به نام خدا

مهدی فیروزبخت

تمرین سری سوم

درس بینایی کامپیوتر

برای تمرین اول و دوم از ۲ تصویر زیر به عنوان ورودی استفاده کرده‌ایم. همچنین حالت نهایی دستی ایجاد شده برای محاسبه دقت در زیر آن قرار داده شده است.

۱.

ابتدا به توضیح روش اوتسو میپردازیم . الگوریتم به طور جامع آستانه ای را جستجو می کند که واریانس درون کلاسی را به حداقل می رساند، که به عنوان مجموع وزنی واریانس های دو کلاس تعریف می شود:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

وزن‌های $w_0$ و $w_1$ احتمالات دو کلاس هستند که با یک آستانه t از هم جدا شده‌اند و $\sigma^2_0$ و $\sigma^2_1$ واریانس‌های این دو کلاس هستند.

احتمال کلاس $w_{0,1}(t)$ از باین های L هیستوگرام محاسبه می شود:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

برای ۲ کلاس، به حداقل رساندن واریانس درون کلاسی معادل به حداکثر رساندن واریانس بین کلاسی است:

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(t)(\mu_0 - \mu_T)^2 + \omega_1(t)(\mu_1 - \mu_T)^2$$
$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

که در آن w به معنی احتمال هر کلاس و μ به معنی میانگین آن کلاس است که میانگین به صورت زیر محاسبه میشود :

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

که با توضیحات بالا به توضیح الگوریتم به همراه کدها میپردازیم.

ابتدا کتابخانه ها و توابع مورد نیاز برای بارگذاری تصاویر را پیاده سازی میکنیم :

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import time
```

```
def load_data(name):
  image = cv2.imread(name)
  cv2_imshow(image)
  return image
```

```
def BGR2GRAY(image):
  return cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
```

سپس با استفاده از تابع operator توابع خود را اجرا میکنیم. در این تابع نام تصویر به عنوان ورودی دریافت میشود سپس تصویر را بارگذاری کرده و تصویر سطح خاکستری آن محاسبه میشود. سپس هیستوگرام آن محاسبه میشود:

```
def operator(name):
  image = load_data(name)
  image_gray = BGR2GRAY(image)
  histg = cv2.calcHist([image_gray] , [0] , None , [255] , [0,255])

  plt.bar(np.arange(255), np.squeeze(histg))
  plt.show()
```

سپس الگوریتم اوتسو را اجرا میکنیم. در این الگوریتم ابتدا برای تمام سطوح خاکستری موجود محاسبات را انجام میدهیم. ابتدا هیستوگرام را به ۲ دسته تقسیم میکنیم. سپس احتمال را برای هر دو دسته محاسبه میکنیم. در مرحله بعد میانگین و واریانس را محاسبه میکنیم. در مرحله اخر با محاسبات به دست آمده و فرمول زیر مقدار میانگین $\mu_T$ را محاسبه میکنیم. سپس این مقدار را درون ارایه قرار میدهیم. این کار را برای تمام سطوح انجام میدهیم. کد الگوریتم به صورت زیر میباشد :

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T$$
$$\omega_0 + \omega_1 = 1$$

```
within = []

for i in range(len(histg)):
  x , y = np.split(histg , [i])
  x1 = np.sum(x)/(image.shape[0] * image.shape[1])
  y1 = np.sum(y)/(image.shape[0] * image.shape[1])

  x2 = np.sum([j*t for j,t in enumerate(x)])/np.sum(x)
  y2 = np.sum([j*t for j,t in enumerate(y)])/np.sum(y)

  x3 = np.sum([(j-x2)**2*t for j,t in enumerate(x)])/np.sum(x)
  x3 = np.nan_to_num(x3)
  y3 = np.sum([(j-y2)**2*t for j,t in enumerate(y)])/np.sum(y)
  y3 = np.nan_to_num(y3)
  within.append(x1*x3 + y1*y3)
```

در مرحله بعد از مقادیر به دست آمده برای $\mu_T$ استفاده کرده و کمترین مقدار آن را پیدا میکنیم. شماره خانه ای که در آن کمترین مقدار قرار دارد برابر با محلی است که مقدار Threshold قرار دارد.

