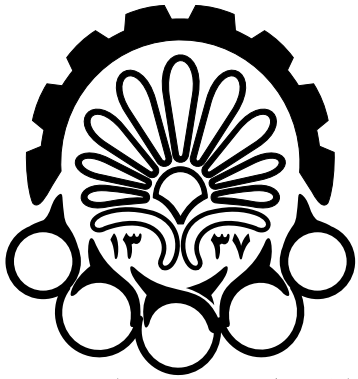


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس : بینایی کامپیوتر

نام و نام خانوادگی: فاطمه توکلی

تمرین شماره ۰۳

سئوال ۱:

در ابتدا الگوریتم را به صورت زیر پیداسازی کرده:

```
def otuso_thresh(img):
    hist = cv.calcHist([img],[0],None,[256],[0,256])
    hist_norm = hist.ravel()/hist.sum()
    Q = hist_norm.cumsum()
    bins = np.arange(256)
    fn_min = np.inf
    thresh = -1
    for i in range(1,256):
        p1,p2 = np.hsplit(hist_norm,[i])
        q1,q2 = Q[i],Q[255]-Q[i]
        if q1 < 1.e-6 or q2 < 1.e-6:
            continue

        b1,b2 = np.hsplit(bins,[i])

        m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
        v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2

        fn = v1*q1 + v2*q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
    print('Threshold value found using Otsu algorithm : '+ str( thresh))
    return thresh
```

سپس آن را بر روی دو تصویر زیر اعمال کرده و به همراه تصویر نهایی در ادامه آورده شده‌است:


```
Threshold value found using Otsu algorithm : 91
```

شکل ۱: تصویر اول انتخاب شده و مقدار آستانه به دست‌آمده با استفاده از الگوریتم otsu

شکل ۲: تغییر تصویر با اعمال آستانه بر روی پیکسل‌ها و جدا کردن پس زمینه


```
Threshold value found using Otsu algorithm : 100
```

شکل ۳: تصویر دوم انتخاب شده و مقدار آستانه به دست آمده

شکل ۴: تغییر تصویر با اعمال آستانه بر روی پیکسل‌ها و جدا کردن پس زمینه

سئوال ۲:

در ابتدا الگوریتم را به صورت زیر پیاده‌سازی کرده:

```
def itr_method(img):
    background = [img[0,0],img[0,-1],img[-1,0],img[-1,-1]]
    background_mue = np.mean(background)

    foreground = np.delete(img,[[0,0],[0,-1],[-1,-1],[-1,0]])
    foreground_mue = np.mean(foreground)

    prev_t = 0
    for i in range(200):
        t = (background_mue + foreground_mue) / 2

        background_1 = img[img < t]
        foreground_1 = img[img >= t]

        backgound_mue = np.mean(background_1)
        foreground_mue = np.mean(foreground_1)

        if t == prev_t:
            break
        prev_t = t
    print('Threshold value found using Otsu algorithm : '+ str( t))
    return t
```

سپس آن را بر روی عکس‌های قبلی اعمال کرده و تصاویر نهایی در زیر آمده است:

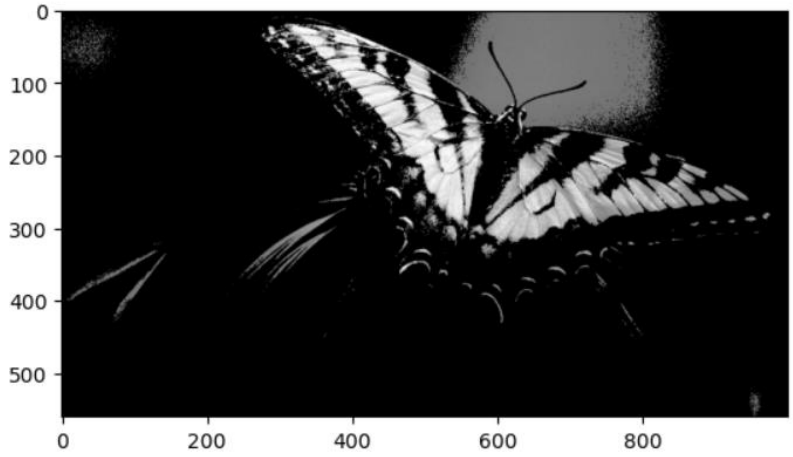

```
Threshold value found using Otsu algorithm : 82.45440347191509
```

شکل ۵ : تصویر اول انتخاب شده و مقدار آستانه به دست‌آمده با استفاده از الگوریتم iterative

شکل ۶: تغییر تصویر با اعمال آستانه بر روی پیکسل‌ها و جدا کردن پس زمینه


```
Threshold value found using Otsu algorithm : 110.9354150778952
```

شکل ۷ : تصویر دوم انتخاب شده و مقدار آستانه به دست‌آمده با استفاده از الگوریتم iterative



