Project #4

Object Category Detection

EECE 5639

Instructor: Octavia Camps

Authors: Juan Pablo Bernal Medina, Shivam Sharma

Abstract

In this project, we implemented the generalized Hough transform to detect objects. First, we used 550 pictures to train the algorithm. In this group of pictures, for each, we found corners using Harris corner detection and took a 25x25 patch around each corner to use them as the visual words. We then applied K-means to all these patches to make clusters of similar features to create our visual vocabulary. Moreover, the distance from each corner to the center of the training image (the center of the object) was recorded as the displacement. Finally, to find the object in a new image, we used SSD to match these clusters to corners in the new image and use the displacements to vote for the center of the object.

Algorithm

Harris corner detector: We began by computing the gradient of the image by using a 3x3 Sobel operator. We then computed the C matrix for each pixel given by:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Where I_x and I_y are the gradients at (x, y). The output of the function gave us a Harris corner response at each pixel which we were then able to use to pinpoint corners on the image.

Non-max suppression: The output of the corner detector is then fed into a non-max suppression algorithm to find local maxima and therefore generate a sparse set of corners. To accomplish this we used a slightly different route and utilized mean shift to find every local maximum. The algorithm would be given a window size and would start searching the image at coordinates half the window size. It would start at that pixel and use the mean shift algorithm to find the nearest maxima. It would look in a window around that pixel and find the maximum. If the maximum was greater than the threshold it would move to that pixel location and repeat the process. This algorithm allowed us to find the corners with the greatest response and limit the number that we would find. The window size, in this case, limited the closest distance between observed corners.

Generate Patch: For each image of the training file, we detected corners and took a 25x25 patch around each corner and saved it to as a .mat file.

Generate Vocabulary: To generate the vocabulary, first, we applied K-means to the set of patches generated in the previous step. The output of the K-means is a set of clusters that groups similar patches together. Going back to the training images, we found corners again and extracted the 25x25 patch around the corner. We then used SSD to match each patch to its corresponding cluster. We also saved each displacement vector corresponding to each cluster. The displacement was taken as the x and y displacement from the corner to the center, as we assumed the center of the car was located at the center of the image.

Match Template: This function was used to find the car in separate test images where the object may be obscured, have more than one visible and be a larger background. Once more, we detected corners in the new image and selected the 25x25 patch around this corner. Then, we matched this template to the cluster using SSD. We took the cluster with the lowest score and using the displacements of the cluster we voted for the center of the object. Each cluster has corresponding displacement vectors found in the previous step. Each displacement vector is voted for and normalized by the total length of the set of displacement vectors. To help against noise we voted in a 5x5 region around the pixel with a Gaussian kernel. These votes were made into a grayscale Hough transform image (GHT) and the whitest pixels would represent the center

of one object. So each bright spot - local maxima should represent a car. Finally, we thresholded the GHT image and used non-max suppression to find the centers of each detected car. We then drew a bounding box around the centers by assuming a constant width and height of 100 and 40 pixels respectively.

Compute Accuracy: We then compared our bounding boxes against the ground truth to determine if we were correct and hence estimate our accuracy. A box was deemed a success if the area of the intersection with the ground truth box divided by the union of two exceeded 0.5.