```
m = np.argmin(within)

algoTime = time.time() - start
print("Time : " , algoTime)
print("Threshold : " , m)

(thresh , mask) = cv2.threshold(image_gray , m , 255 , cv2.THRESH_BINARY)
cv2_imshow(mask)
```

این مقدار را وارد cv2.threshold کرده و ماسک مربوط به این تصویر را به دست می آوریم. سپس با استفاده از این ماسک تصویر را به روی تصویر اصلی محاسبه میکنیم. همچنین مقدار زمان و عدد TH را چاپ میکنیم.

*در ادامه برای رسیدن به تصویر بهتر از یک کرنل استفاده میکنیم که نویز را از بین میبرد و تصویر را بهتر نمایش میدهد که تصویر به دست آمده خارج از خواست سوال است.

```
segmented_img = cv2.bitwise_and(image, image, mask=mask)
cv2_imshow(segmented_img)

# define kernel size
kernel = np.ones((7, 7), np.uint8)
# Remove unnecessary noise from mask
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

segmented_img_kernel = cv2.bitwise_and(image, image, mask=mask)
cv2_imshow(segmented_img_kernel)

return segmented_img , segmented_img_kernel
```

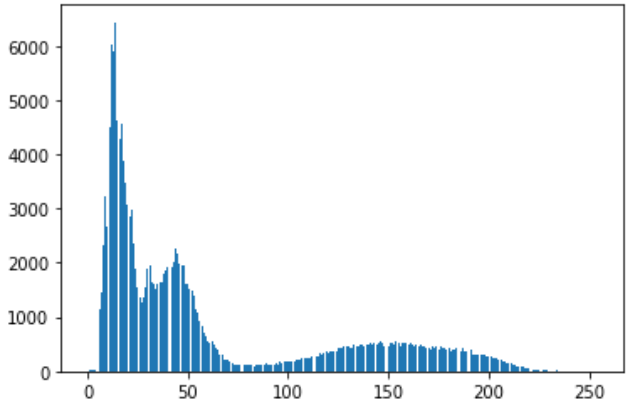
در ادامه با استفاده از تابع Accuracy مقدار صحتی برای تصاویر به دست می آوریم که به صورت زیر میباشد:

```
def accuracy(inputs, target):
  first = inputs.reshape(inputs.shape[0] * inputs.shape[1])
  second = target.reshape(inputs.shape[0] * inputs.shape[1])
  truth = 0
  for i in range(first.shape[0]):
    if (first[i] == second[i]).all():
      truth += 1