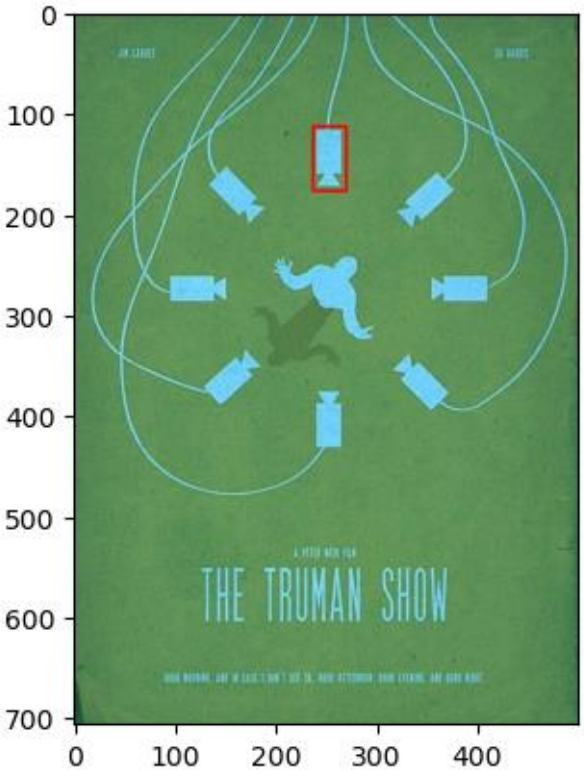
شکل ۸ : تغییر تصویر با اعمال آستانه بر روی پیکسل‌ها و جدا کردن پس زمینه

سئوال ۳:

نتایج آزمایش‌ها نشان می‌دهد که مقادیر آستانه برای روش تکراری کمی به سمت منطقه پیش‌زمینه تغییر می‌کند و در نتیجه اشیاء ضعیف را با وضوح بیشتری نسبت به روش Otsu برای هر یک از تصاویر شناسایی می‌کند. همچنین از خروجی های فوق مشاهده می شود که تقسیم بندی با استفاده از روش تکرار شونده از دقت بیشتری نسبت به روش Otsu برخوردار است. تصویر با استفاده از Iterative Otsu بهتر تقسیم بندی می شود و دقیق تر است. بنابراین، نتیجه گرفته شده این است که روش Iterative الگوریتم تقسیم بندی مبتنی بر آستانه بهتری در مقایسه با روش Otsu از نظر پیچیدگی زمانی و دقت است.

سئوال ۴:

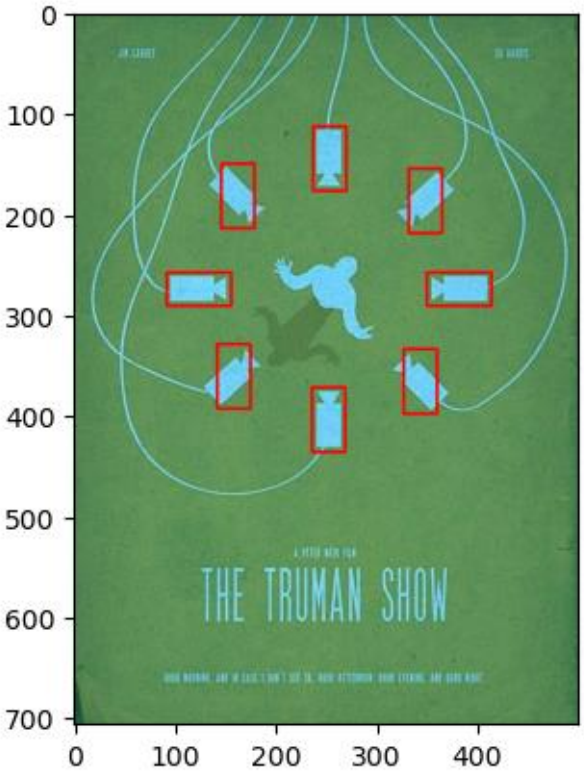
در قسمت اول سئوال خواسته شده که با استفاده از تابع cv.matchTemplate و کلیشه مورد نظر را روی تصویر تطبیق داده‌ایم و همانطور که در شکل ۹ دیده می‌شود تنها کلیشه هم جهت را توانسته پیدا کند و دیگر کلیشه‌ها با چرخش‌های متفاوت را نتوانسته تشخیص دهد:



شکل ۹ : تصویر حاصل از تطبیق کلیشه

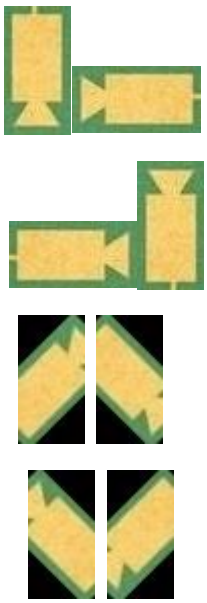
همانطور که مشاهده شد این روش در مقابل چرخش‌های مختلف مقاوم نیست و در صورتی که ملزم به استفاده از تابع matchTemplate هستیم تنها راه حل این است که کلیشه را در جهت های مختلف چرخانده و با استفاده از چرخش‌های مختلف کلیشه به دست آمده به تشخیص آن‌ها در تصویر پرداخت:

با استفاده از تابع imutils.rotate می‌توان کلیشه را به جهت‌های مختلف چرخاند و تطبیق را انجام داد. نتیجه نهایی در شکل ۱۰ آمده‌است:



شکل ۱۰: حاصل تطبیق مجموعی از کلیشه‌های به دست آمده

کلیشه های حاصل از چرخش به دست آمده :



شکل ۱۱: کلیشه‌ها با چرخش‌های به دست آمده

پایان.