Experiments

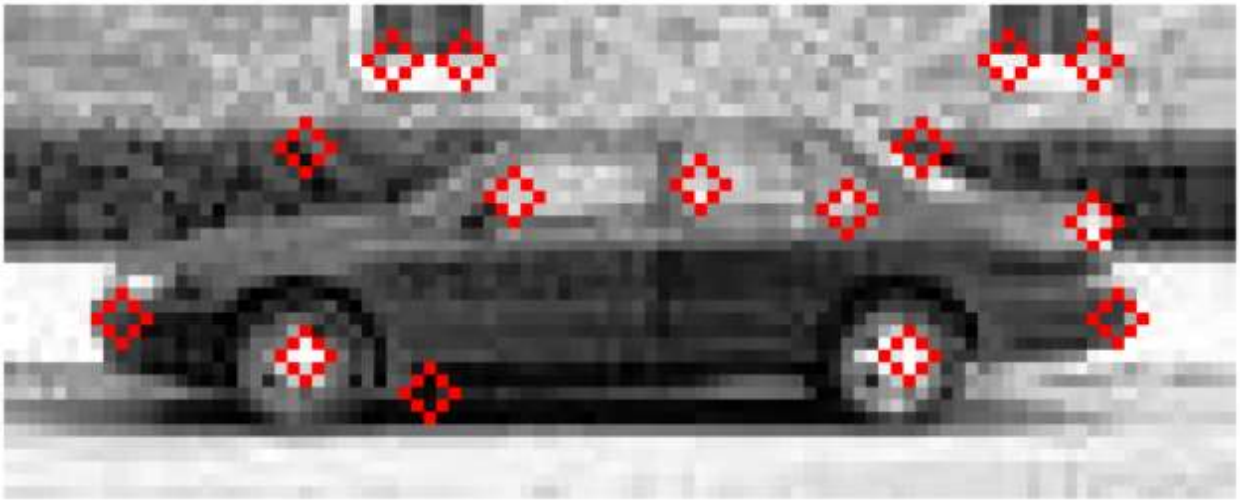


Fig 1. Corners detected in train image 1

Figure 1 shows a sample training image that was used to generate the visual words. A Harris corner detector is used to find corners and then 25x25 windows around each corner as saved in a matrix. This is done for all 550 training images to generate windows for each corner in each image.

The window matrix was then clustering into sets using k-means. We found that 25 clusters proved to be optimal. Figure 2 below shows one of the windows after the k-means clustering that most likely is for the wheel of the car

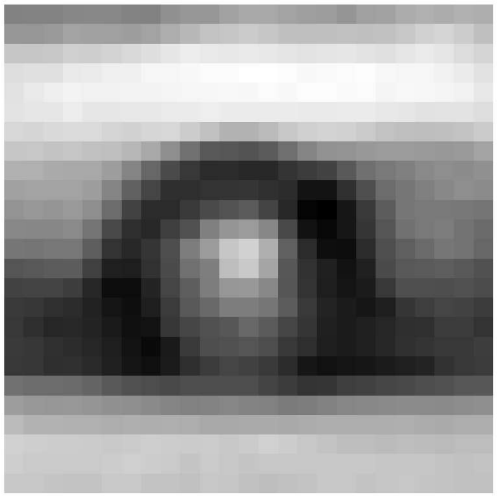


Fig 2. 25x25 window corresponding to a wheel

After the k-means clustering, we cycle back through all the training images and match each corner to one of the clusters using minimum SSD. We then store the position of the corner - the displacement from the center in a matrix keeping track of which cluster the corner belongs to. In the end, we have a set of displacement vectors for each cluster.

Now we look through the test data and once again find corners and match them to the cluster using SSD. We then vote for a group of pixels based on the displacement vectors belonging to that matched cluster. Once this is done for each corner in the test image we end up with a grayscale General Hough Transform image such as the one below - Figure 3.

