Homework 2 - Generalized Hough Transform

Theory

Task 1: ii

Task 2: i

In [1]:
```
123
```

Out[1]:
```
123
```

Programming

Find object in an image using a template:
 title  title

In [81]:
```
#!/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

    """
    rows,cols = img.shape
    result = np.zeros((rows,cols))

    for row in range(0,img.shape[0]):
        for col in range(0,img.shape[1]):
            r_min=max(0, row-d)
            r_max=min(rows+1, row+d+1)
            c_min=max(0, col-d)
            c_max=min(cols+1, col+d+1)
```

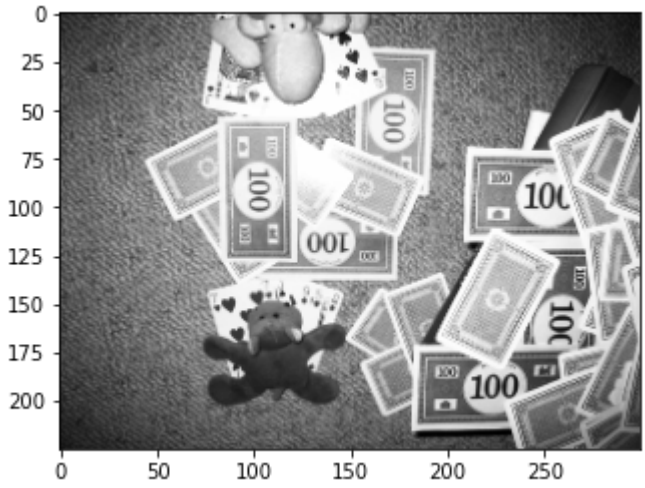
```
            result[row,col]=img[row,col] if img[row,col]==np.max(img[r_min:r_max,c_
            and np.max(result[r_min:r_max,c_min:c_max]) == 0 else 0

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

    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.

    """

    gX=utils.calcDirectionalGrad(img)
    gX = np.abs(gX)
    tH=np.max(np.abs(gX))*thresh
    for row in range(0,gX.shape[0]):
        for cow in range(0,gX.shape[1]):
            gX[row,cow]=1 if np.abs(gX[row,cow])>tH else 0

    return gX

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:
```

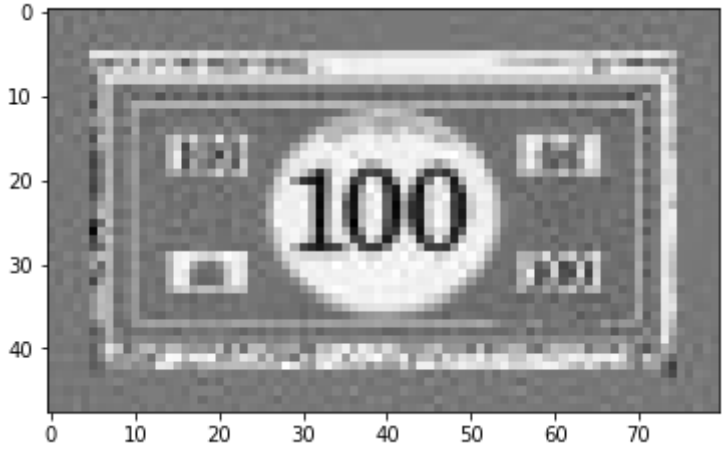
```
    rows, cols=  img.shape
    rows1,cols1=template.shape

    # -compute gradient of the image
    img_gra=utils.calcDirectionalGrad(img)

    # -compute gradient of the template
    temp_binary=calcBinaryMask(template, thresh = 0.3)
    temp_gra=utils.calcDirectionalGrad(template)
    temp_b_gra=temp_gra*temp_binary

    # -copy template gradient into larger frame
    temp_lag=np.zeros_like(img, dtype=complex)
    temp_lag[0:rows1,0:cols1]=temp_b_gra

    # -apply a circular shift so the center of the original template is in the
    #   upper left corner
    temp=utils.circularShift(temp_lag,int(cols1/2),int(rows1/2))

    # -normalize template
    temp=temp/np.sum(np.abs(temp))

    # -compute correlation
    img_f = np.fft.fft2(img_gra)
    te_f = np.fft.fft2(temp)
    vote=  img_f*np.conj(te_f)
    vote = np.real(np.fft.ifft2(vote))


    return vote


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
    hough_table=[]
    index=0
    for angle in np.array(angles):
        for scale in np.array(scales):
            # -distort template
```

```
            temp=utils.rotateAndScale(template, angle, scale)

            # -compute the correlation
            cor=correlation(img, temp)
            hough_table.append((cor,angle,scale))
            index=index+1
            if index%20==0:
                print("finish a loop"+str(index))
            # -store results with parameters in a list

    return hough_table
```

