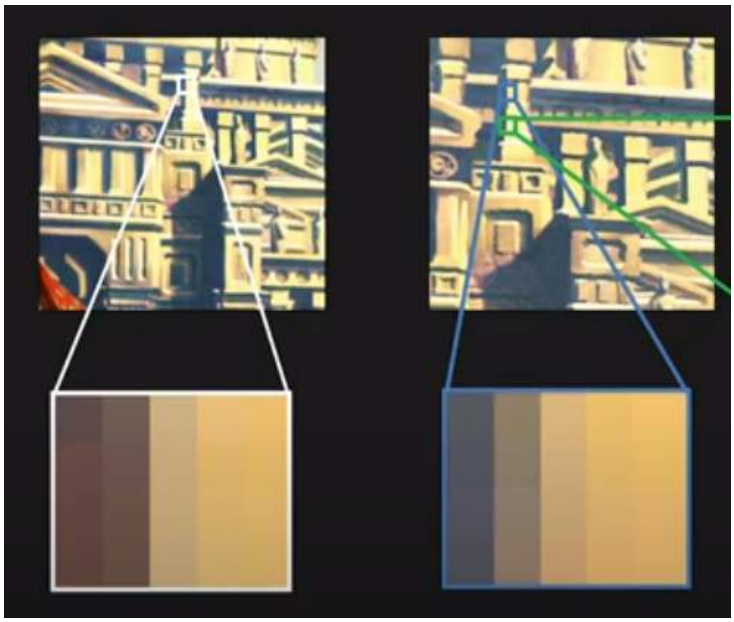
به نام خدا
مهدی فیروزبخت
تمرین سری پنجم
درس بینایی کامپیوتر

۱.
برای شناسایی اشیا در تصاویر راه های زیادی وجود دارد که هر چه تصویر پیچیده تر باشد راهکارها محدودتر میشوند. یکی از روش های عالی در این زمینه مدل SIFT که مخفف عبارت Scale Invariant Fourier Transform میباشد. در این مدل ابتدا ویژگی های unique را یافته که این ویژگی ها با نام نقاط مورد علاقه هستند سپس این نقاط یافته شده را به صورت هیستوگرامی از زوایای گرادیان ارائه می شود را درون نمونه را با نقاط درون تصویر اصلی انطباق میدهیم تا به بهترین نتیجه برسیم. در ادامه مراحل SIFT را بررسی میکنیم :

ابتدا بررسی میکنیم که نقاط مورد علاقه چیست؟
نقاط مورد علاقه یک اصطلاح کلی در بینایی کامپیوتر برای نقاطی در تصویر است که قابل شناسایی هستند و برای پردازش سطح بالاتر مرتبط هستند. این نقاط دارای ویژگی هایی هستند :
۱. اطراف این نقاط باید از نظر محتوایی غنی باشد (از نظر رنگ های متفاوت ، روشنایی های متفاوت و ...)
۲. باید یک ارائه مناسب برای مقایسه با سایر نقاط داشته باشد.
۳. موقعیت این نقطه در تصویر باید مشخص باشد.
۴. باید نسبت به چرخش ، بزرگنمایی و روشنایی حساس نباشد.
۵. باید گونه ای باشد که حالت Unique داشته باشد. برای مثال در یک تصویر ممکن است لبه ها مناسب نباشند. زیرا اطراف لبه ها به گونه ای است که حالت رنگ ها و روشنایی های یکسانی در بخش های مختلف تصویر از آن وجود دارد که این باعث میشود این نقطه حالت یکتا نداشته باشد. تصویر زیر نشان دهنده مفهوم توضیح داده شده است:



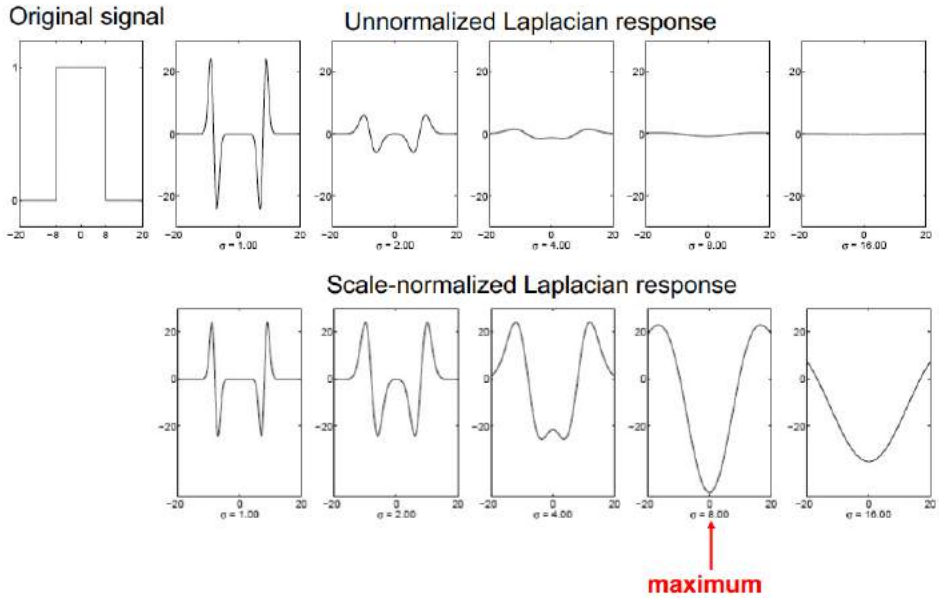
در ادامه نیاز داریم مفهوم Blob را توضیح دهیم :
حباب به یک قطعه اشاره دارد. تجزیه و تحلیل حباب ها اساسی ترین روش پردازش تصویر برای تجزیه و تحلیل ویژگی های شکل یک جسم، مانند وجود، تعداد، مساحت، موقعیت، طول و جهت قطعه ها است.
در پردازش تصاویر برای SIFT از حباب ها استفاده میکنیم. این حباب ها به عنوان نقاط مورد علاقه محاسبه میشوند.

مراحل بررسی حباب ها به صورت زیر میباشد:

۱. ابتدا یک نقطه مرکزی را می یابیم.
۲. یک سایز برای حباب در نظر میگیریم.
۳. یک زاویه برای حباب می یابیم.
۴. یک توضیح یا امضا برای حباب می یابیم که توصیف کننده این حباب میباشد.

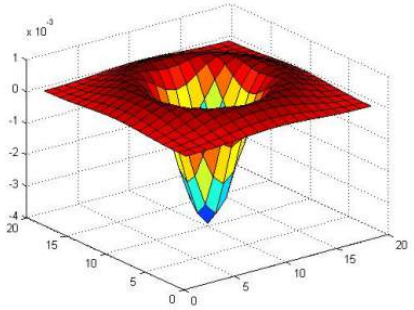
حال به مفهوم Blob detection میپردازیم :

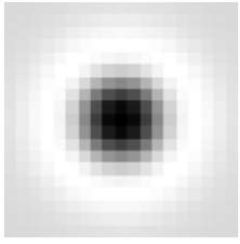
برای یافتن حباب ها مانند یافتن حباب عمل میکنیم. از مشتق دوم گاسین استفاده کرده و آن را در تصویر اصلی کانولوشن میکنیم اما برای بهبود نتیجه توان دوم سیگما را در آن ضرب میکنیم. سپس در محلی که مقدار حداکثر وجود دارد نقطه حباب است. در تصویر زیر علت ضرب توان دوم سیگما قابل مشاهده است :



این کارراهکاری است که برای تصاویر ۱ بعدی استفاده میشود. برای تصاویر ۲ بعدی از NLoG استفاده میکنیم. این فیلتر را با سیگما های متفاوت به تصویر اعمال میکنیم و یک حالت ۳بعدی ایجاد میشود که در هر لایه آن یک تصویری که فیلتر بر آن اعمال شده وجود دارد که با افزایش ارتفاع سیگما نیز افزایش می یابد. به این شکل ساخته شده scale-space میگویند.

$$\nabla^2_{\text{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$





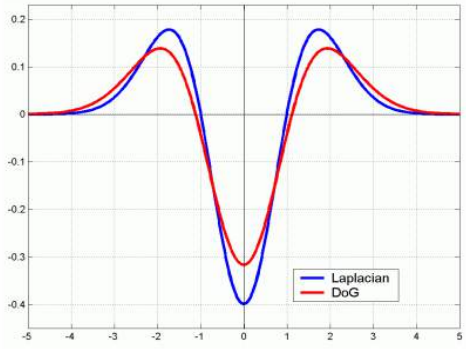
در مرحله بعد لازم است همسایگی های ۳*۳*۳ در نظر گرفت و نقاط ماکزیمم را در آن یافت. این نقاط ماکزیمم نقاطی هستند که با عنوان حباب در نظر گرفته میشوند. میتوان یک مقدار محدود کننده در نظر گرفت که حباب های قوی را فقط در نظر گرفت.
برای پیاده سازی NLoG میتوانیم از DoG استفاده کنیم. زیرا میتوانیم لاپلاسین را با استفاده از تفاضل گاسین ها تقریب زد:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

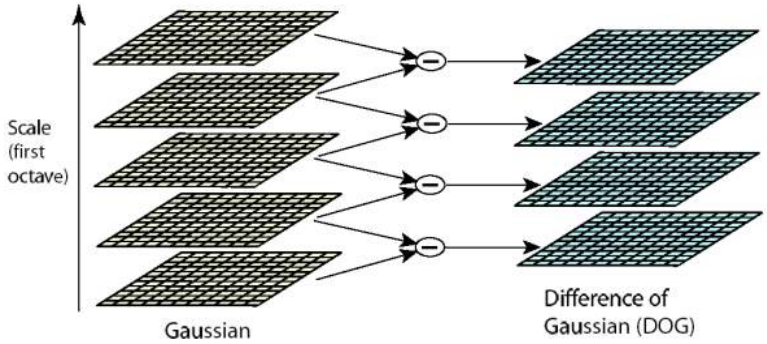
(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

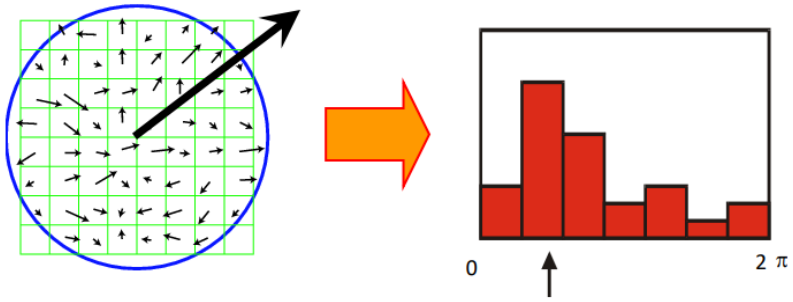


پس میتوانیم ابتدا به تصاویر فیلتر های مختلف گاسین را اعمال کنیم و سپس تفاضل ۲ به ۲ تصاویر به دست آمده را محاسبه کنیم. نتیجه به دست آمده همانند لاپلاسین گاسین خواهد بود.

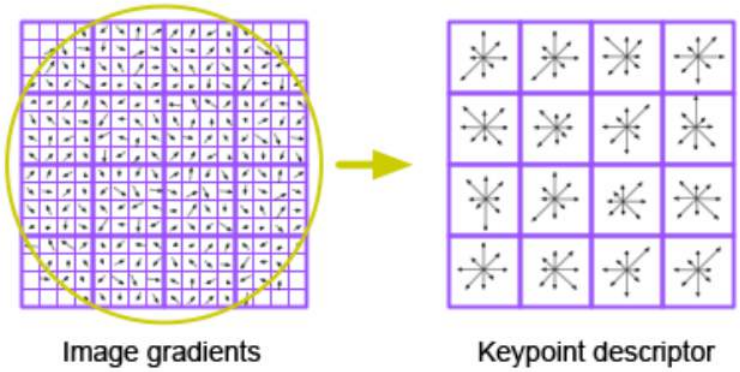


سپس مقادیر ماکزیمم را میابیم و نقاط به دست آمده همان نقاط مورد علاقه یا حباب ها میباشند. شعاع این حباب ها رابطه مستقیم با فیلتری است که برای به دست آمدن تصویر آن استفاده شده است.

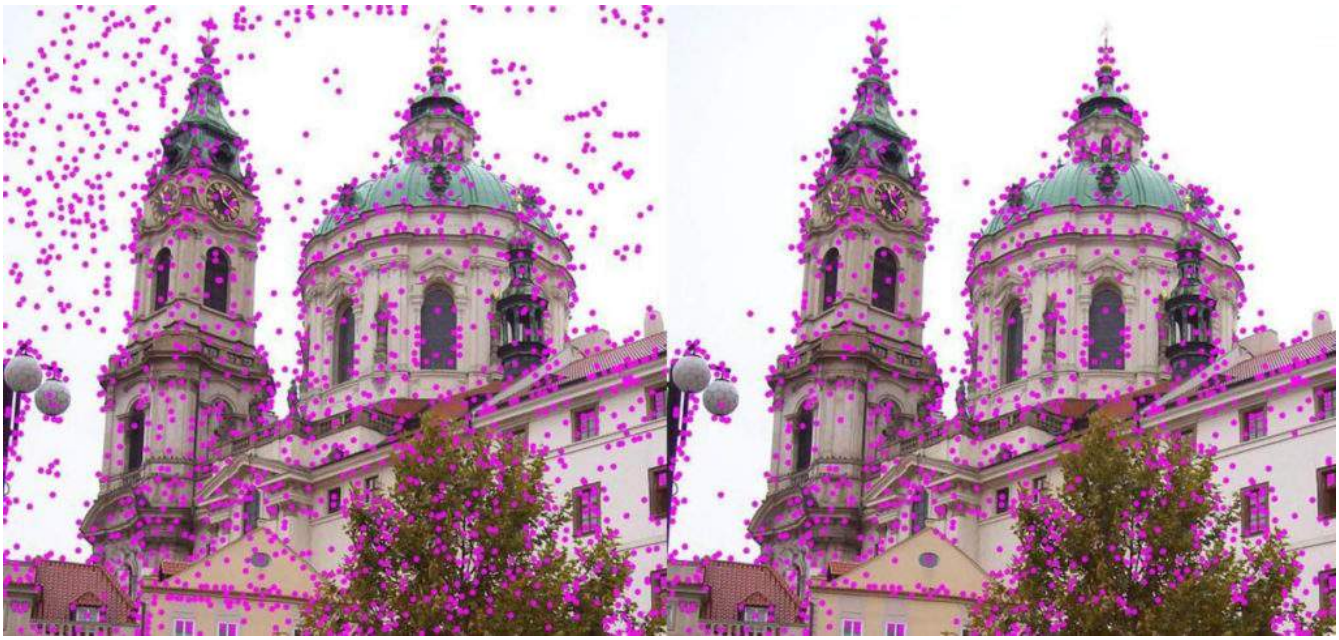
تصویر بالا نمونه ای از حباب هایی است که برای تصویر پروانه با استفاده از مدل ذکر شده ساخته میشود. برای معرفی جهت حباب نیازمند این هستیم تا درون هر حباب هیستوگرام جهت های گرادیان را بیابیم و جهتی که بیشترین تعداد را دارا میباشد جهت حباب خواهد بود.



حال حباب های ساخته شده نیازمند ارایه شدن هستند. روش های متفاوتی برای ارایه وجود دارد. یکی از روش ها این است که درون حباب را به چند منطقه تقسیم میکنیم. سپس برای هر منطقه هیستوگرام جهت گرادیان را میابیم و به صورت یک سیگنال در نظر گرفته و این سیگنال های مناطق را به یکدیگر پیوند میدهیم تا یک ارایه ای از جهت ها را ایجاد کند.



تصویر زیر نمونه ای از تصویری است که توسط SIFT توصیف شده است :



.۲
ابتدا کتابخانه های مورد نیاز را وارد میکنیم. سپس با تابع load_data را مینویسم. این تابع تصویر را گرفته ، بارگذاری کرده و حالت سطح خاکستری آن را به عنوان خروجی میدهد:

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from google.colab.patches import cv2_imshow
import random
import time
```

```
def load_data(image):
  img = cv2.imread(image , 1)
  gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  return img , gray
```

سپس با استفاده از تابع rotate_image تصویر را میچرخانیم. در این حالت زاویه چرخش را به عنوان ورودی دریافت میکنیم و با استفاده از تابع getRotationMatrix2D و warpAffine تصویر را میچرخانیم.
در تابع create_rotated_images نیز درجه های رندوم را محاسبه میکنیم :

```
def rotate_image(image , angle , scale):
  # Rotating the image after Warp
  center = (image.shape[1]//2, image.shape[0]//2)
  rot_mat = cv2.getRotationMatrix2D( center, angle, scale )
  warp_rotate_dst = cv2.warpAffine(image, rot_mat, (image.shape[1], image.shape[0]))
  return warp_rotate_dst
```

```
def create_rotated_images(image , gray):
  random_angle = random.randint(0,360)
  rotated_image = rotate_image(image , random_angle , 0.5)
  rotated_gray_image = rotate_image(gray , random_angle , 0.5)

  return rotated_image , rotated_gray_image , random_angle
```

در تابع بعدی که تابع operate_SIFT میباشد ابتدا با استفاده از تابع SIFT_create یک مدل SIFT ساخته و سپس با استفاده از detectAndCompute نقاط را یافته و فاصله را محاسبه میکنیم.

