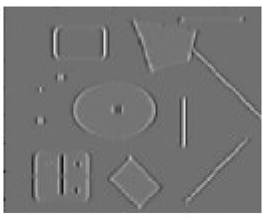
Min Project 2

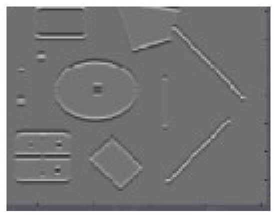
Feature Detection and Matching

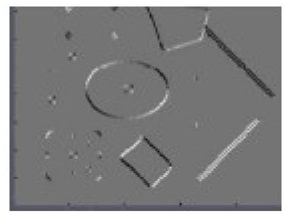
made By:
Omar Yasser
Mohammad Saleh

Interest Point Detection:

Harries Corner Detection:







$$I_x \Leftrightarrow \frac{\partial I}{\partial x} \qquad I_y \Leftrightarrow \frac{\partial I}{\partial y} \qquad I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

1. Compute the Harris matrix over a window:

$$H = \sum_{(u,v)} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \qquad I_x = \frac{\partial f}{\partial x}, I_y = \frac{\partial f}{\partial y}$$

Typically Gaussian weights

2. Compute the determinate and the trace from that:

$$H = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

$$R = det(M) - \alpha \cdot trace(M)^2$$

SIFT features Descriptors:

Within each 4×4 window, gradient magnitudes and orientations are calculated. These orientations are put into an 8 bin histogram. the amount added to the bin depends on the magnitude of the gradient, also depends on the distance from the key point So gradients that are far away from the key point will add smaller values to the histogram.

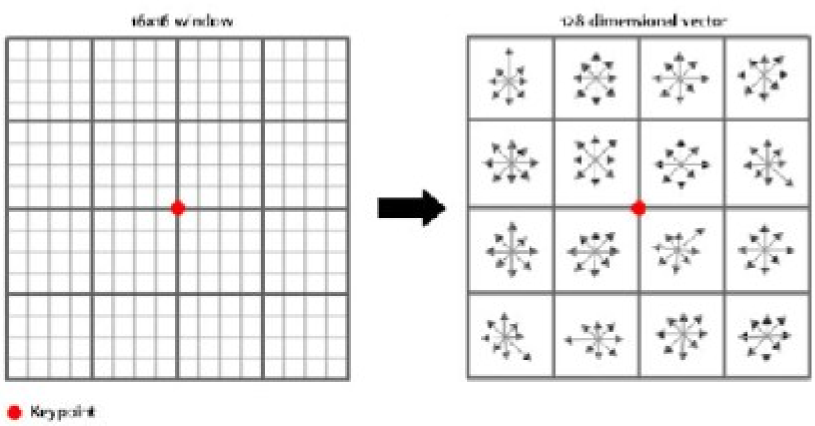
$$GradientVector = \begin{bmatrix} L(x+1, y) - L(x-1, y) \\ L(x, y+1) - L(x, y-1) \end{bmatrix}$$

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

This is done for all sixteen 4×4 regions. Ending up with 4x4x8 = 128 numbers.

This keypoint is uniquely identified by this feature vector.



Our Code:

Haris Detector:

```
def get_interest_points(image, feature_width, threshold, k,
xIgnoreFromBegin=0,xIgnoreFromEnd=0,yIgnoreFromBegin=0,yIgnoreFromEnd=0):
    if len(image.shape) == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image

    x = []
    y = []


    image_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)
    Ix, Iy = np.gradient(image_gaussian)
    Ixx = Ix * Ix
    Iyy = Iy * Iy
    Ixy = Ix * Iy
```

```
    row = image_gaussian.shape[0]
    column = image_gaussian.shape[1]

    offset = feature_width // 2

    for i in range(offset+yIgnoreFromBegin, row - offset-yIgnoreFromEnd):
       for j in range(offset+xIgnoreFromBegin, column - offset-xIgnoreFromEnd):
             Sxx = Ixx[i - offset:i + offset + 1, j - offset:j + offset + 1].sum()
             Syy = Iyy[i - offset:i + offset + 1, j - offset:j + offset + 1].sum()
             Sxy = Ixy[i - offset:i + offset + 1, j - offset:j + offset + 1].sum()

             det = (Sxx * Syy) - (Sxy * Sxy)
             trace = Sxx + Syy
             R = det - k * trace

              if R > threshold:
                     x.append(i)
                    y.append(j)

# These are placeholders - replace with the coordinates of your interest points!
   x = np.array(x)
   y = np.array(y)
   return x, y
```

