COMPUTER VISION

PROJECT-2

DETECTING HARRIS CORNERS AND MATCHING IMAGES

Aritra Raut , Sourjya Chatterjee
April 22, 2021

1 INTRODUCTION :

Nowadays,Computer-Vision is heavily used in every unit of society , especially in the field of security (like identifying any object or any person in a video).So to this we have to match some special features in each frame of the video . And we can easily understand that matching with respect to the corners in any two images is easier that any other points on those images.So in this report we will discuss the procedures to identify those corners and extract special features with the help of which we can match two images(or two frames of a video).

2 ALGORITHM :

2.1 Harris Corner Detection :

We took the image ,shown below-

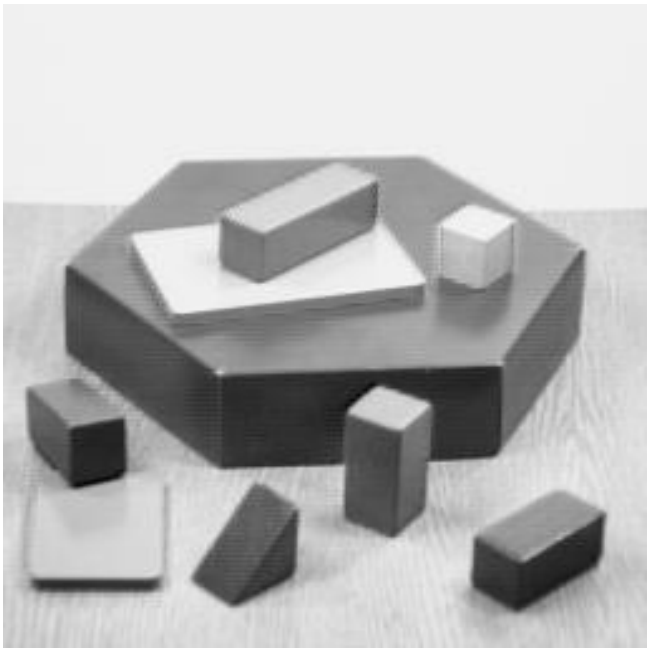
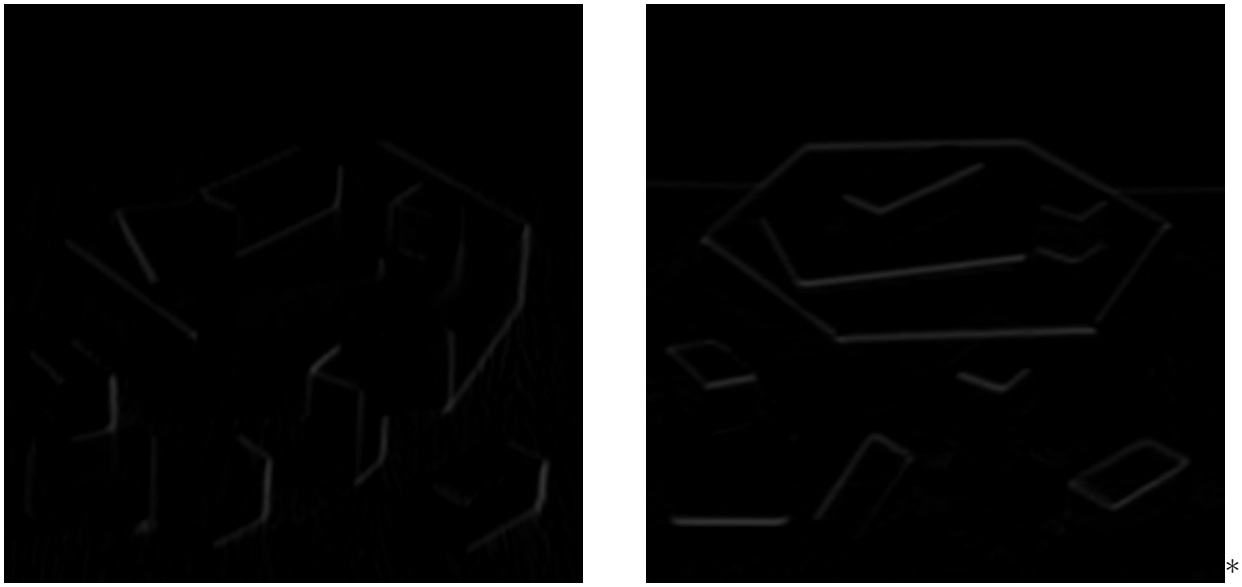


Figure 1: Main Image

Now,detailed algorithm is described below-
Key idea behind this is to identify those points at which the image has two or more gradient changes.
1. At first we applied convolution with a gaussian filter on that image to remove all the gaussian noise which can mislead us.