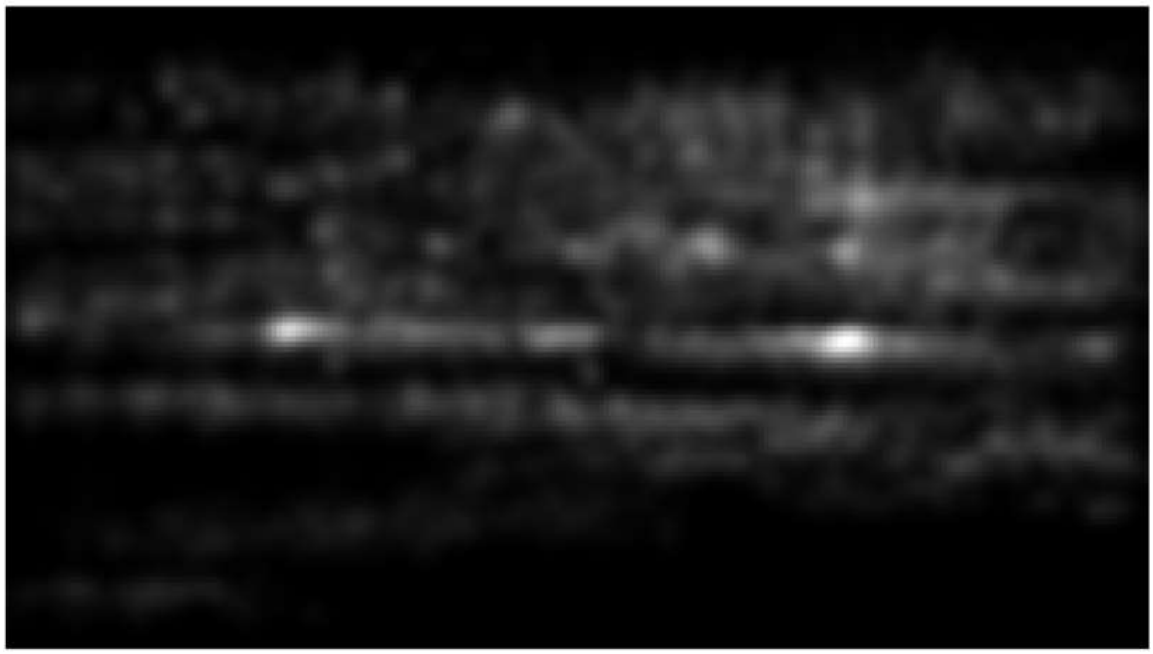


Fig 3. Test image 5 GHT image

We then use non-max suppression to find a sparse set of local maxima on the GHT image. The local maxima signify the center of a car and we then use a fixed size to draw a bounding box around the car, shown in figure 4 below.

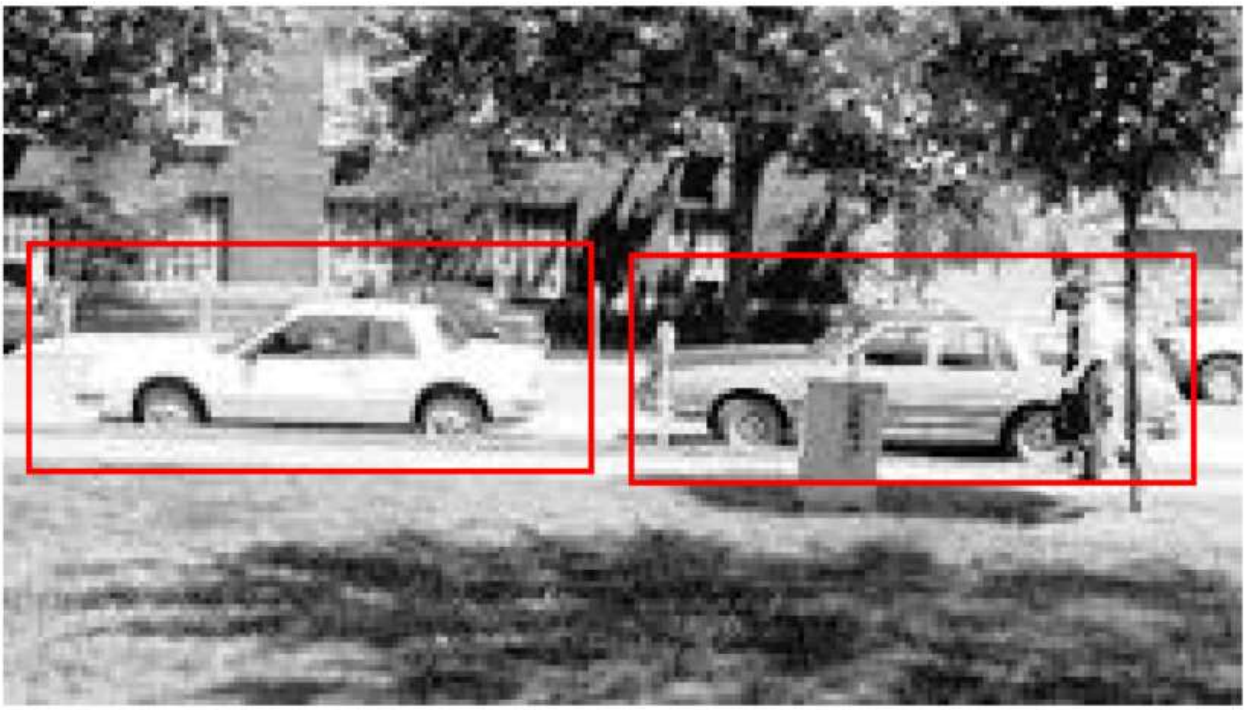


Fig 4. Test image 5 detection and bounding boxes drawn

Values of parameters used

Corner detection:

Harris and Sobel window size were chosen to be 5. This was mainly due to the immensely larger runtime if the harris window size was any bigger. The program works well enough with a window size of 5 so we kept it at that.