Sift Descriptors generation (features Generation):

getting magnitude and direction of a desiered point:

```
def gradientMagnitudeAndDirection(i, j, image):
    gradientMagnitude = (((image[i + 1, j] - image[i - 1, j]) ** 2 + (image[i, j + 1] -
image[i, j - 1]) ** 2) ** 0.5)
    gradientOrientation = (180 / math.pi) * math.atan2((image[i, j + 1] - image[i, j -
1]),
                                                       (image[i + 1, j] - image[i - 1,
j]))
    return gradientMagnitude, gradientOrientation
```

histogram generation for each block:

```
def histogram(i, j, image):
    i = int(i)
    j = int(j)
    hist = [0] * 8
    for b in range(i - 4, i):
        for c in range(j - 4, j):
            magnitude, theta = gradientMagnitudeAndDirection(b, c, image)
```

```
        while (theta < 0):
            theta = theta + 360
        if theta >= 0 and theta <= 45:
            hist[0] += magnitude
        if theta > 45 and theta <= 90:
            hist[1] += magnitude
        if theta > 90 and theta <= 135:
            hist[2] += magnitude
        if theta > 135 and theta <= 180:
            hist[3] += magnitude
        if theta > 180 and theta <= 225:
            hist[4] += magnitude
        if theta > 225 and theta <= 270:
            hist[5] += magnitude
        if theta > 270 and theta <= 315:
            hist[6] += magnitude
        if theta > 315 and theta <= 360:
            hist[7] += magnitude
    return hist
```

Creating the Descriptors through applying the histogram function to each of the 16 block:

```
def descriptor(i, j, image):
    dis = [0] * 16
    dis[0] = histogram(i - 4, j - 4, image)
    dis[1] = histogram(i - 4, j, image)
    dis[2] = histogram(i - 4, j + 4, image)
    dis[3] = histogram(i - 4, j + 8, image)
    dis[4] = histogram(i, j - 4, image)
    dis[5] = histogram(i, j, image)
    dis[6] = histogram(i, j + 4, image)
    dis[7] = histogram(i, j + 8, image)
    dis[8] = histogram(i + 4, j - 4, image)
    dis[9] = histogram(i + 4, j, image)
    dis[10] = histogram(i + 4, j + 4, image)
    dis[11] = histogram(i + 4, j + 8, image)
    dis[12] = histogram(i + 8, j - 4, image)
    dis[13] = histogram(i + 8, j, image)
    dis[14] = histogram(i + 8, j + 4, image)
    dis[15] = histogram(i + 8, j + 8, image)
    return dis
```

Finally save the descriptor:

```
def get_features(image, x, y, feature_width):
    if len(image.shape) == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image
    image_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)
    features = []
# print(len(x))
    for i in range(len(x)):
       thisKeyPointdescriptor = descriptor(x[i], y[i], image_gaussian)
       features.append(thisKeyPointdescriptor)
```

```
    return np.array(features)
```

Matching the features(nearest neighbor method):

Getting the euclidean distance:

```
def euclidean_distance(im1_feature, im2_feature):
    sub = np.subtract(im1_feature, im2_feature)
    pow = np.multiply(sub, sub)
    the_euclidean_distance = np.sum(pow)
    return the_euclidean_distance
```

Getting the 2 nearest neighbors:

```
def get_neighbors(im1_feature, im2_features):
    minDistance1=1000
    minDistance1Index=-1
    minDistance2=1000
    minDistance2Index= -1
    for i in range(len(im2_features)):
        dist = euclidean_distance(im1_feature, im2_features[i, :, :])
        if(dist < minDistance1):
            minDistance2=minDistance1
            minDistance1=dist
            minDistance2Index=minDistance1Index
            minDistance1Index=i
    return  minDistance1,minDistance2,minDistance1Index
```

Calculation Distance1/Distance2 and comparing with the tolerance:

```
def match_features(im1_features, im2_features):
matches = []
confidences = []

tolerance = 1

for i in range(len(im1_features)):
    minDistance1, minDistance2, minDistance1Index = get_neighbors(im1_features[i,:,:],
im2_features)
    if (minDistance1/minDistance2)  < tolerance:
        matches.append((i,minDistance1Index))
        confidences.append(minDistance1/minDistance2)

# return matches
```

```
# These are placeholders - replace with your matches and confidences!
return np.array(matches),np.array(confidences)
```

Results:

Notre Dame:

the other two images required extra credit load and unsatisfactory results were outputted from using the current code even with varying the tolerance, threshold and suppression of false edges.