2.Then we have taken image derivatives with respect to x(Ix)and y(Iy) axis (We have used sobel filters to do this), which will help me at the next step.And image derivatives are as shown below-

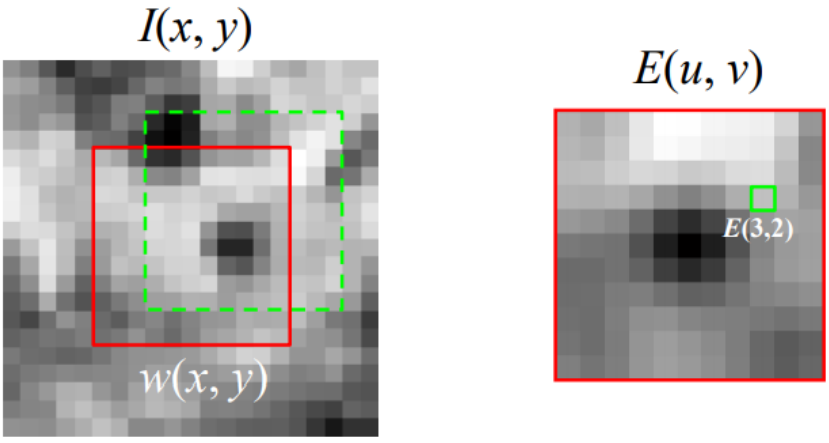


(a) derivative wrt x-axis(Ix) (b) derivative wrt y-axis(Iy)

Figure 2: Image gradients

3. Now, to do our desired work , we have to detect the change in appearance of window w(x,y) for the shift [u,v] (Here we will use a Gaussian window) :

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{1}$$



Now basically we want to find out how this function behaves for small shifts.But calculating it naively is a very slow method , so we recall Taylor series expansion.
Local quadratic approximation of E(u,v) in the neighbourhood of (0,0) is given by the second order Taylor expansion :

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + (1/2) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2}$$

Now,

$$E_u(u,v) = \sum_{x,y} 2w(x,y)[I(x+u, y+v) - I(x,y)]I_x(x+u, y+v) \tag{3}$$

$$E_v(u,v) = \sum_{x,y} 2w(x,y)[I(x+u,y+v) - I(x,y)]I_y(x+u,y+v) \tag{4}$$

$$\begin{aligned} E_{uu}(u,v) = &\sum_{x,y} 2w(x,y)I_x(x+u,y+v)I_x(x+u,y+v) \\ &+ \sum_{x,y} 2w(x,y)[I(x+u,y+v) - I(x,y)]I_{xx}(x+u,y+v) \end{aligned} \tag{5}$$

$$\begin{aligned} E_{vv}(u,v) = &\sum_{x,y} 2w(x,y)I_y(x+u,y+v)I_y(x+u,y+v) \\ &+ \sum_{x,y} 2w(x,y)[I(x+u,y+v) - I(x,y)]I_{yy}(x+u,y+v) \end{aligned} \tag{6}$$

$$\begin{aligned} E_{uv}(u,v) = &\sum_{x,y} 2w(x,y)I_y(x+u,y+v)I_x(x+u,y+v) \\ &+ \sum_{x,y} 2w(x,y)[I(x+u,y+v) - I(x,y)]I_{xy}(x+u,y+v) \end{aligned} \tag{7}$$

So, we can easily check that,

$$E_u(0,0) = E_v(0,0) = 0 \tag{8}$$

$$E_{uu}(0,0) = \sum_{x,y} 2w(x,y)I_x(x,y)I_x(x,y) \tag{9}$$

$$E_{vv}(0,0) = \sum_{x,y} 2w(x,y)I_y(x,y)I_y(x,y) \tag{10}$$

$$E_{uv}(0,0) = \sum_{x,y} 2w(x,y)I_x(x,y)I_y(x,y) \tag{11}$$

The quadratic approximation simplifies to -

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{12}$$

And here our desired M matrix is -

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} Ix * Ix & Ix * Iy \\ Ix * Iy & Iy * Iy \end{bmatrix} = \sum_{x,y} w(x,y) \begin{bmatrix} Ix^2 & Ix * Iy \\ Ix * Iy & Iy^2 \end{bmatrix} \tag{13}$$