Harris corners threshold: 2.5 - arbitrarily chosen to get a good amount of corners. The main purpose is to allow negative Harris pixels to be ignored.

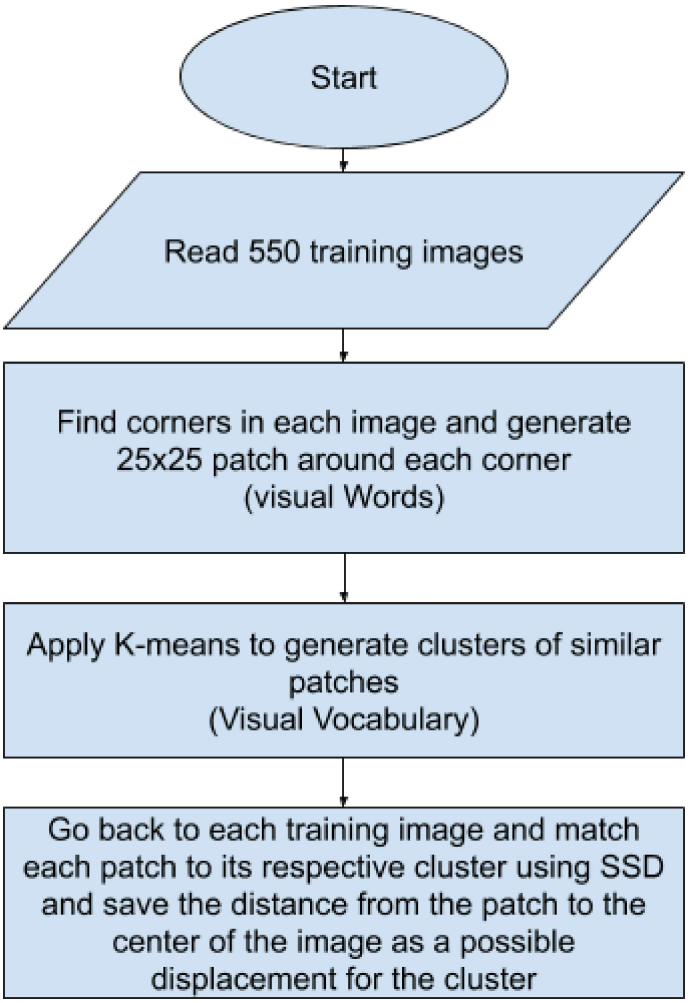
The non-max suppression window size was chosen to be 15. This is important as the window size determines the minimum distance between the observed corner (in our implementation). So for a window size of 15, the minimum lateral distance between corners is 8 pixels.

