Homework 2 - Generalized Hough Transform

Theory

< Insert your answers here >

Part 1

For part 1, we have that:

$$R(\theta) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Such that applying this rotation + a 90 degree rotation yields table (ii) from the slides.

Part 2

Given that the angle intervals are 90 degree from each other and we have 3 gradient directions its safe to say we're dealing with a triangle.

Programming

Find object in an image using a template:

In [ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import cv2
import utils
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics.pairwise import euclidean_distances

def nonMaxSuprression(img, d=5):
```

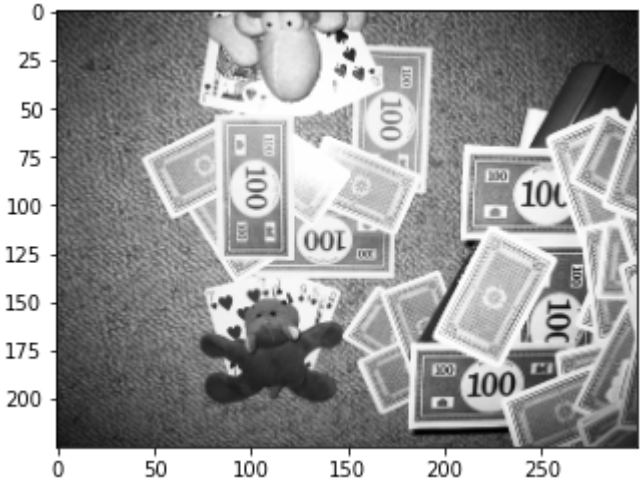
```
    """
    Given an image set all values to 0 that are not
    the maximum in its (2d+1,2d+1)-window

    Parameters
    ----------
    img : ndarray
        an image
    d : int
        for each pixels consider the surrounding (2d+1,2d+1)-window

    Returns
    -------
    result : ndarray

    """
    # TODO
    # iterate over pixels
    # iterate over (2d+1,2d+1) neighborhood window
    # supress non-maxima to 0
    # store results in new array

    result = np.zeros(img.shape)

    cols, rows = img.shape

    for x in range(0, cols):
        for y in range(0, rows):
            maxx = x + d
            maxy = y + d
            minx = x - d
            miny = y - d
            if maxx > cols:
                maxx = cols - 1
            if minx < 0:
                minx = 0
            if maxy > rows:
                maxy = rows - 1
            if miny < 0:
                miny = 0
            local_max = np.amax(img[minx:maxx, miny:maxy])
            if img[x,y] == local_max:
                result[x,y] = local_max

    return result

def calcBinaryMask(img, thresh = 0.3):
    """
    Compute the gradient of an image and compute a binary mask
    based on the threshold. Corresponds to O^B in the slides.

    Parameters
    ----------
    img : ndarray
        an image
    thresh : float
        A threshold value. The default is 0.3.

    Returns
    -------
```

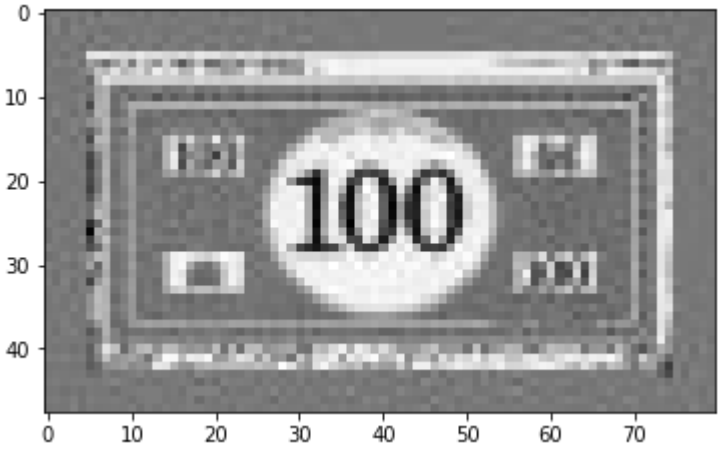
```
    binary : ndarray
        A binary image.

    """

    # TODO:
    # -compute gradients
    # -threshold gradients
    # -return binary mask

    gradients = utils.calcDirectionalGrad(img)

    absGradients = np.absolute(gradients)

    ret, out = cv2.threshold(absGradients, thresh*np.amax(absGradients), 255, cv2.THRE

    return out


def normalize(img):
    abs_img = np.absolute(img)
    norm_img = np.sum(abs_img)
    return np.array(list(map(lambda x:x/norm_img, img)))

def correlation(img, template):
    """
    Compute a correlation of gradients between an image and a template.

    Note:
    You should use the formula in the slides using the fourier transform.
    Then you are guaranteed to succeed.

