CS 342: Computer Vision

Project 2: Harris Corner Detection And SIFT Descriptor

Sourav Bhattacharjee
Sawan Aich
Ramakrishna Mission Vivekananda Educational and Research Institute
Department of computer science
MSc, Big Data Analytics

Table of Contents

1. What are features of an image?

Features are basically specific patterns in an image which are unique, can be easily tracked and can be easily compared. If we go for a definition of such a feature, we may find it difficult to express it in words, but we know what they are. Characteristics of good features are

Repeatability

– The same feature can be found in several images despite geometric and photometric transformations

• Saliency

– Each feature is distinctive

• Compactness and efficiency

– Many fewer features than image pixels

• Locality

– A feature occupies a relatively small area of the image; robust to clutter and occlusion

Corners are considered to be good features in an image, because whenever you move a patch around it,the intensity changes in both direction,so it looks different, which means it is uniquely identifiable.

2. How to detect corners?

By now we know that corners are regions in the image with large variation in intensity in all the directions,which is considered to be good features of images. Now the question is how to find one. One early attempt to find these corners was done by Chris Harris and Mike Stephens in their paper A Combined Corner and Edge Detector in 1988, so now it is called Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions. This is expressed as below:

$$E(u,v) = \sum_{x,y} w(x,y).(I(x+u,y+v) - I(x,y))^2$$

where w(x,y) is window function, I(x+u,y+v) is shifted intensity and I(x,y) is the intensity. Window function is either a rectangular window or gaussian window which gives weights to pixels underneath.We have to maximize this function E(u,v) for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to above equation and using some mathematical steps (please refer any standard text books you like for full derivation), we get the final equation as:

$$E(u,v) \approx [u \;\; v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

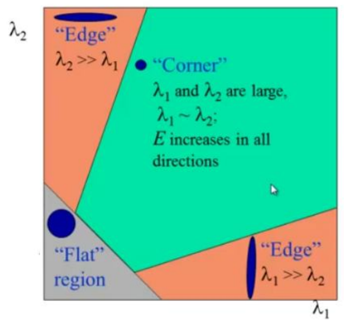
where

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Here, I_x and I_y are image derivatives in x and y directions respectively.
Now we calculate the response function,which basically gives a score that determines if a window can contain a corner or not.

$$R = det(M) - k(trace(M))^2$$

where
• det(M)= $\lambda_1 \lambda_2$
• trace(M) = $\lambda_1 + \lambda_2$
• λ_1 and λ_2 are eigenvalues of M.
So the values of these eigen values decide whether a region is corner, edge or flat.
• When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
•When $R < 0$ which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
•When R is large, which happens when λ_1 and λ_2 are large and λ_1 similar to λ_2, the region is a corner.
It can be represented in a nice picture as follows:



So the result of Harris Corner Detection is a grayscale image with these scores.

3. Harris corner detector code explanation

In the *harris_corners*() function, we take the image and perform Gaussian blur on the image in order to detect the corners. Then we calculate the gradient I_x (along x-axis) and I_y (along y-axis). Then the derivatives are squared and the second order moment matrix M is formed using the derivatives. From there, the determinant and the trace of the matrix is calculated which gives us the value of R. If the value of R is greater than the pre-defined threshold, then we can get the corners.