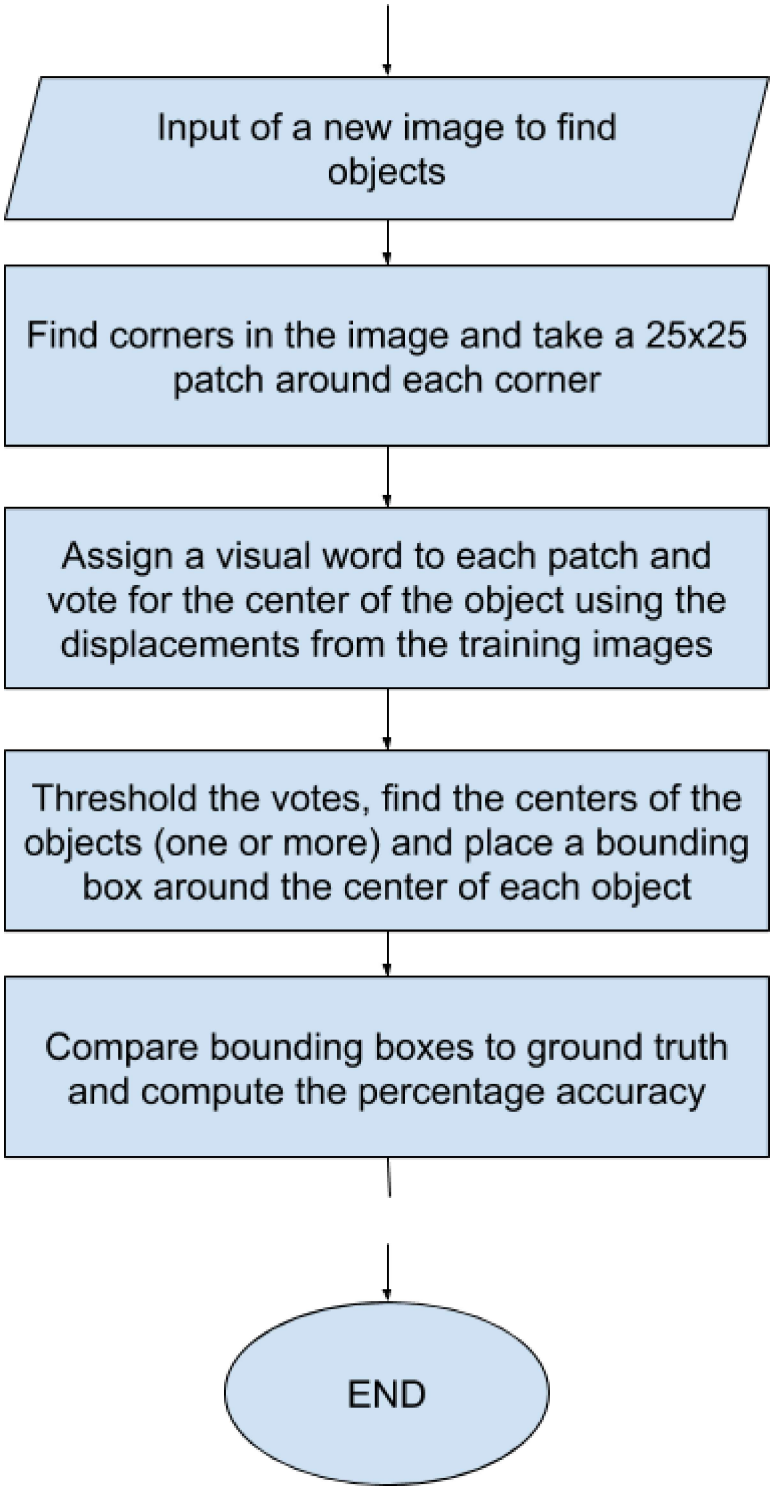
K-means:

For a 25 k-means cluster, our detection accuracy was 70%. We correctly identified 85 cars from a total of 120 from all the 100 test images. By changing the k means number of clusters, our accuracy changed as well but it was the best for a size of 25.

With too few clusters problems arise due to other non-car corners adding noise to the GHT image. This is the case for all cluster sizes, not just a small number, but in this particular case since there are fewer patches to match to it is more likely that a non-car corner will be matched to a car corner and produce many bright spots on the GHT image.
On the other hand with too many clusters each patch becomes too specific. We will no longer try to match certain parts of a car together to one cluster, instead since there will be so many clusters each corner will have a much more specific match and displacement vector. This decreases the probability of a car producing one bright spot on the GHT image.

Flowchart:




Input of a new image to find objects
Find corners in the image and take a 25x25 patch around each corner
Assign a visual word to each patch and vote for the center of the object using the displacements from the training images
Threshold the votes, find the centers of the objects (one or more) and place a bounding box around the center of each object
Compare bounding boxes to ground truth and compute the percentage accuracy
END

Conclusions

In conclusion, to be able to find different types of cars in different types of images we trained the algorithm using 550 images of different cars. From these training images, we found interesting features (corners) group similar features into 25 clusters and found the distances from each feature to the center of each car. This training data was used to develop a generalized Hough transform to find as many cars as a picture could have. Then we used 100 test images with 120 cars in total. We then in each image found corners and match out cluster template to patches around each corner. Finally, we used the Hough transform with the training data to vote for the center of a car. Our algorithm was able to correctly detect 70% (85 cars) of the cars in the test images.