    However, you can also compute the correlation directly.
    The resulting image must have high positive values at positions
    with high correlation.

    Parameters
    ----------
    img : ndarray
        a grayscale image
    template : ndarray
        a grayscale image of the template

    Returns
    -------
    ndarray
        an image containing the correlation between image and template gradients.
    """

    # TODO:
    # -compute gradient of the image
    # -compute gradient of the template
    # -copy template gradient into larger frame
    # -apply a circular shift so the center of the original template is in the
    #   upper left corner
    # -normalize template
    # -compute correlation

    image_gradients = utils.calcDirectionalGrad(img)
```

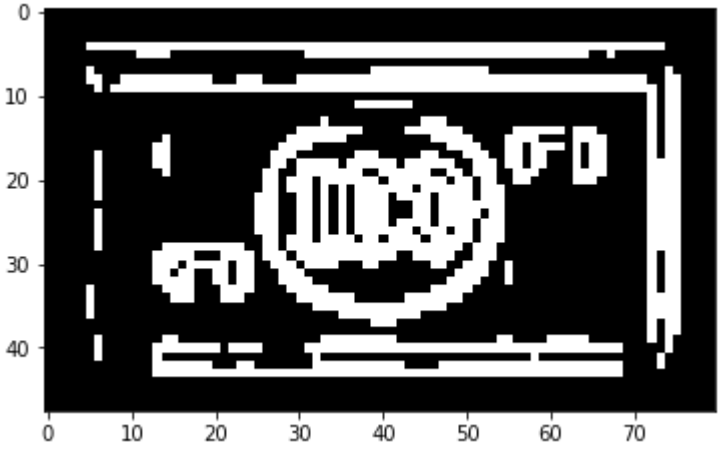
```
    template_gradients = utils.calcDirectionalGrad(template)

    corr_mask = np.zeros(img.shape) + 1.0j*np.zeros(img.shape)

    template_cols, template_rows = template.shape

    template_gradients = normalize(template_gradients)

    corr_mask[0:template_cols, 0:template_rows] = template_gradients*calcBinaryMask(te

    shifted_corr_mask = utils.circularShift(corr_mask, template_rows//2, template_cols

    return np.fft.ifft2(np.fft.fft2(image_gradients)*np.fft.fft2(np.conjugate(shifted_


def GeneralizedHoughTransform(img, template, angles, scales):
    """
    Compute the generalized hough transform. Given an image and a template.

    Parameters
    ----------
    img : ndarray
        A query image
    template : ndarray
        a template image
    angles : list[float]
        A list of angles provided in degrees
    scales : list[float]
        A list of scaling factors

    Returns
    -------
    hough_table : list[(correlation, angle, scaling)]
        The resulting hough table is a list of tuples.
        Each tuple contains the correlation and the corresponding combination
        of angle and scaling factors of the template.

        Note the order of these values.
    """
    # TODO:
    # for every combination of angles and scales
    # -distort template
    # -compute the correlation
    # -store results with parameters in a list

    result = []

    for theta in angles:
        for s in scales:
            score = np.abs(np.real(correlation(img, utils.rotateAndScale(template, the
            result.append((score, theta, s))

    return result
```

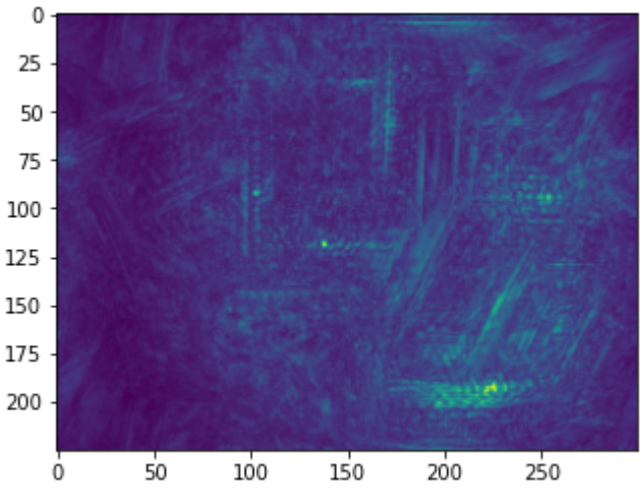
Main Program

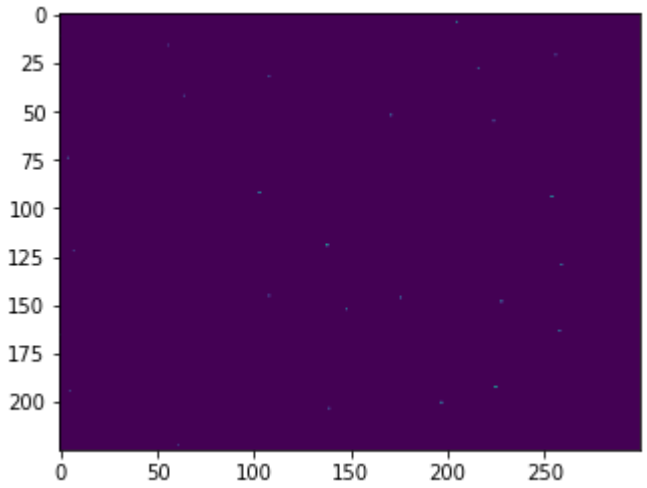
In [ ]:
```
# Load query image and template
query = cv2.imread("data/query.jpg", cv2.IMREAD_GRAYSCALE)
template = cv2.imread("data/template.jpg", cv2.IMREAD_GRAYSCALE)