4. We can say that this M matrix determines the gradient direction at every point on that image,so , we will try find those points at which both the eigenvalues of this M matrix are large . So to achieve this, we have calculated the R score of the whole image at every position as,

$$\begin{aligned} R &= det(M) - \alpha * trace(M)^2 \\ &= ((Ix^2 * Iy^2) - (Ix * Iy)^2) - \alpha * (Ix^2 + Iy^2) \end{aligned} \tag{14}$$

Where, α usually lies between (0.04-0.06) and as we know that det(M)=product of eigenvalues and trace(M)=sum of eigenvalues.

5. Now we have taken only those points whose R value is higher than a certain threshold*Max(R).so, our list of corners are -

$$Corners = R[R > Threshold * Max(R)] \tag{15}$$

6. Non maximum suppression : Now we have run a window (of size approximately 10*10)over the whole image and taken the points with highest R value within the window, in our final corner-points list.

Now , we have our final list of corner points, and after marking the corners on that image, the image looks like-

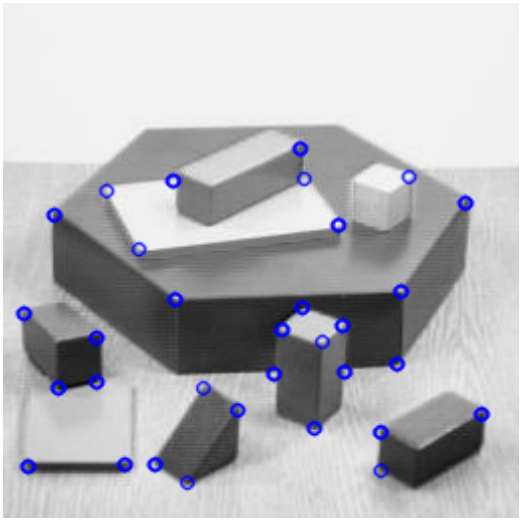


Figure 3: Corner points marked

Now ,to check if our harris corner detection algorithm is working properly or not, we have used some library functions and tried to match different images of the same object , taken from different perspectives.

2.1.1 Matching Results(Using library functions) :

1.

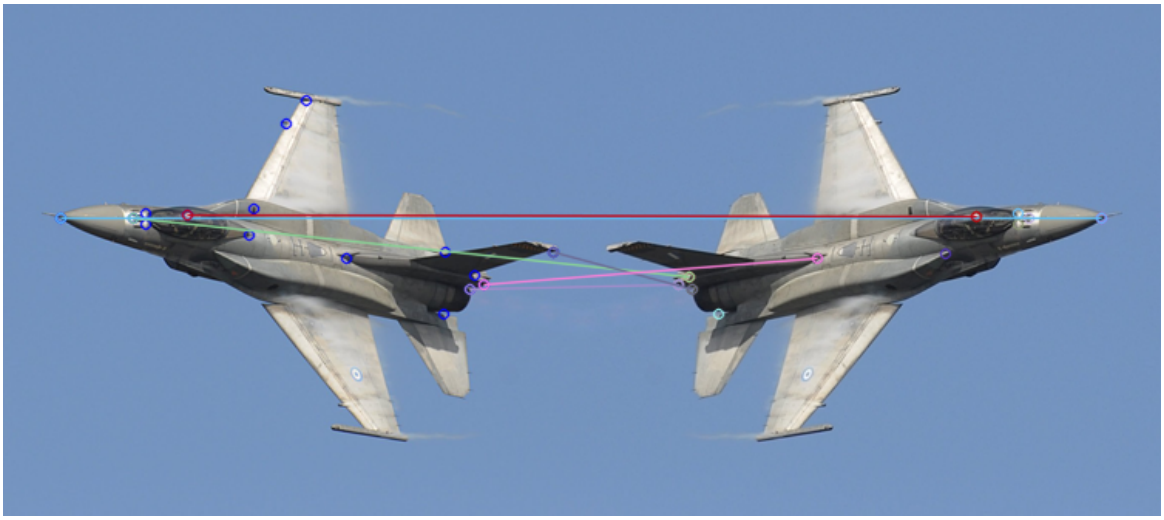


Figure 4: Matching of two images of a plane

2.

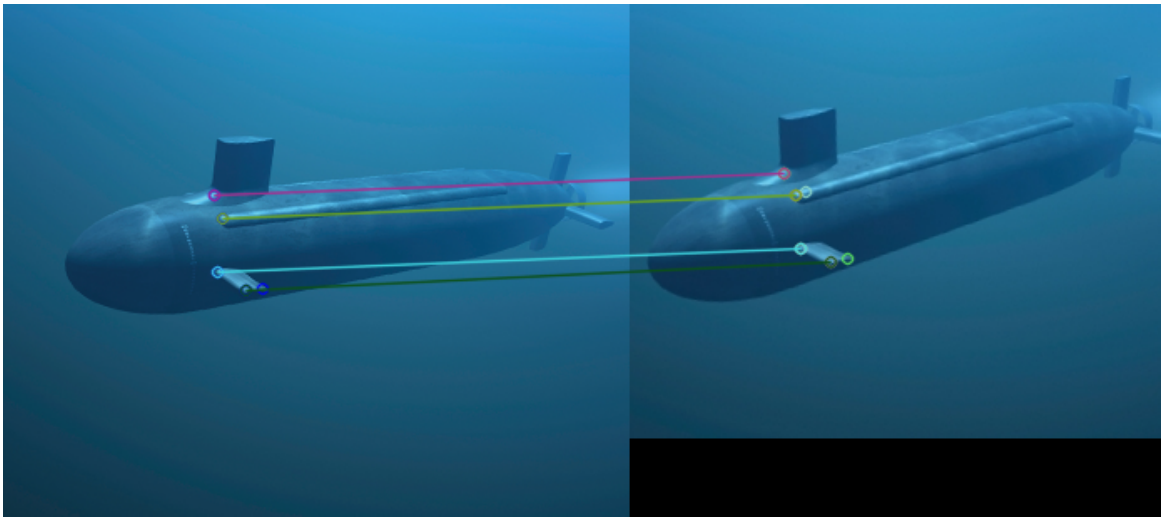


Figure 5: Matching of two images of a submarine

3.

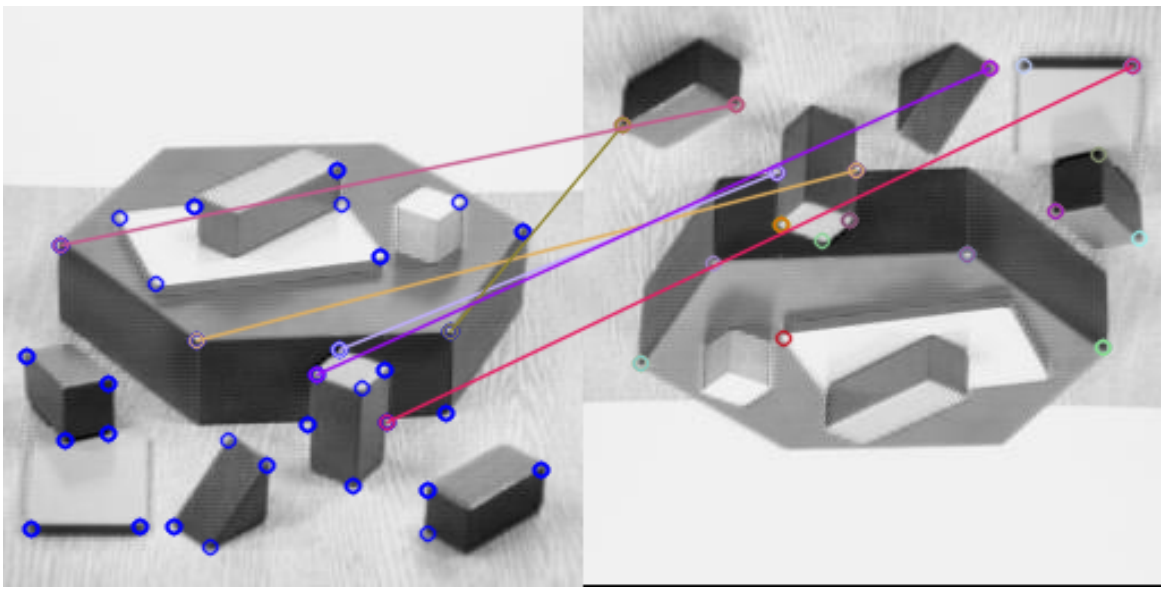


Figure 6: Matching of our experimental image and flipped version of it

2.2 Feature Matching:

Key idea of this section is to extract a feature vector around every key-point.This process is divided into two steps-