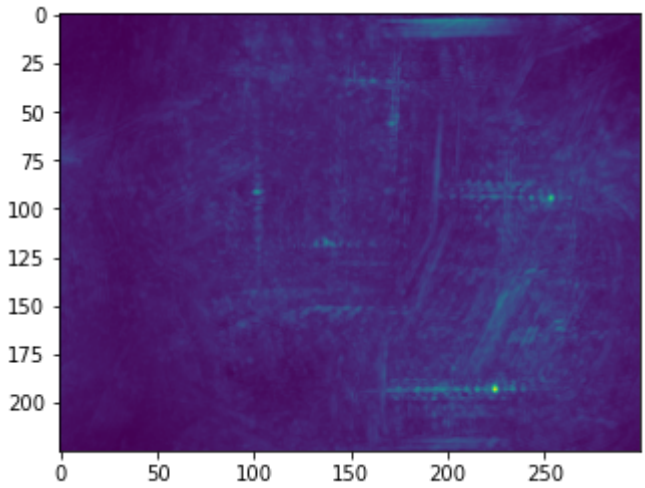
Main Program

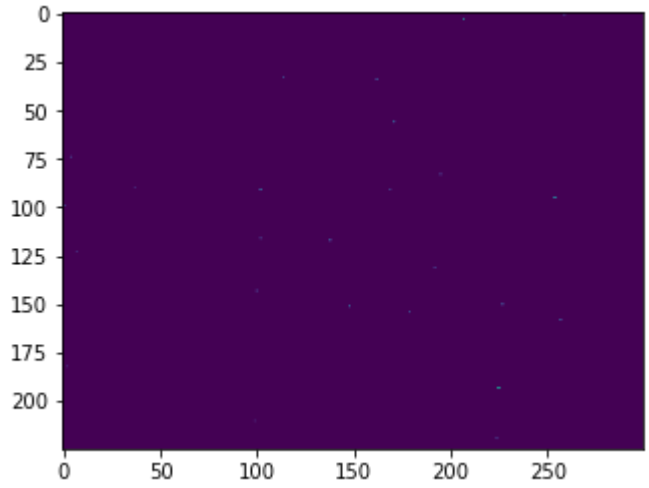
In [82]:
```
# Load query image and template
query = cv2.imread("data/query.jpg", cv2.IMREAD_GRAYSCALE)
template = cv2.imread("data/template.jpg", cv2.IMREAD_GRAYSCALE)

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

# 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.sh
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)
```



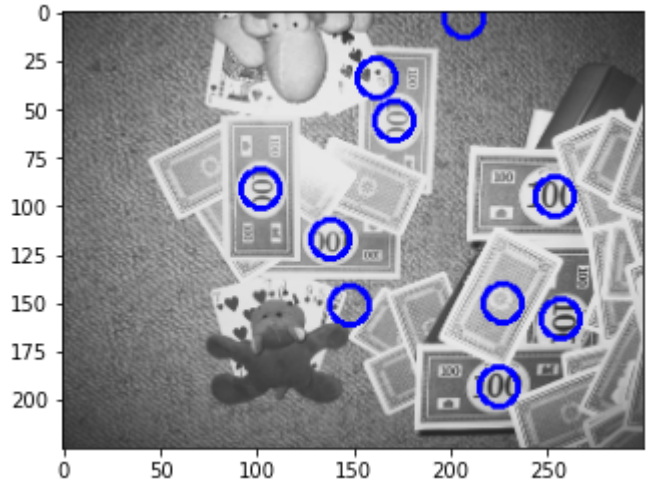


```
finish a loop20
finish a loop40
finish a loop60
finish a loop80
finish a loop100
finish a loop120
finish a loop140
finish a loop160
finish a loop180
finish a loop200
finish a loop220
finish a loop240
finish a loop260
finish a loop280
finish a loop300
finish a loop320
finish a loop340
finish a loop360
```





```
148 151
171 56
227 150
138 117
162 34
207 3
257 158
102 91
254 95
225 193
```