# Visualize images
utils.show(query)
utils.show(template)
utils.show(calcBinaryMask(template))

# Create search space and compute GHT
angles = np.linspace(0, 360, 36)
scales = np.linspace(0.9, 1.3, 10)
ght = GeneralizedHoughTransform(query, template, angles, scales)

# extract votes (correlation) and parameters
votes, thetas, s = zip(*ght)

# Visualize votes
votes = np.stack(votes).max(0)
plt.imshow(votes)
plt.show()

# nonMaxSuprression
votes = nonMaxSuprression(votes, 20)
plt.imshow(votes)
plt.show()

# Visualize n best matches
n = 10
coords = zip(*np.unravel_index(np.argpartition(votes, -n, axis=None)[-n:], votes.shape
vis = np.stack(3*[query],2)
for y,x in coords:
    print(x,y)
    vis = cv2.circle(vis,(x,y), 10, (255,0,0), 2)
utils.show(vis)
```



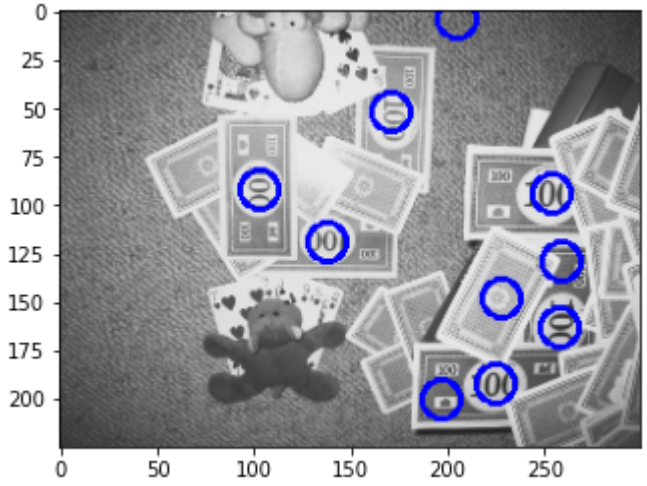

0
10
20
30
40
0
10
20
30
40
50
60
70


0
10
20
30
40
0
10
20
30
40
50
60
70


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250