  return truth / first.shape[0]
```

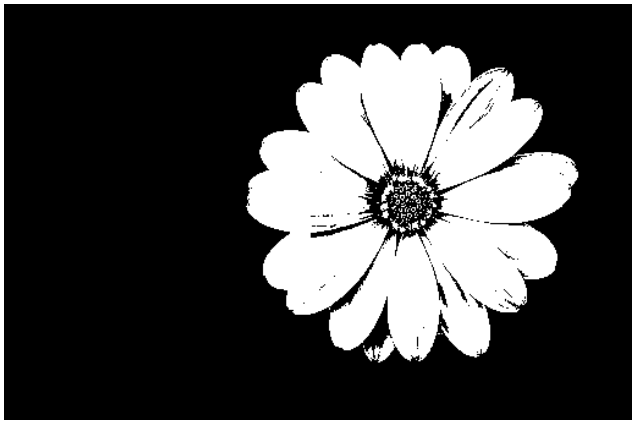
در ادامه به بررسی این الگوریم به روی تصاویر میپردازیم:

تصویر اولیه به صورت زیر میباشد :

هستوگرام تصویر به صورت زیر میباشد :




این تصویر را وارد الگوریتم کرده و مقدار ماسک آن به صورت زیر میباشد :



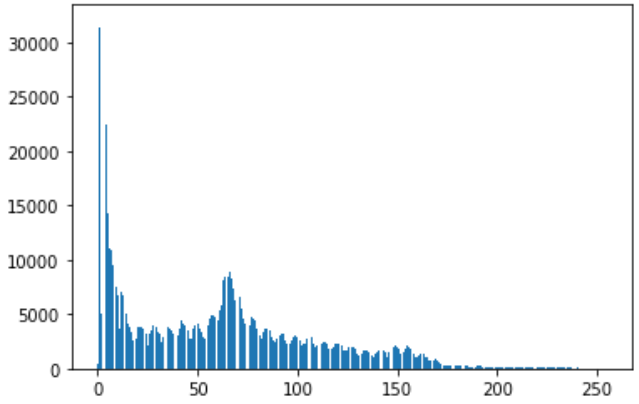
که نتیجه این تصویر به روی تصویر اصلی به صورت زیر میباشد:

در ادامه از فیلتر به روی آن استفاده کرده تا به تصویر زیر برسیم :

در ادامه به بررسی این الگوریم به روی تصویر دوم میپردازیم:

تصویر اولیه به صورت زیر میباشد :

هستوگرام تصویر به صورت زیر میباشد :




این تصویر را وارد الگوریتم کرده و مقدار ماسک آن به صورت زیر میباشد :

که نتیجه این تصویر به روی تصویر اصلی به صورت زیر میباشد:

در ادامه از فیلتر به روی آن استفاده کرده تا به تصویر زیر برسیم :

.۲

ابتدا به توضیح روش تکراری میپردازیم . در این روش ابتدا ۴ پیکسل کناری تصویر را به عنوان تصویر زمینه در نظر میگیریم و باقی تصویر را به عنوان تصویر اصلی در نظر میگیریم. مقدار میانگین را برای تصویر زمینه و اصلی به دست می آوریم و مقدار آن ها را در μ0 و μ1 قرار میدهیم. سپس مقدار میانگین این ۲ مقدار را به عنوان T در نظر میگیریم و با استفاده از این مقدار پیکسل ها را به ۲ دسته زمینه و اصلی در نظر میگیریم و مراحل را از ابتدا محاسبه میکنیم. این کار را انقدر انجام میدهیم تا مقدار T ثابت شود.

ابتدا کتابخانه ها و توابع مورد نیاز برای بارگذاری تصاویر را پیاده سازی میکنیم :

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import time
```

```
def load_data(name):
  image = cv2.imread(name)
  cv2_imshow(image)
  return image
```

```
def BGR2GRAY(image):
  return cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
```

سپس با استفاده از تابع operator توابع خود را اجرا میکنیم. در این تابع نام تصویر به عنوان ورودی دریافت میشود سپس تصویر را بارگذاری کرده و تصویر سطح خاکستری آن محاسبه میشود. سپس هیستوگرام آن محاسبه میشود:

```
def operator(name):
  image = load_data(name)
  image_gray = BGR2GRAY(image)
  histg = cv2.calcHist([image_gray] , [0] , None , [255] , [0,255])

  plt.bar(np.arange(255), np.squeeze(histg))
  plt.show()
```

سپس الگوریتم تکراری را اجرا میکنیم.

```
start = time.time()
corners = [image_gray[0 , 0] , image_gray[image_gray.shape[0]-1 , 0] , image_gray[0 ,
 image_gray.shape[1]-1] , image_gray[image_gray.shape[0]-1 , image_gray.shape[1]-1]]
mu_B = np.sum(corners) / 4

new_image = image_gray.copy()

new_image[0,0] = 0
new_image[image_gray.shape[0]-1 , 0] = 0
new_image[0 , image_gray.shape[1]-1] = 0
new_image[image_gray.shape[0]-1 , image_gray.shape[1]-1] = 0

new_image_sum = np.sum(new_image)
pixels = new_image.shape[0] * new_image.shape[1]
mu_O = new_image_sum / pixels

T = (mu_B + mu_O)/2
```