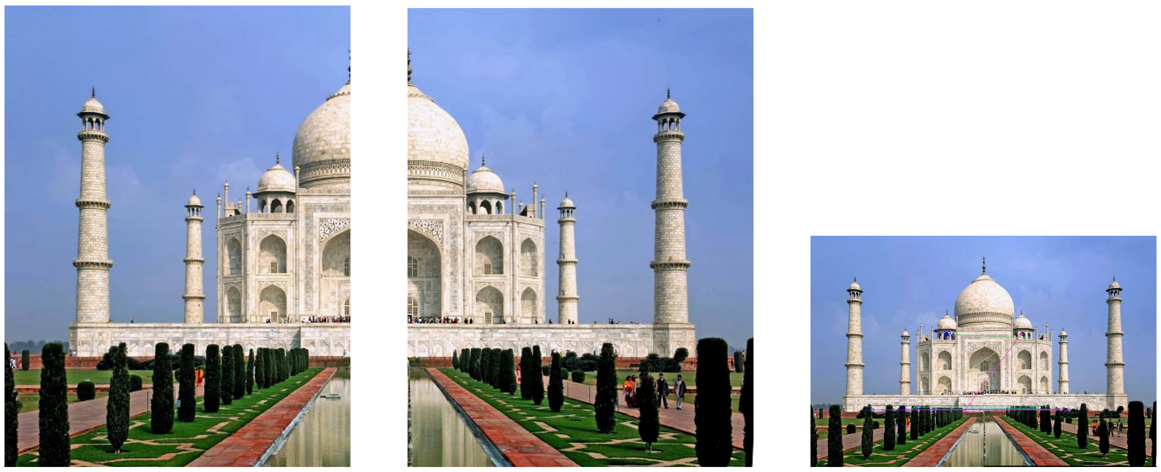


Fig. 1. Harris corner detection on Taj Mahal

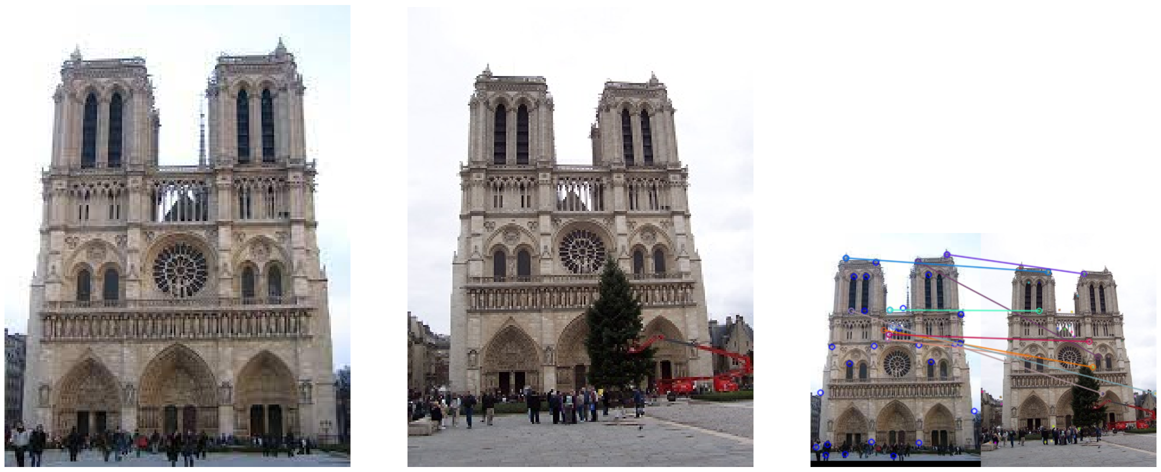


Fig. 2. Harris corner detection on Notredame

4. What is SIFT?

Matching features across different images in a common problem in computer vision. When all images are similar in nature (same scale, orientation, etc) simple corner detectors can work. But when you have images of different scales and rotations, you need to use the Scale Invariant Feature Transform(SIFT).

4.1 Keypoint orientation

The idea is to collect gradient directions and magnitudes around each keypoint. Then we figure out the most prominent orientation(s) in that region. And we assign this orientation(s) to the keypoint.

Any later calculations are done relative to this orientation. This ensures rotation invariance.The size of the ”orientation collection region” around the keypoint depends on it’s scale. The bigger the scale, the bigger the collection region.
Gradient magnitudes and orientations are calculated using these formulae:

$$m(x,y) = \sqrt{(l(x+1,y) - l(x-1,y))^2 + (l(x,y+1) - l(x,y-1))^2}$$
$$\theta(x,y) = \tan^{-1}((l(x,y+1) - l(x,y-1))/(l(x+1,y) - l(x-1,y)))$$

The magnitude and orientation is calculated for all pixels around the keypoint. Then, A histogram is created for this.

In this histogram, the 360 degrees of orientation are broken into 36 bins (each 10 degrees). Lets say the gradient direction at a certain point (in the ”orientation collection region”) is 18.759 degrees, then it will go into the 10-19 degree bin. And the ”amount” that is added to the bin is proportional to the magnitude of gradient at that point.

Once you’ve done this for all pixels around the keypoint, the histogram will have a peak at some point.

Also, any peaks above 80 percent of the highest peak are converted into a new keypoint. This new keypoint has the same location and scale as the original. But it’s orientation is equal to the other peak.

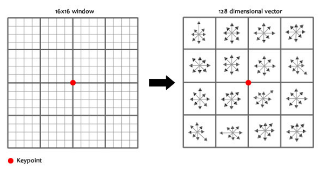
So, orientation can split up one keypoint into multiple keypoints.

To assign an orientation we use a histogram and a small region around it. Using the histogram, the most prominent gradient orientation(s) are identified. If there is only one peak, it is assigned to the keypoint. If there are multiple peaks above the 80 percent mark, they are all converted into a new keypoint (with their respective orientations).