Haris threshold (R) is set to 2 for image one and 2.5 for image two:

```
(x1, y1) = student.get_interest_points(image1, feature_width, 2,0.06)
(x2, y2) = student.get_interest_points(image2, feature_width,2.5,0.06)
```

The nearest neighbour tolerance is set to 0.6:

```
tolerance = 0.6
```

Image one interest Points:

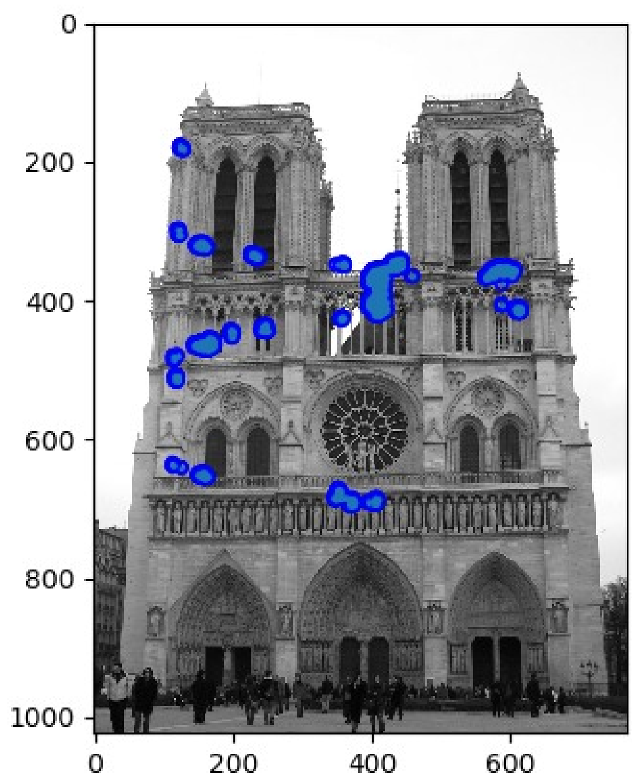
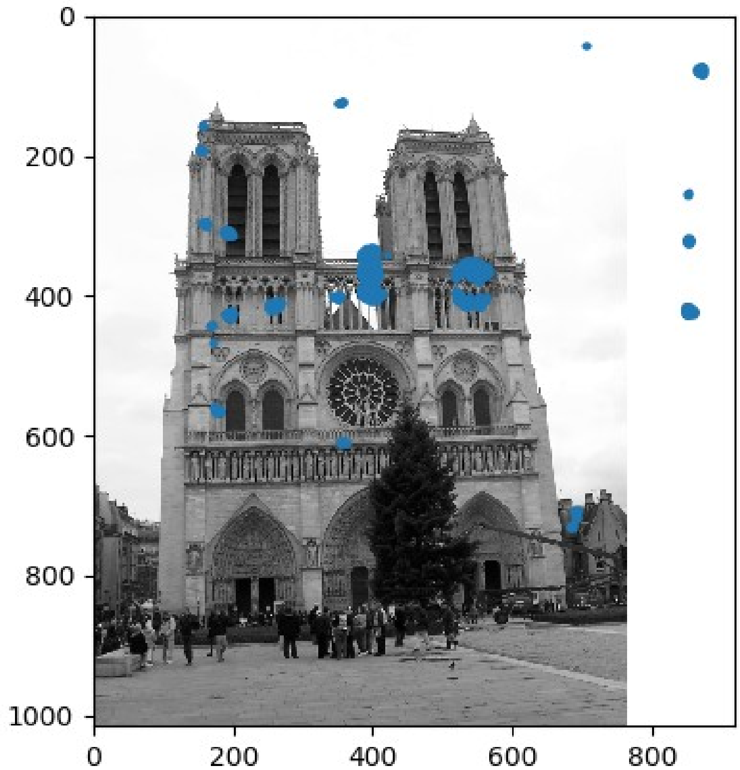
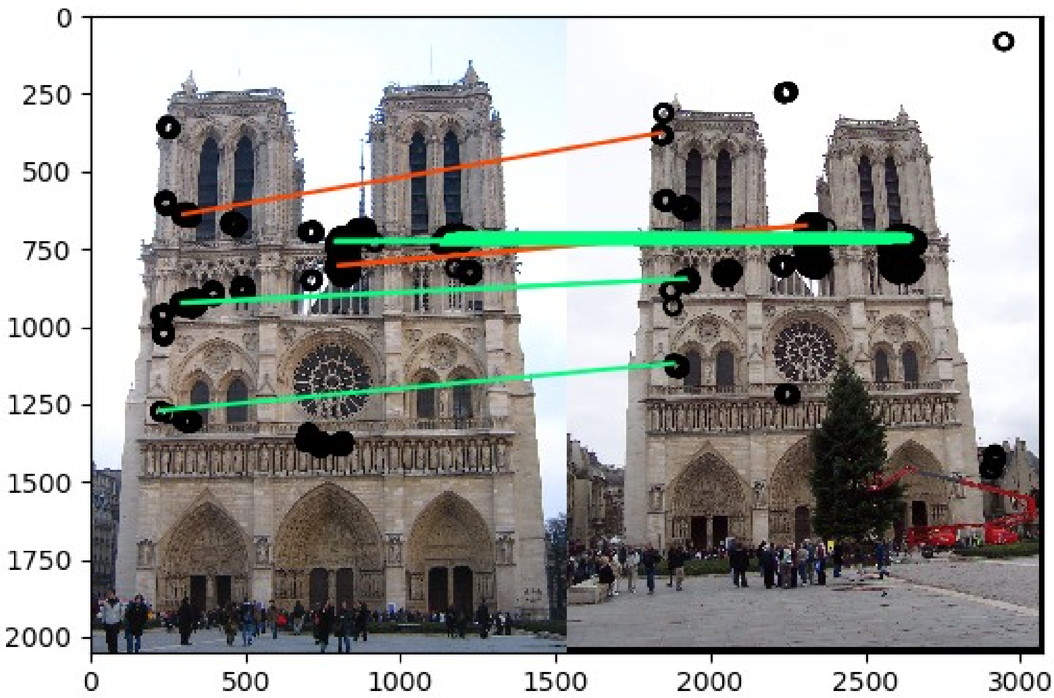