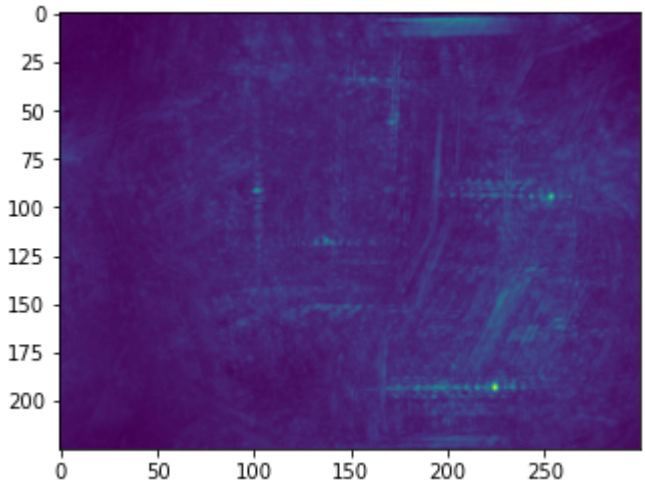
Test your implementation

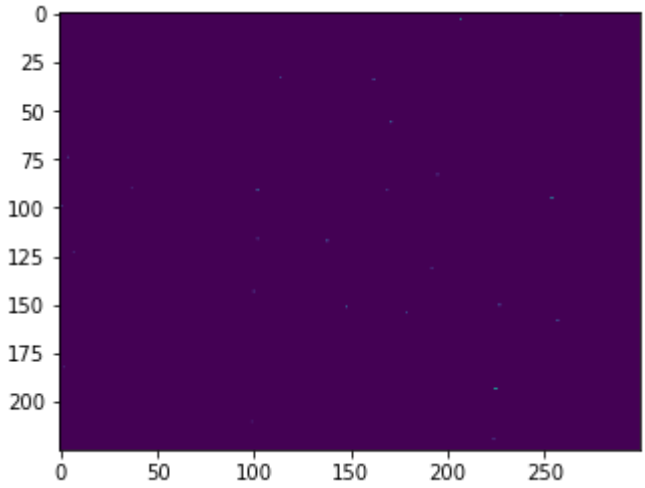
In [ ]:
```
import utils
import cv2
import json
from matplotlib import pyplot as plt
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
```

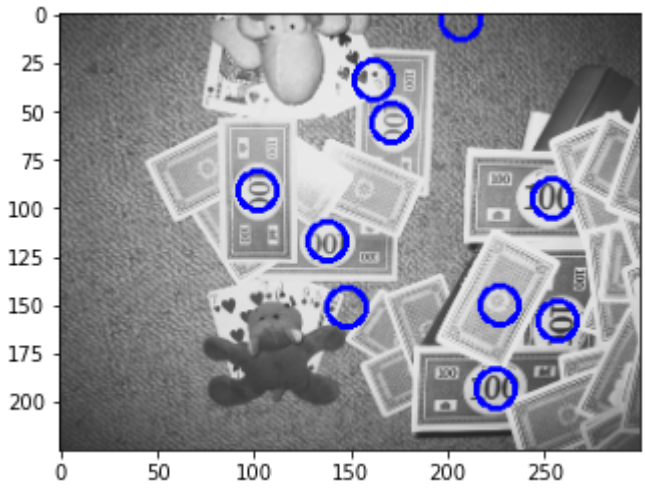
In [89]:
```
from sklearn.metrics.pairwise import euclidean_distances

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)
```

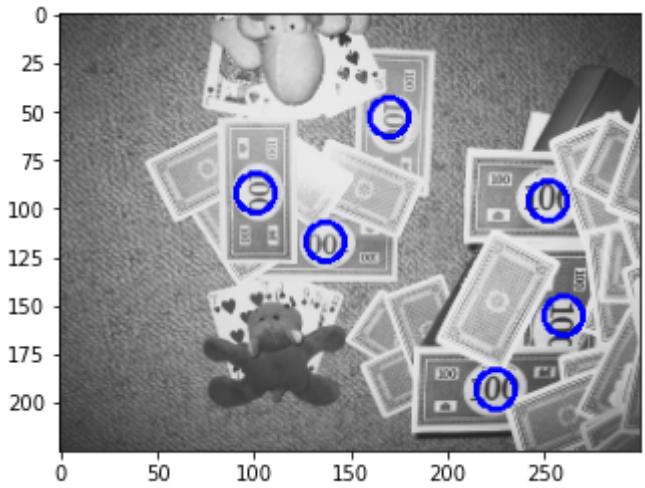
```
    votes, thetas, s = zip(*ght)
    votes = np.stack(votes).max(0)
    plt.imshow(votes)
    plt.show()

    #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:],

    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 centr

    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()
```

```
finish a loop20
finish a loop40
finish a loop60
finish a loop80
finish a loop100
finish a loop120
finish a loop140
finish a loop160
finish a loop180
finish a loop200
finish a loop220
finish a loop240
finish a loop260
finish a loop280
finish a loop300
finish a loop320
finish a loop340
finish a loop360
```









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

In [ ]:

In [ ]:

In [ ]: