCH_I1,0)
d2 = cv2.matchShapes(im1,im2,cv2.CONTOURS_MATCH_I2,0)
d3 = cv2.matchShapes(im1,im2,cv2.CONTOURS_MATCH_I3,0)
```

C++

```
double d1 = matchShapes(im1, im2, CONTOURS_MATCH_I1, 0);
double d2 = matchShapes(im1, im2, CONTOURS_MATCH_I2, 0);
double d3 = matchShapes(im1, im2, CONTOURS_MATCH_I3, 0);
```

Note that there are three kinds of distances that you can use via a third parameter (CONTOURS_MATCH_I1, CONTOURS_MATCH_I2 or CONTOURS_MATCH_I3).

Two images (im1 and im2) are similar if the above distances are small. You can use any distance measure. They usually produce similar results. **I personally prefer d2**.

Let's see how these three distances are defined.

Let $D(A, B)$ be the distance between shapes A and B, and H_i^A and H_i^B be the i^{th} log transformed Hu Moments for shapes A and B. The distances corresponding to the three cases is defined as

1. **CONTOURS_MATCH_I1**

$$D(A, B) = \sum_{i=0}^{6} \left| \frac{1}{H_i^B} - \frac{1}{H_i^A} \right| \tag{8}$$

2. **CONTOURS_MATCH_I2**

$$D(A, B) = \sum_{i=0}^{6} \left| H_i^B - H_i^A \right| \tag{9}$$

3. **CONTOURS_MATCH_I3**

$$D(A, B) = \sum_{i=0}^{6} \frac{\left| H_i^A - H_i^B \right|}{\left| H_i^A \right|} \tag{10}$$

When we use the shape matching on the images : S0, K0 and S4 (transformed and flipped version of S0), we get the following output : Shape Distances Between ----------
S0.png and S0.png : 0.0
S0.png and K0.png : 0.10783054664091285
S0.png and S4.png : 0.008484870268973932 5.2 Custom distance measure In case you want to define you own custom distance measure between two shapes, you can easily do so. For example, you may want to use the Euclidean distance between the Hu Moments given by

$$D(A, B) = \sqrt{\sum_{i=0}^{6} \left(H_i^B - H_i^A \right)^2} \tag{11}$$

First, you calculate log transformed Hu Moments as mentioned in the previous section, and then calculate the distance yourself instead of using **matchShapes**. **Acknowledgements** : Code for this post was jointly written by **Krutika Bapat** and **Vishwesh Shrimali**

Subscribe & Download Code

If you liked this article and would like to download code (C++ and Python) and example images used in this post, please click here. Alternately, sign up to receive a free Computer Vision Resource Guide. In our newsletter, we share OpenCV tutorials and examples written in C++/Python, and Computer Vision and Machine Learning algorithms and news.

Download Example Code

Tags: Comparison | Hu Moments | Moment Invariants | moments | rotation | Scaling | Translation

LOAD COMMENTS