Image two interest points:



Matching the two images:



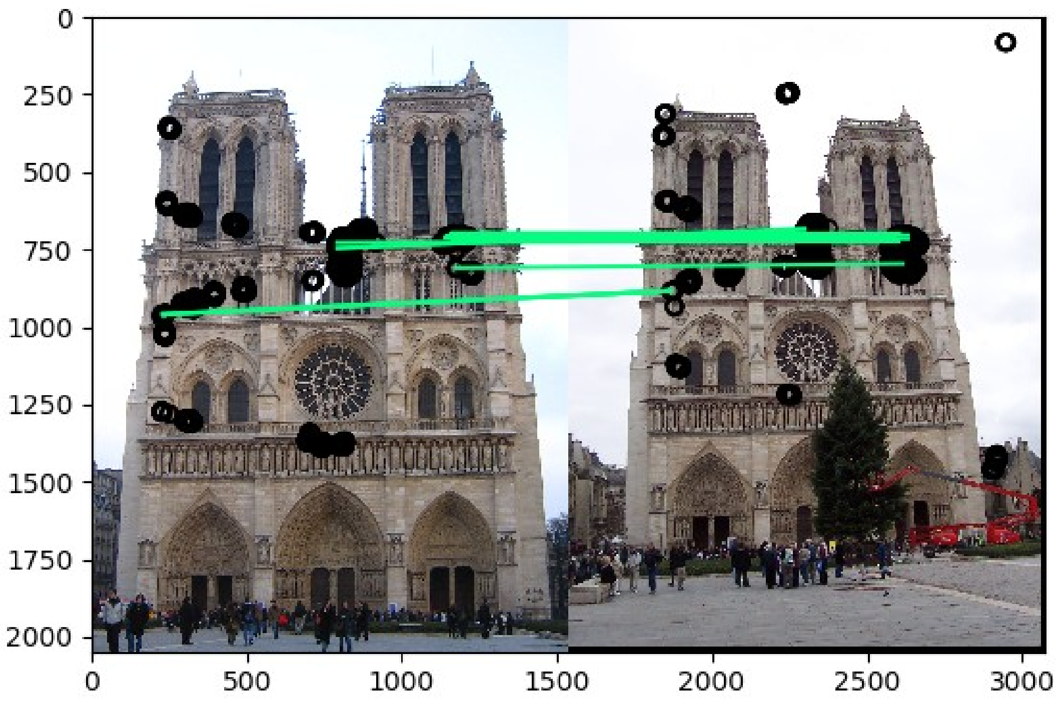
Accuracy:

```
Namespace(pair='notre_dame')
Getting interest points...
Done!
Getting features...
Done!
Matching features...
Done!
Matches: 126
Accuracy on 50 most confident: 94%
Accuracy on 100 most confident: 97%
Accuracy on all matches: 97%
Vizualizing...
```

Setting Tolerance to 0.4 :

```
tolerance = 0.4
```

Matching the two images:



Accuracy:

```
Namespace(pair='notre_dame')
Getting interest points...
Done!
Getting features...
Done!
Matching features...
Done!
Matches: 18
Accuracy on all matches: 100%
Vizualizing...
```