ابتدا ۴ پیکسل کناری تصویر را به عنوان تصویر زمینه در نظر میگیریم و باقی تصویر را به عنوان تصویر اصلی در نظر میگیریم. مقدار میانگین را برای تصویر زمینه و اصلی به دست می آوریم و مقدار آن ها را در μ0 و μ1 قرار میدهیم. سپس مقدار میانگین این ۲ مقدار را به عنوان T در نظر میگیریم و با استفاده از این مقدار پیکسل ها را به ۲ دسته زمینه و اصلی در نظر میگیریم. این کار را انقدر انجام میدهیم تا مقدار T ثابت شود.

```
while(True):
  B = []
  O = []
  for item in new_image:
    if item >= T :
      O.append(item)
    else:
      B.append(item)

  mu_B = np.sum(B) / len(B)
  mu_O = np.sum(O) / len(O)

  if (mu_B + mu_O)/2 == T:
    break
  else:
    T = (mu_B + mu_O)/2

algoTime = time.time() - start
print("Time : " , algoTime)
print("Threshold :" , T)
```

این مقدار را وارد cv2.threshold کرده و ماسک مربوط به این تصویر را به دست می آوریم. سپس با استفاده از این ماسک تصویر را به روی تصویر اصلی محاسبه میکنیم. همچنین مقدار زمان و عدد TH را چاپ میکنیم.

*در ادامه برای رسیدن به تصویر بهتر از یک کرنل استفاده میکنیم که نویز را از بین میبرد و تصویر را بهتر نمایش میدهد که تصویر به دست آمده خارج از خواست سوال است.

```
segmented_img = cv2.bitwise_and(image, image, mask=mask)
cv2_imshow(segmented_img)

# define kernel size
kernel = np.ones((7, 7), np.uint8)
# Remove unnecessary noise from mask
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

segmented_img_kernel = cv2.bitwise_and(image, image, mask=mask)
cv2_imshow(segmented_img_kernel)

return segmented_img , segmented_img_kernel
```

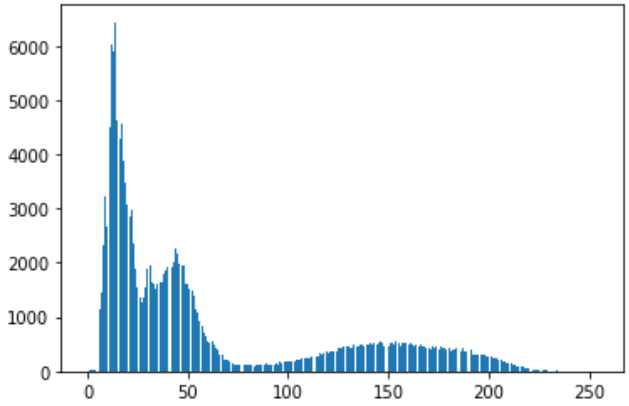
در ادامه با استفاده از تابع Accuracy مقدار صحتی برای تصاویر به دست می آوریم که به صورت زیر میباشد:

```
def accuracy(inputs, target):
  first = inputs.reshape(inputs.shape[0] * inputs.shape[1])
  second = target.reshape(inputs.shape[0] * inputs.shape[1])
  truth = 0
  for i in range(first.shape[0]):
    if (first[i] == second[i]).all():
      truth += 1

  return truth / first.shape[0]
```

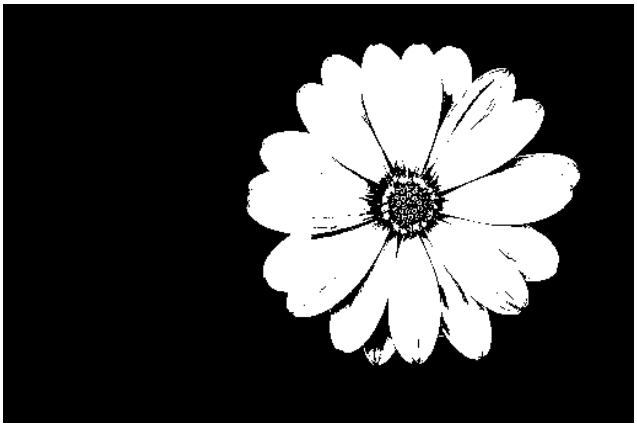
در ادامه به بررسی این الگوریم به روی تصاویر میپردازیم:

تصویر اولیه به صورت زیر میباشد :

هستوگرام تصویر به صورت زیر میباشد :




این تصویر را وارد الگوریتم کرده و مقدار ماسک آن به صورت زیر میباشد :



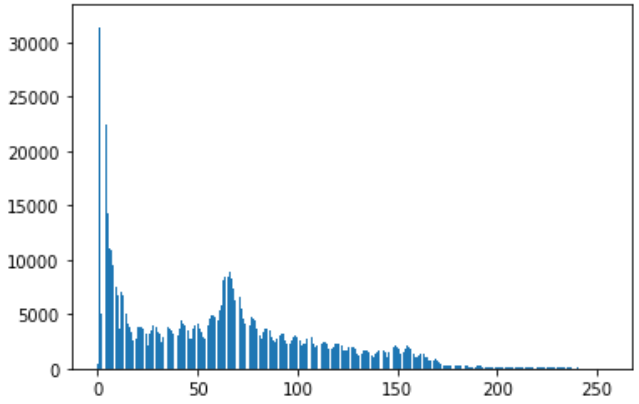
که نتیجه این تصویر به روی تصویر اصلی به صورت زیر میباشد:

در ادامه از فیلتر به روی آن استفاده کرده تا به تصویر زیر برسیم :

در ادامه به بررسی این الگوریم به روی تصویر دوم میپردازیم:

تصویر اولیه به صورت زیر میباشد :

هستوگرام تصویر به صورت زیر میباشد :




این تصویر را وارد الگوریتم کرده و مقدار ماسک آن به صورت زیر میباشد :

که نتیجه این تصویر به روی تصویر اصلی به صورت زیر میباشد:

در ادامه از فیلتر به روی آن استفاده کرده تا به تصویر زیر برسیم :

.۳

در این سوال به بررسی ۲ الگوریتم میپردازیم. ابتدا برای تصویر اول :

| نام الگوریتم | TH | زمان | دقت |
|---|---|---|---|
| Utso – no filter | 92 | 0.44 | 0.902 |
| Utso | 92 | 0.44 | 0.919 |
| Iterative – no filter | 91.911 | 3.808 | 0.903 |
| Iterative | 91.911 | 3.808 | 0.919 |

سپس برای تصویر دوم :

| نام الگوریتم | TH | زمان | دقت |
|---|---|---|---|
| Utso – no filter | 74 | 0.44 | 0.626 |
| Utso | 74 | 0.44 | 0.645 |
| Iterative – no filter | 62.92 | 8.358 | 0.636 |
| Iterative | 62.92 | 8.358 | 0.647 |

با توجه به مقادیر به دست آمده ، قابل مشاهده است روش تکراری از لحاظ زمانی تر از روش اوتسو میباشد. در مقابل روش تکراری به مقدار بهتری از دقت رسیده است که این مقدار بسیار ناچیز بوده و مقدار بهبود چشمگیر نمیباشد.

در ادامه استفاده از فیلتر نیز باعث شده است تصویر به دست آمده مقداری نتیجه بهتری نسبت به مقدار بدون فیلتر را ایجاد نماید.

.۴

در ابتدا کتابخانه های مورد نیاز را قرار داده ایم. سپس تصویر اصلی و الگو را بارگذاری کرده و تصویر سطح خاکستری آن را به دست آورده ایم.