Appendix

More successful detections:

Fig 3. Test image 41

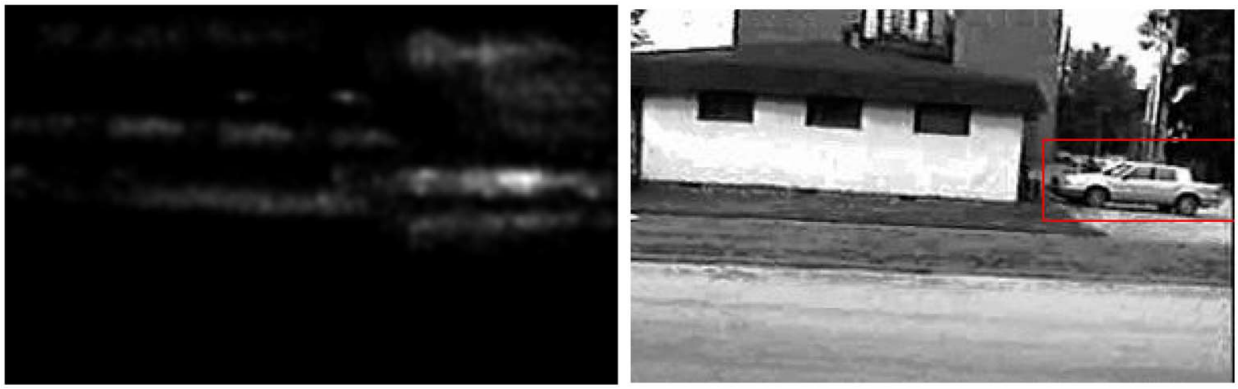


Fig 4. Test image 60

Code:

```python
import numpy as np
from matplotlib import pyplot as plt
import math
import cv2
import os
import gc
import scipy.io as sio


def Harris(img, window_size=3, sobel_size=3, k=0.04, step_size=1):
    if len(img.shape) > 2:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    h, w = img.shape[0], img.shape[1]

    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=sobel_size)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=sobel_size)

    Ixx = sobelx ** 2
    Iyy = sobely ** 2
    Ixy = sobelx * sobely

    R = np.zeros_like(img, dtype=np.float64)

    offset = np.floor(window_size / 2).astype(np.int8)
    for y in range(offset, h - offset, step_size):
        for x in range(offset, w - offset, step_size):
            Sxx = np.sum(Ixx[y - offset:y + 1 + offset, x - offset:x + 1 + offset])
            Syy = np.sum(Iyy[y - offset:y + 1 + offset, x - offset:x + 1 + offset])
            Sxy = np.sum(Ixy[y - offset:y + 1 + offset, x - offset:x + 1 + offset])
            r = (Sxx * Syy) - (Sxy ** 2) - k * (Sxx + Syy) ** 2
            if r < 0:
                r = 0
            R[y][x] = r

    R /= np.max(R)
    R *= 255.0
    # _, R = cv2.threshold(R, 0.01 * np.max(R), 255, 0)
    return R


def non_max_suppression(img, window_size=5, thresh=2.5):
    h, w = img.shape[0], img.shape[1]

    offset = np.floor(window_size / 2).astype(np.int8)

    temp = np.zeros_like(img)
    temp[offset:h - offset, offset:w - offset] = img[offset:h - offset, offset:w - offset]
    img = temp
    del temp

    index = np.zeros((h, w, 3), dtype=np.int16)

    index[:, :, 0] = img

    for y in range(h):
        for x in range(w):
            index[y][x][1] = x
            index[y][x][2] = y

    corners = []

    global NMS_index
    global NMS_corners
    NMS_index = np.zeros((h, w))
```

```
    NMS_corners = np.zeros((h, w))

    for y in range(offset, h - offset, offset):
        for x in range(offset, w - offset, offset):
            ret = mean_shift_converge(index, (x, y), window_size, thresh)
            if ret:
                corners.append(ret)

    corners = list(set(corners))
    corners.sort()
    return corners


def mean_shift_converge(index, point, window_size=5, thresh=2.5):
    x, y = point
    offset = np.floor(window_size / 2).astype(np.int8)

    if NMS_index[y][x] == 1:
        return None

    window = index[y - offset:y + 1 + offset, x - offset:x + 1 + offset]

    if np.any(NMS_corners[y - offset:y + 1 + offset, x - offset:x + 1 + offset]):
        return None

    if np.max(window[:, :, 0]) > thresh:
        window = np.reshape(window, (window_size ** 2, 3))
        window[::-1] = window[window[:, 0].argsort()]
        if window[0][1] == x and window[0][2] == y:
            NMS_index[y][x] = 1
            NMS_corners[y][x] = 1
            return x, y
        else:
            ret = mean_shift_converge(index, (window[0][1], window[0][2]), window_size, thresh)
            if ret:
                NMS_index[y][x] = 1
                return ret[0], ret[1]
            else:
                return None
    else:
        return None


def generate_patch(harris_window=5, display=False):
    train_path = "CarTrainImages"

    WINDOW_SIZE = 25
    OFFSET = int(np.floor(WINDOW_SIZE/2))