2.2.1 Keypoint Orientation :

After the previous step we have legitimate key points.The next thing is to assign an orientation to each keypoint. This orientation provides rotation invariance . 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. Gradient magnitudes and orientations are calculated using these formulae:

$$m(x,y) = \sqrt{Iy\_window^2 + Ix\_window^2} \tag{16}$$

$$\theta(x,y) = tan^{-1}(Iy\_window/Ix\_window) \tag{17}$$

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

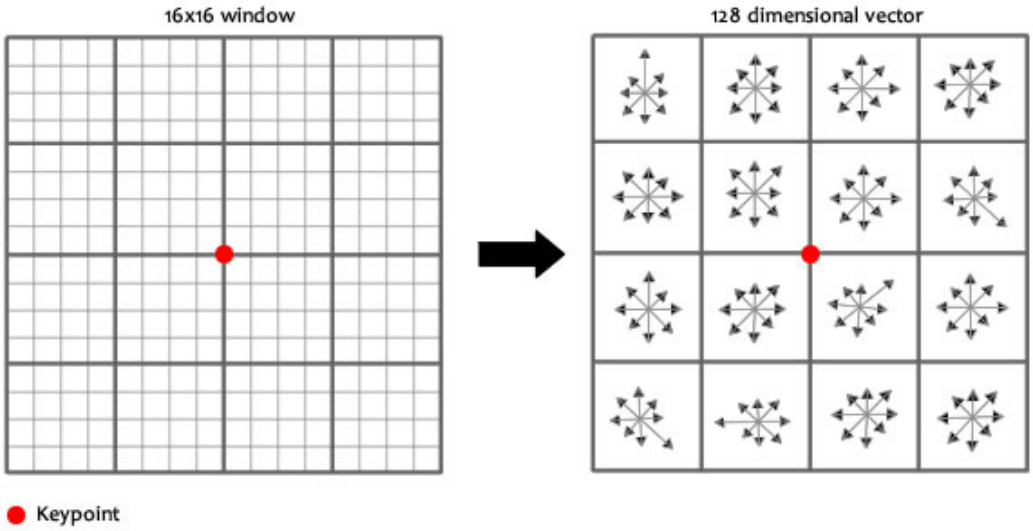
In this histogram, the 360 degrees of orientation are broken into 36 bins (each 10 degrees). Lets say the gradient direction at a certain point (in the "orientation collection region") is 18.759 degrees, then it will go into the 10-19 degree bin. And the "amount" that is added to the bin is proportional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

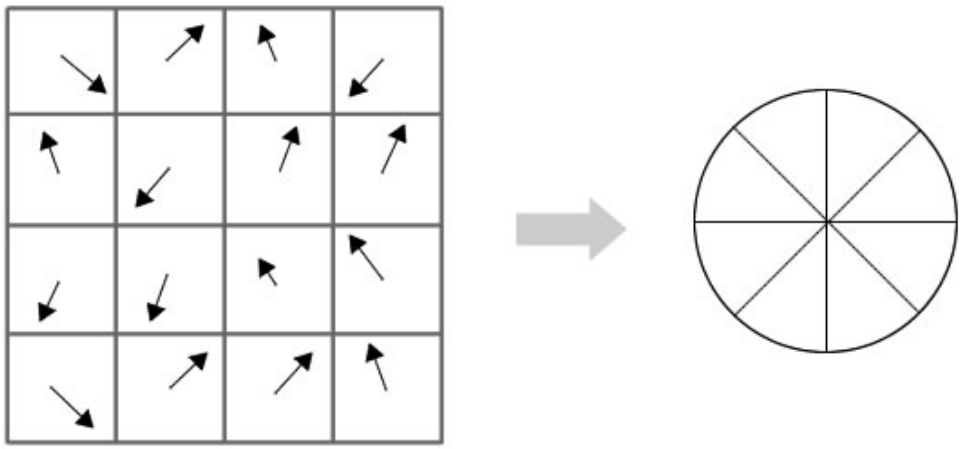
2.2.2 Generating a Feature :