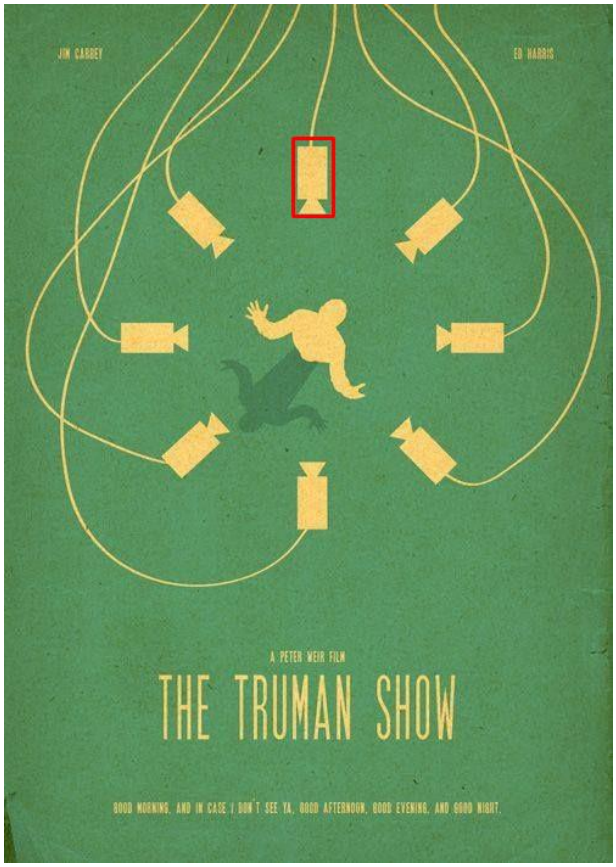
```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import time
```

```
img = cv2.imread("img.jpeg")
template = cv2.imread("template.jpeg")


# Convert it to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
```

در مرحله بعد یک کپی از تصویر اصلی گرفته ایم. سپس عرض و طول تصویر الگو را محاسبه کرده ایم. سپس تصویر سطح خاکستری اصلی و تصویر سطح خاکستری الگو را به تابع cv2.matchTemplate را میدهیم. سپس یک مقدار threshold برای تصویر الگو در نظر میگیریم و نقاطی که از این محدوده را در نظر میگیریم. سپس با استفاده از این نقاط یک مستطیل دور الگو یافت شده در تصویر کپی شده ایجاد میکنیم :

```
new_img = img.copy()
# Store width and height of template in w and h
w, h = template_gray.shape[::-1]

# Perform match operations.
res = cv2.matchTemplate(img_gray, template_gray, cv2.TM_CCOEFF_NORMED)


# Specify a threshold
threshold = 0.8

# Store the coordinates of matched area in a numpy array
loc = np.where(res >= threshold)
# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
  cv2.rectangle(new_img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)
  break

# Show the final image with the matched area.
cv2_imshow(new_img)
```

تصویر به دست آمده به صورت زیر میباشد :




همانطوری که قابل مشاهده است تابع استفاده فقط میتواند تصاویری که دقیقا با الگو هم زاویه است را به دست آورد که این اشکال این روش میباشد زیرا نسبت به زاویه و چرخش ضعف دارد.

برای رفع این مشکل از چندین راهکار متفاوت میتوان استفاده کرد. ما در این سری از تمارین از راهکار زیر استفاده میکنیم :

در این راهکار الگو را با زاویه های متفاوت ایجاد میکنیم . در این تمرین دوربین ها با زاویه های متفاوت ۴۵ درجه قرار دارند پس میتوانیم فقط الگوها با زاویه های ضریب ۴۵ درجه ایجاد کنیم. سپس این تصاویر با زاویه های متفاوت را وارد الگوریتم قسمت قبلی کرده و تصاویر را به دست می آوریم.

ابتدا لازم داریم تابعی بنویسیم که تصاویر را در زاویه های متفاوت بچرخواند. پس از تابع rotate_image استفاده میکنیم:

```
def rotate_image(image, angle):
    image_center = tuple(np.array(image.shape[1::-1]) / 2)
    rot_mat = cv2.getRotationMatrix2D(image_center, -angle, 1.0)
    result = cv2.warpAffine(image, rot_mat, image.shape[1::-1], flags=cv2.INTER_LINEAR)
    return result
```

در ادامه لازم داریم که این چرخش را برای تمام ضرایب ۴۵ درجه ایجاد کنیم پس تصاویر را در یک حلقه به دست می آوریم :

```
rot_range = [0,360]
rot_interval = 45
image_maxwh = img_gray.shape


res = []
for next_angle in range(rot_range[0], rot_range[1], rot_interval):
  if next_angle == 0:
      rotated_template = template_gray
  else:
      rotated_template = rotate_image(template_gray, next_angle)

  res.append([cv2.matchTemplate(img_gray, rotated_template, cv2.TM_CCOEFF_NORMED) , next_angle])
```