    patch = []

    i = 0
    for serial_number in os.listdir(train_path + '/'):
        train_image = cv2.cvtColor(cv2.imread(train_path + '/' + serial_number), cv2.COLOR_BGR2GRAY)
        print(serial_number)

        w, h = train_image.shape[1], train_image.shape[0]

        ret = Harris(train_image, harris_window, harris_window, 0.04, step_size=1)
        centroids = non_max_suppression(ret, 9)

        for center in centroids:
            x = int(center[0])
            y = int(center[1])
            if x < OFFSET or x >= (w-OFFSET) or y < OFFSET or y >= (h-OFFSET):
```

```
                continue
            window = train_image[y-OFFSET:y+1+OFFSET, x-OFFSET:x+1+OFFSET]

            if i == 0:
                patch = (window.flatten()).copy()
            else:
                patch = np.vstack([patch, window.flatten()])

            i += 1

        output = cv2.cvtColor(train_image.copy(), cv2.COLOR_GRAY2BGR)
        for center in centroids:
            cv2.circle(output, (int(center[0]), int(center[1])), 2, (255, 0, 0), 1)

        if display:
            plt.imshow(output, cmap='gray')
            plt.axis('off')
            plt.show()

    print(patch.shape)
    sio.savemat('patch.mat', {'patch': patch})


def generate_vocabulary():
    M = sio.loadmat('kmeans.mat')
    idx = M['idx']
    C = M['C']

    K = []

    window_size = int(np.sqrt(len(C[0])))

    for frame in C:
        K.append(np.reshape(frame, (window_size, window_size)))
    K = np.array(K)

    train_path = "CarTrainImages"

    OFFSET = int(np.floor(window_size / 2))

    displacements = [list()] * len(K)

    for serial_number in os.listdir(train_path + '/'):
        train_image = cv2.cvtColor(cv2.imread(train_path + '/' + serial_number), cv2.COLOR_BGR2GRAY)
        print(serial_number)

        w, h = train_image.shape[1], train_image.shape[0]

        ret = Harris(train_image, 5, 5, 0.04, step_size=1)
        centroids = non_max_suppression(ret, 9)

        for center in centroids:
            x = int(center[0])
            y = int(center[1])
            if x < OFFSET or x >= (w-OFFSET) or y < OFFSET or y >= (h-OFFSET):
                continue
            SSD = []
            for f in range(len(K)):
                window = train_image[y-OFFSET:y+1+OFFSET, x-OFFSET:x+1+OFFSET]
                ssd = np.mean((window-K[f]) ** 2)
                SSD.append([ssd, f, (x-w/2, y-h/2)])
            SSD.sort(key=lambda z: z[0])

            if len(displacements[SSD[0][1]]) == 0:
                displacements[SSD[0][1]] = [SSD[0][2]]
```

```
            else:
                displacements[SSD[0][1]].append(SSD[0][2])

    sio.savemat('displacement.mat', {'d': displacements})


def match_template():
    M = sio.loadmat('displacement.mat')
    displacement = np.squeeze(M['d'])

    M = sio.loadmat('kmeans.mat')
    idx = M['idx']
    C = M['C']
    K = []
    window_size = int(np.sqrt(len(C[0])))
    OFFSET = int(np.floor(window_size / 2))
    for frame in C:
        K.append(np.reshape(frame, (window_size, window_size)))
    K = np.array(K)

    test_path = "CarTestImages"

    kernel = np.array( [[0.023528, 0.033969,  0.038393,  0.033969,  0.023528],
                        [0.033969, 0.049045,  0.055432,  0.049045,  0.033969],
                        [0.038393, 0.055432,  0.062651,  0.055432,  0.038393],
                        [0.033969, 0.049045,  0.055432,  0.049045,  0.033969],