```
205 4
259 129
171 52
197 200
254 94
103 92
138 119
228 148
225 192
258 163
```

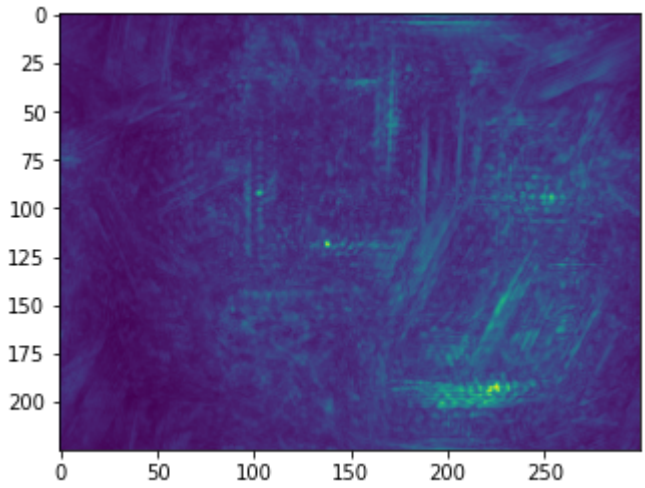


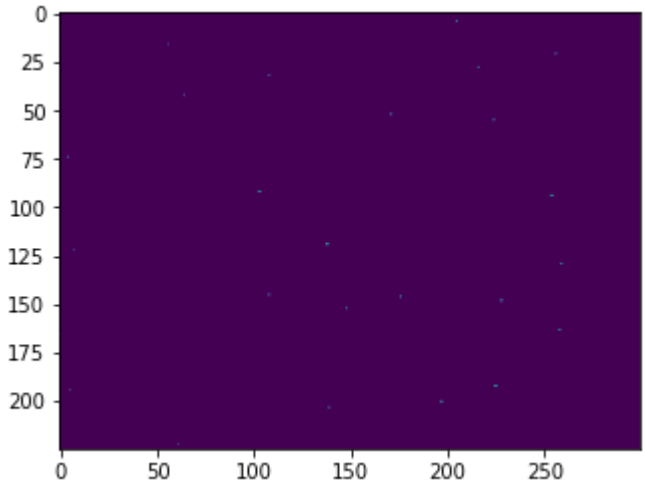
Test your implementation

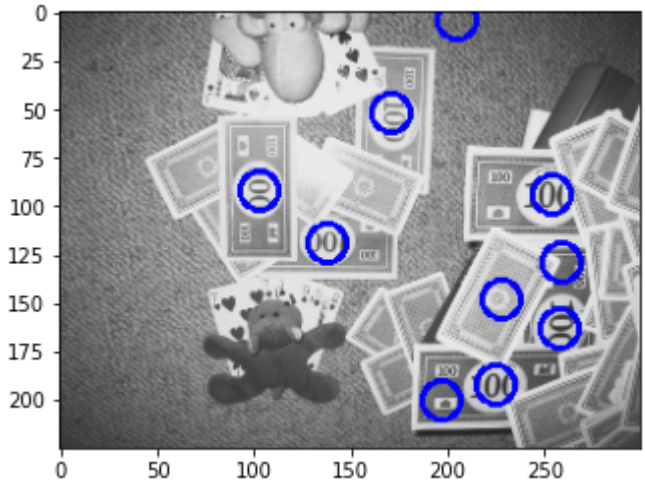
In [ ]:
```
import utils
import cv2
import json
from matplotlib import pyplot as plt
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
```

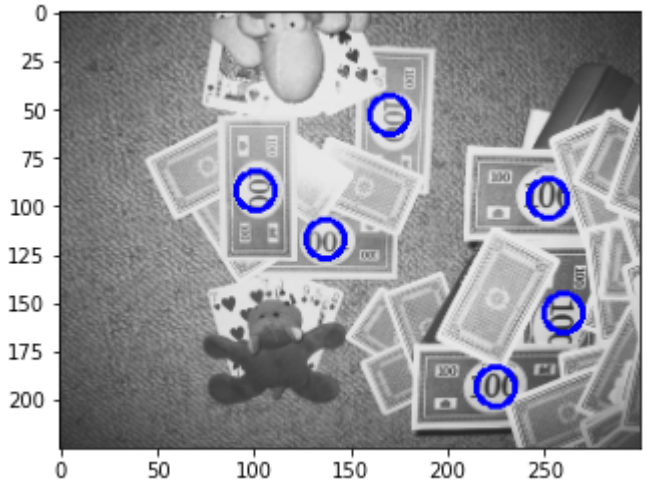
In [ ]:
```
from sklearn.metrics.pairwise import euclidean_distances

def testGHT():
    query = cv2.imread("data/query.jpg", cv2.IMREAD_GRAYSCALE)
```

```
    template = cv2.imread("data/template.jpg", cv2.IMREAD_GRAYSCALE)


    angles = np.linspace(0, 360, 36)
    scales = np.linspace(0.9, 1.3, 10)
    ght = GeneralizedHoughTransform(query, template, angles, scales)

    votes, thetas, s = zip(*ght)
    votes = np.stack(votes).max(0)
    plt.imshow(votes)
    plt.show()

    # votes = correlation(query, template)
    votes = nonMaxSuprression(votes, 20)
    plt.imshow(votes)
    plt.show()

    n = 10
    coords = list(zip(*np.unravel_index(np.argpartition(votes, -n, axis=None)[-n:], vo

    vis = np.stack(3*[query],2)
    for y,x in coords:
        vis = cv2.circle(vis,(x,y), 10, (255,0,0), 2)
    utils.show(vis)

    f = open("centroids.txt", "r")
    centroids = f.read()
    f.close()

    centroids = centroids.split("\n")[:-1]
    centroids = [centroid.split() for centroid in centroids]
    centroids = np.array([[int(centroid[0]),int(centroid[1])] for centroid in centroid

    vis = np.stack(3*[query],2)
    for x,y in centroids:
        vis = cv2.circle(vis,(x,y), 10, (255,0,0), 2)
    utils.show(vis)

    coords = np.array(coords)[:,::-1]


    d = euclidean_distances(centroids, coords).min(1)

    correct_detections = np.count_nonzero((d<10))

    score = { "scores": {"Correct_Detections": correct_detections }}

    print(json.dumps(score))

testGHT()
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250



{"scores": {"Correct_Detections": 6}}