در ادامه برای رسم مستطیل لازم داریم تابعی ایجاد کنیم که این تابع با استفاده از درجه ورودی مستطیلی رسم نماید که تابع آن به صورت زیر میباشد. در این تابع از تابعی برای محاسبه مرکز مستطیل نیز استفاده شده است :

```
def findCenter(image , start_point, end_point , rotation):
  center_point = [(start_point[0]+end_point[0])//2, (start_point[1]+end_point[1])//2]
  height = end_point[1] - start_point[1]
  width = end_point[0] - start_point[0]
  angle = np.radians(rotation)

  return center_point , height , width , angle
```

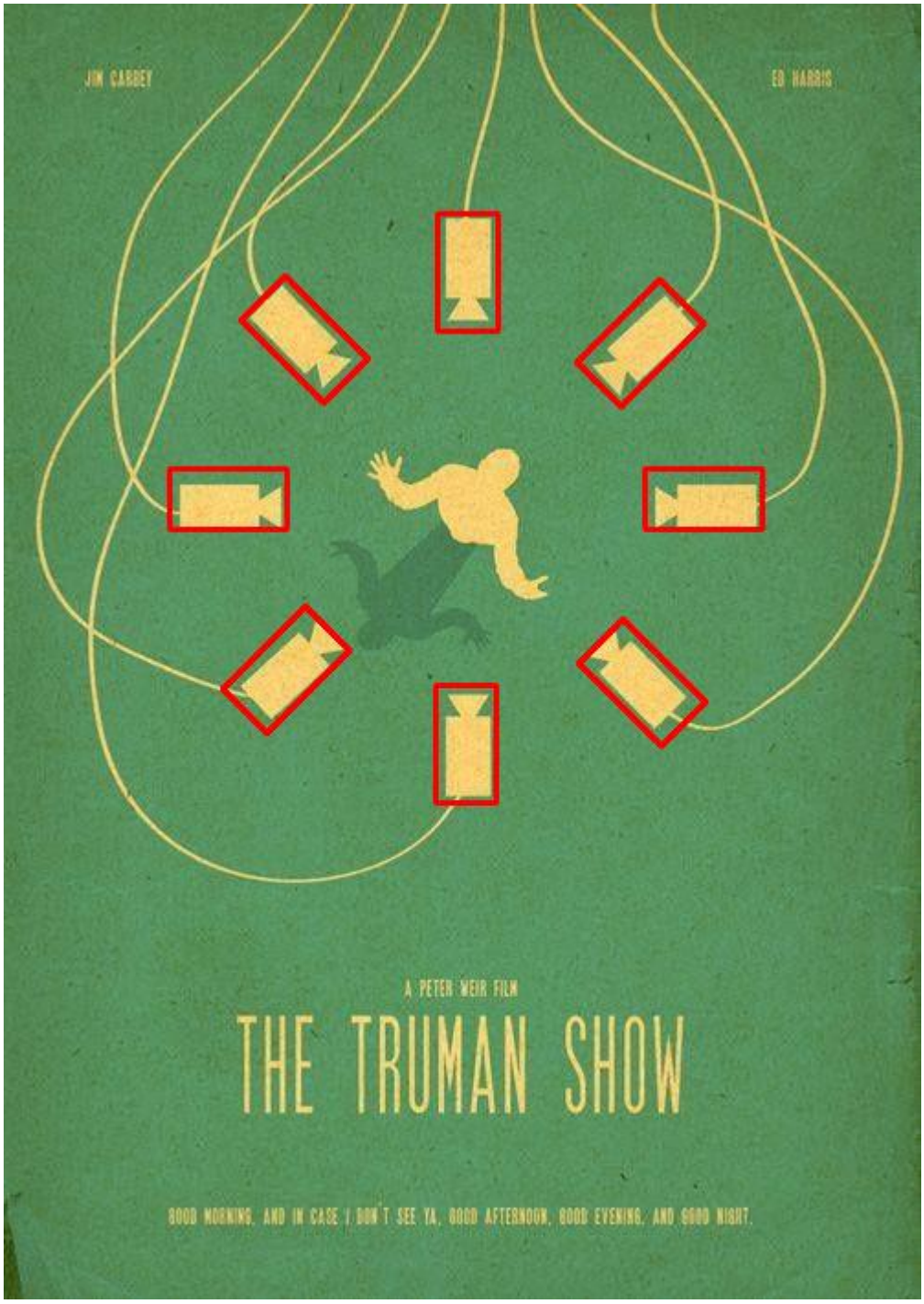
```
def rotated_rectangle(image, start_point, end_point, color, thickness, rotation=0):
    center_point , height , width , angle = findCenter(image, start_point, end_point , rotation)

    # Determine the coordinates of the 4 corner points
    rotated_rect_points = []
    x = center_point[0] + ((width / 2) * cos(angle)) - ((height / 2) * sin(angle))
    y = center_point[1] + ((width / 2) * sin(angle)) + ((height / 2) * cos(angle))
    rotated_rect_points.append([x,y])
    x = center_point[0] - ((width / 2) * cos(angle)) - ((height / 2) * sin(angle))
    y = center_point[1] - ((width / 2) * sin(angle)) + ((height / 2) * cos(angle))
    rotated_rect_points.append([x,y])
    x = center_point[0] - ((width / 2) * cos(angle)) + ((height / 2) * sin(angle))
    y = center_point[1]- ((width / 2) * sin(angle)) - ((height / 2) * cos(angle))
    rotated_rect_points.append([x,y])
    x = center_point[0] + ((width / 2) * cos(angle)) + ((height / 2) * sin(angle))
    y = center_point[1] + ((width / 2) * sin(angle)) - ((height / 2) * cos(angle))
    rotated_rect_points.append([x,y])
    cv2.polylines(image, np.array([rotated_rect_points], np.int32), True, color, thickness)
```