Now we create a fingerprint for each keypoint. This is to identify a keypoint. If an eye is a keypoint, then using this fingerprint, we'll be able to distinguish it from other keypoints, like

ears, noses, fingers, etc. 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



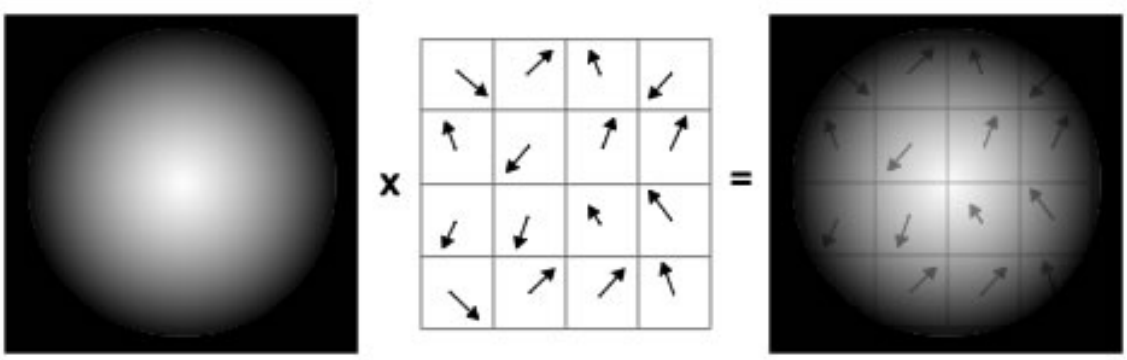
calculated. These orientations are put into an 8 bin histogram. orientation in the range 0-44



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.

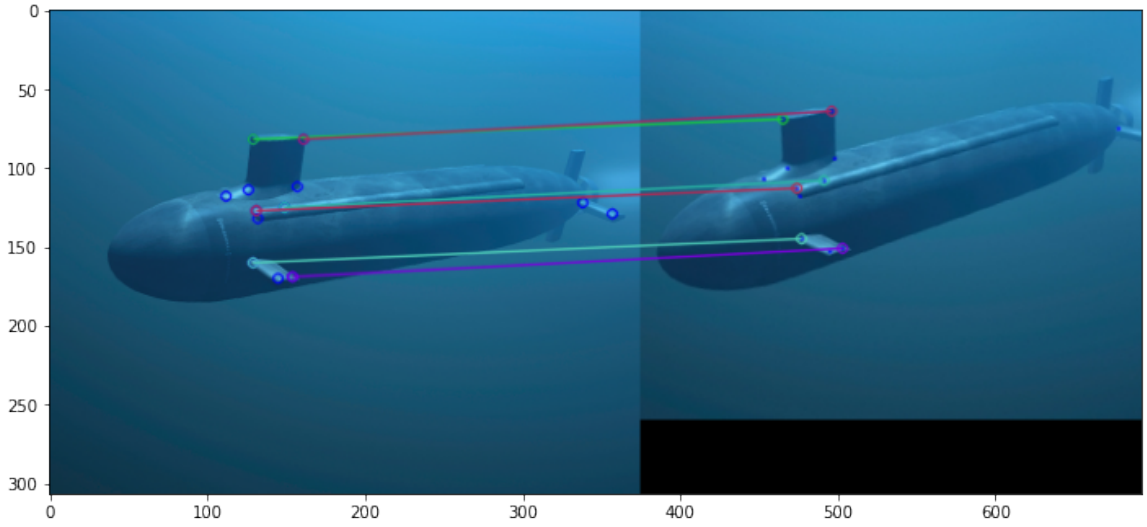
This is done using a "gaussian weighting function". This function simply generates a gradient (it's like a 2D bell curve). You multiple it with the magnitude of orientations, and you get a weighted thingy. The farther away, the lesser the magnitude.