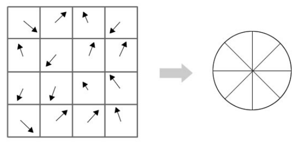
4.2 Generating features

We want to generate a very unique fingerprint for the keypoint. It should be easy to calculate. We also want it to be relatively lenient when it is being compared against other keypoints. Things are never EXACTLY same when comparing two different images.

To do this, a 16x16 window around the keypoint. This 16x16 window is broken into sixteen 4x4 windows.



Within each 4x4 window, gradient magnitudes and orientations are calculated. These orientations are put into an 8 bin histogram.



orientation in the range 0-44 degrees add to the first bin. 45-89 add to the next bin. And so on.And (as always) the amount added to the bin depends on the magnitude of the gradient.

Unlike the past, the amount added also depends on the distance from the keypoint. So gradients that are far away from the keypoint will add smaller values to the histogram.

This is done using a ”gaussian weighting function”. This function simply generates a gradient (it’s like a 2D bell curve). You multiple it with the magnitude of orientations, and you get a weighted thingy. The farther away, the lesser the magnutide.

5. SIFT examples

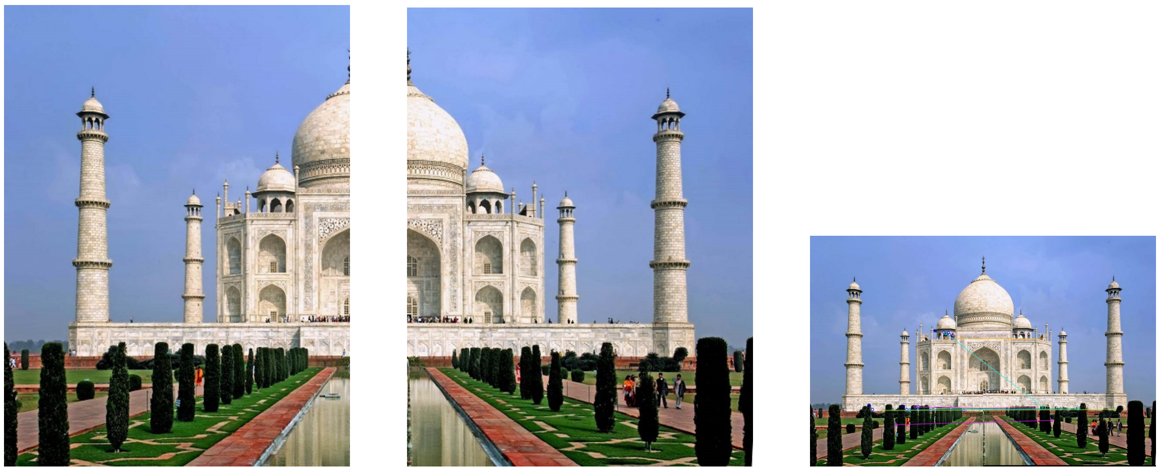


Fig. 3. SIFT on Taj Mahal

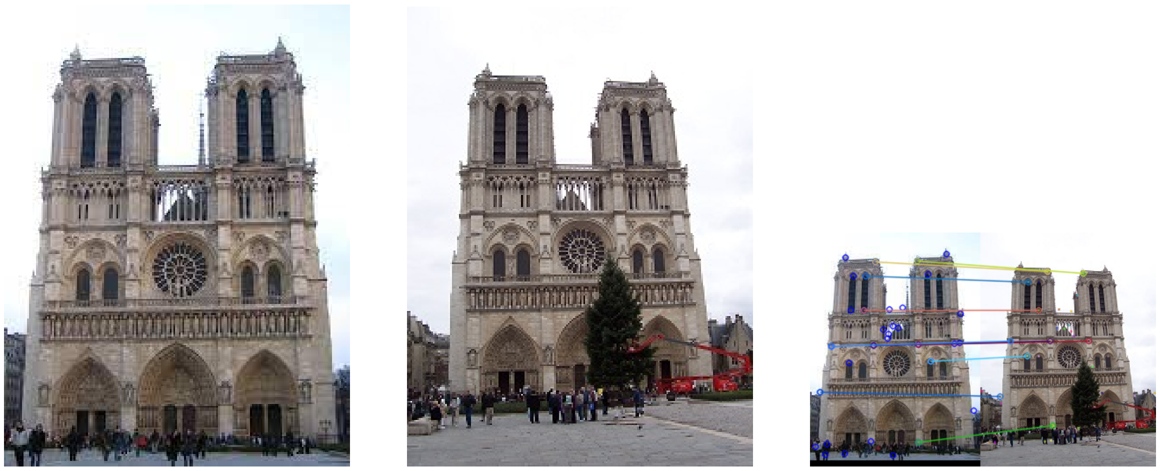


Fig. 4. SIFT on Notredame