در ادامه نیاز داریم الگو های به دست آمده را وارد تابع کرده و تصاویر را برای آن ها به دست آوریم. در استفاده از این تابع ممکن است تعدادی مستطیل با مرکز های متفاوت به دست آید که نیاز داریم از میانگین این مستطیل ها استفاده کنیم . پس در این کد مستطیل هایی که به روی هر دوربین ایجاد شده را در نظر گرفته و میانگین آن ها را برای مستطیل اصلی ایجاد میکنیم.

کد این بخش به صورت زیر میباشد :

```
new_img = img.copy()
# Store width and height of template in w and h
w, h = template_gray.shape[::-1]

Centers = []
for item , rotate in res :
  # Specify a threshold
  threshold = 0.8

  # Store the coordinates of matched area in a numpy array
  loc = np.where(item >= threshold)
  i = 0
  # Draw a rectangle around the matched region.
  for pt in zip(*loc[::-1]):
    times = 0
    flag = True
    # Centers.append([pt, rotate])
    center , height , width , angle = findCenter(new_img, pt, (pt[0] + w, pt[1] + h) , rotate)
    for i , point in enumerate(Centers):
      center_point , height , width , angle = findCenter(new_img, point[0], (point[0][0] + w, point[0][1] + h) , point[1])

      if center[0] <= center_point[0] + width and center[0] >= center_point[0] - width:
        if center[1] <= center_point[1] + height and center[1] >= center_point[1] - height:
          if point[2] < 200:
            Centers[i] = [((point[0][0]+pt[0]) / 2 , (point[0][1]+pt[1]) / 2) ,point[1] ,point[2]+1]
          flag = False
          break

    if flag:
      Centers.append([(pt[0] , pt[1]), rotate , times])

for item in Centers:
  pt = item[0]
  rotated_rectangle(new_img , pt, (pt[0] + w, pt[1] + h) , (0, 0, 255), 2 , item[1])

# # Show the final image with the matched area.
cv2_imshow(new_img)
```

تصویر به دست آمده از این کد به صورت زیر میباشد :




با تشکر

مهدی فیروزبخت