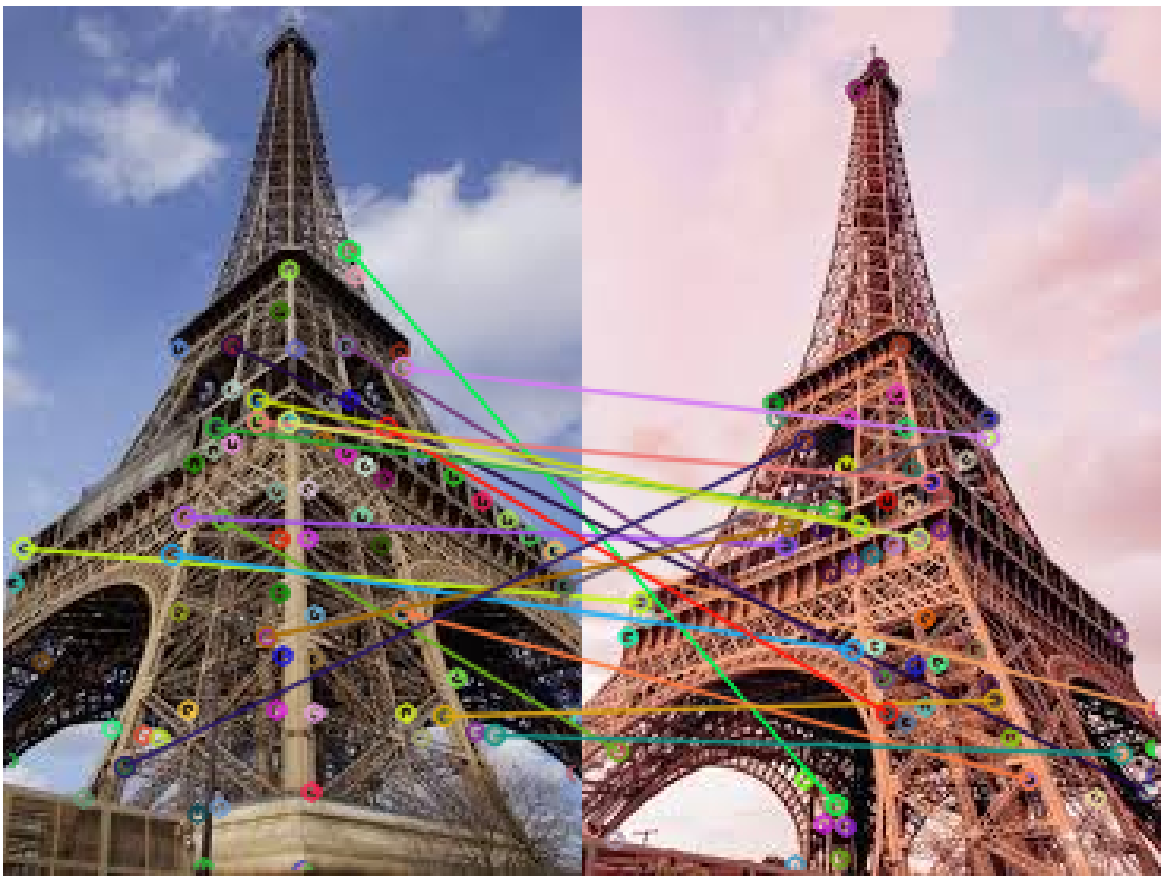
Doing this for all 16 pixels, you would've "compiled" 16 totally random orientations into 8 predetermined bins. You do this for all sixteen 4x4 regions. So you end up with 4x4x8 = 128 numbers. Once you have all 128 numbers, you normalize them (just like you would normalize a vector in school, divide by root of sum of squares). These 128 numbers form the "feature vector". This keypoint is uniquely identified by this feature vector. And one more thing to add the feature vector uses gradient orientations. Clearly, if you rotate the image, everything changes. All gradient orientations also change. To achieve rotation independence, the keypoint's rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint's orientation.

2.3 Results :

We have applied our algorithm on various pictures and the results are given below :
1.



2.



2.4 Conclusion :

As we can see our algorithm is doing pretty good. Though it hasn't marked all the corners properly but we are quite satisfied with the result.Probably the orientation of few corner points are same though they are not actually same,and as we have taken harris corners as our keypoints so we didn't used scale invariance , these may be the reasons of our algorithm's failure.