                        [0.023528, 0.033969,  0.038393,  0.033969,  0.023528]])
    kernel_offset = 2

    c = 0
    for serial_number in os.listdir(test_path + '/'):
        c += 1
        if c < 18:
            continue

        test_image = cv2.cvtColor(cv2.imread(test_path + '/' + serial_number), cv2.COLOR_BGR2GRAY)
        print(serial_number)

        w, h = test_image.shape[1], test_image.shape[0]

        ret = Harris(test_image, 3, 3, 0.04, step_size=1)
        centroids = non_max_suppression(ret, 15)

        output = cv2.cvtColor(test_image.copy(), cv2.COLOR_GRAY2BGR)
        for center in centroids:
            cv2.circle(output, (int(center[0]), int(center[1])), 2, (255, 0, 0), 1)

        # plt.imshow(output, cmap='gray')
        # plt.axis('off')
        # plt.show()

        GHT_image = np.zeros_like(test_image, dtype=np.float64)
        for center in centroids:
            x = int(center[0])
            y = int(center[1])
            if x < OFFSET or x >= (w-OFFSET) or y < OFFSET or y >= (h-OFFSET):
                continue
            SSD = []
            for f in range(len(K)):
                window = test_image[y-OFFSET:y+1+OFFSET, x-OFFSET:x+1+OFFSET]
                ssd = np.mean((window-K[f]) ** 2)
                SSD.append([ssd, f, (x-w/2, y-h/2)])
            SSD.sort(key=lambda l: l[0])

            vote_scale = 1000 / len(displacement[SSD[0][1]])
```

```
            for d in displacement[SSD[0][1]]:
                x_new = int(x - d[0])
                y_new = int(y - d[1])
                if x_new < 2 or x_new >= w-2 or y_new < 2 or y_new >= h-2:
                    continue

                GHT_image[y_new-kernel_offset: y_new+kernel_offset+1, x_new-kernel_offset: x_new+kernel_offset+1] +=
kernel * vote_scale

        GHT_image = cv2.GaussianBlur(GHT_image, (5, 5), 1.5)

        offset = 25
        temp = np.zeros_like(GHT_image)
        temp[offset:h - offset, offset:w - offset] = GHT_image[offset:h - offset, offset:w - offset]

        GHT_centroids = non_max_suppression(GHT_image, window_size=93, thresh=0.8*temp.max())

        max_index = np.where(GHT_image == np.amax(GHT_image))
        if len(GHT_centroids) == 0:
            GHT_centroids.append((max_index[1][0], max_index[0][0]))

        # output = cv2.cvtColor(test_image.copy(), cv2.COLOR_GRAY2BGR)
        # cv2.circle(output, (int(max_index[1]), int(max_index[0])), 2, (0, 255, 0), 1)
        # for center in centroids:
        #     cv2.circle(output, (int(center[0]), int(center[1])), 2, (0, 0, 255), 1)
        # for center in GHT_centroids:
        #     cv2.circle(output, (int(center[0]), int(center[1])), 2, (255, 0, 0), 1)

        output = cv2.cvtColor(test_image.copy(), cv2.COLOR_GRAY2BGR)
        for center in GHT_centroids:
            window = GHT_image[center[1]-20:center[1]+21, center[0]-50:center[0]+51]
            top_left = (int(center[0]) - 50, int(center[1]) - 20)
            cv2.rectangle(output, (top_left[0], top_left[1]), (top_left[0]+100, top_left[1]+40), (255, 0, 0), 1)

        fig, (ax1, ax2) = plt.subplots(2, 1)
        ax1.imshow(GHT_image, cmap='gray')
        ax1.axis('off')
        ax2.imshow(output, cmap='gray')
        ax2.axis('off')

        plt.show()


if __name__ == "__main__":
    # generate_patch(display=False)
    # generate_vocabulary()
    match_template()
```