```
def operate_SIFT(image , gray):
  # Applying SIFT detector
  sift = cv2.xfeatures2d.SIFT_create()
  kp , des = sift.detectAndCompute(gray, None)

  # Marking the keypoint on the image using circles
  img=cv2.drawKeypoints(gray ,
                        kp ,
                        image ,
                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

  return img , kp , des
```

در تابع بعدی نیاز داریم تا حباب های به دست آمده را در نمونه و تصویر اصلی با یکدیگر مقایسه کرده و شبیه ترین ها را به عنوان مشابه در نظر بگیریم.

برای این کار از مدل FlannBasedMatcher استفاده میکنیم. در ادامه از تابع knnMatch استفاده کرده و نزدیک ترین نقاط را میابیم. در ادامه نقاط به دست آمده را محدود میکنیم و نقاطی که از یک محدوده نزدیک تر هستند را در نظر میگیریم.

```
def find_matches(des_main , des_rotated):
  FLANN_INDEX_KDTREE = 1
  index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
  search_params = dict(checks = 50)

  flann = cv2.FlannBasedMatcher(index_params, search_params)
  matches = flann.knnMatch(des_main,des_rotated,k=2)
  # store all the good matches as per Lowe's ratio test.
  good = []
  for m,n in matches:
      if m.distance < 0.1*n.distance:
          good.append(m)

  return good , len(matches)
```

در ادامه با استفاده از کد زیر نقاطی که به دست آمده را از هر دو تصویر در نظر میگیریم . سپس با استفاده از findHomography میتوانیم Homography matrix و مقدار ماسک که شامل مقادیر inliers است.

```
MIN_MATCH_COUNT = 10
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ KP_main[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ KP_rotated[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
    h,w = rotated_gray_image.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,M)
    img2 = cv2.polylines(rotated_gray_image,[np.int32(dst)],True,255,3, cv2.LINE_AA)
else:
    print( "Not enough matches are found - {}/{}".format(len(good), MIN_MATCH_COUNT) )
    matchesMask = None


draw_params = dict(matchColor = (0,0,255), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)
```

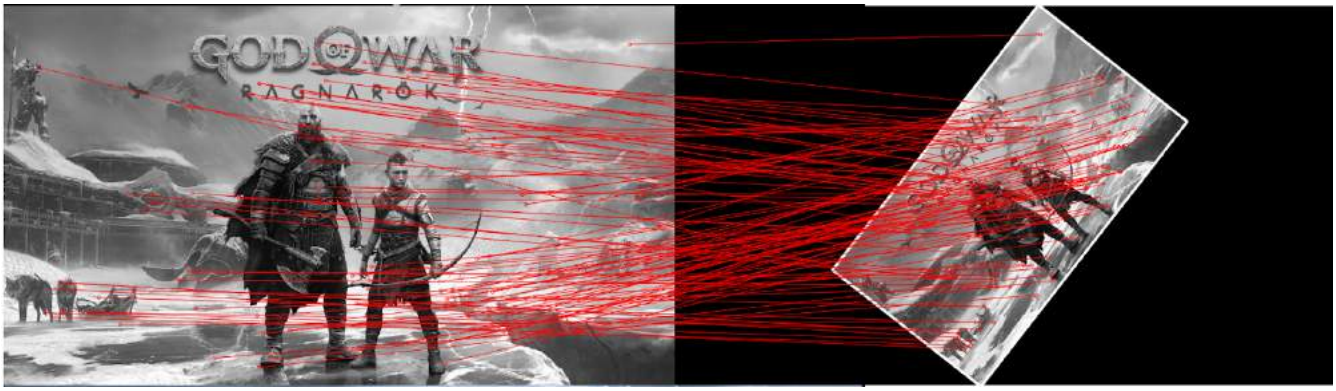
سپس در مرحله آخر نیز با استفاده از ماتریس homography میتوانیم تصویر دوم را به تصویر اول بچرخانیم:

```
img3 = cv2.drawMatches(gray,KP_main,rotated_gray_image,KP_rotated,good,None,**draw_params)
end = time.time() - start
cv2_imshow(img3)
```

حال نتیجه به دست آمده از کد های بیان شده برای ۳ درجه متفاوت در ادامه قرار داده شده است :

.۱
ابتدا چرخش لازم را ایجاد میکنیم:

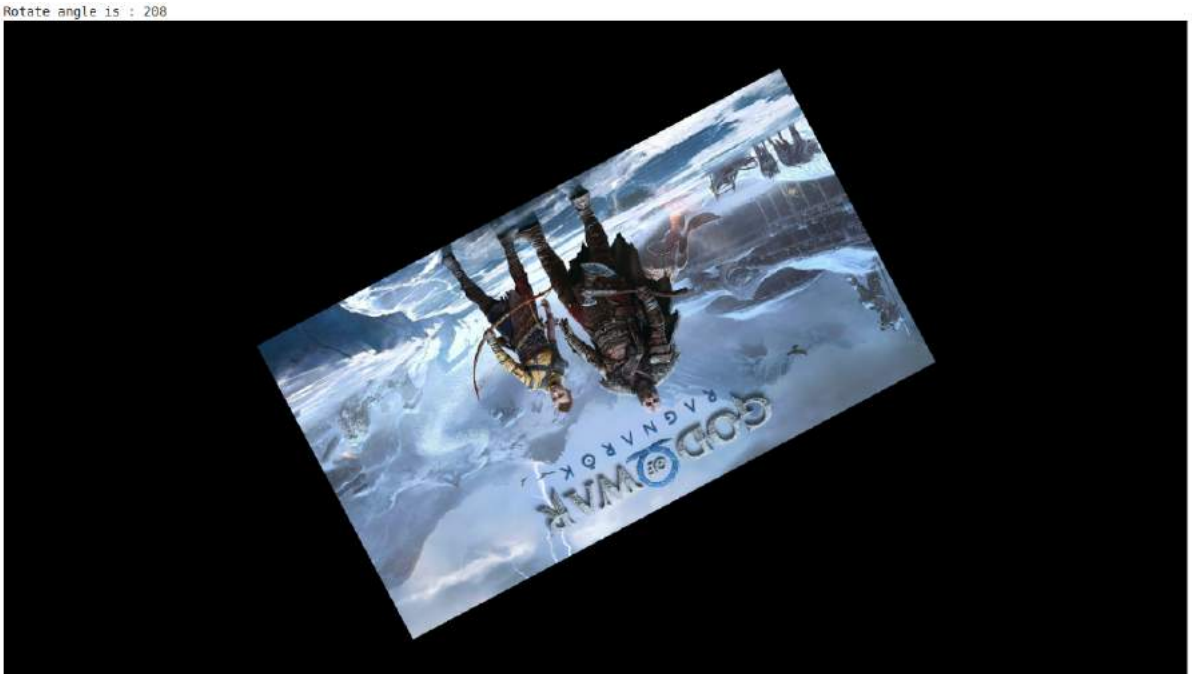
سپس نتیجه الگوریتم :

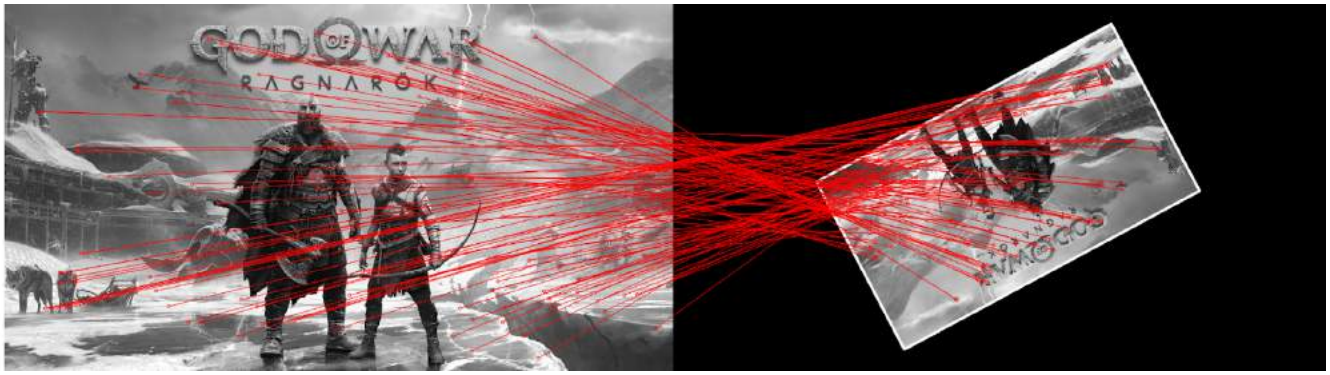


تصویر چرخش داده شده توسط الگوریتم :

.۲

.۳





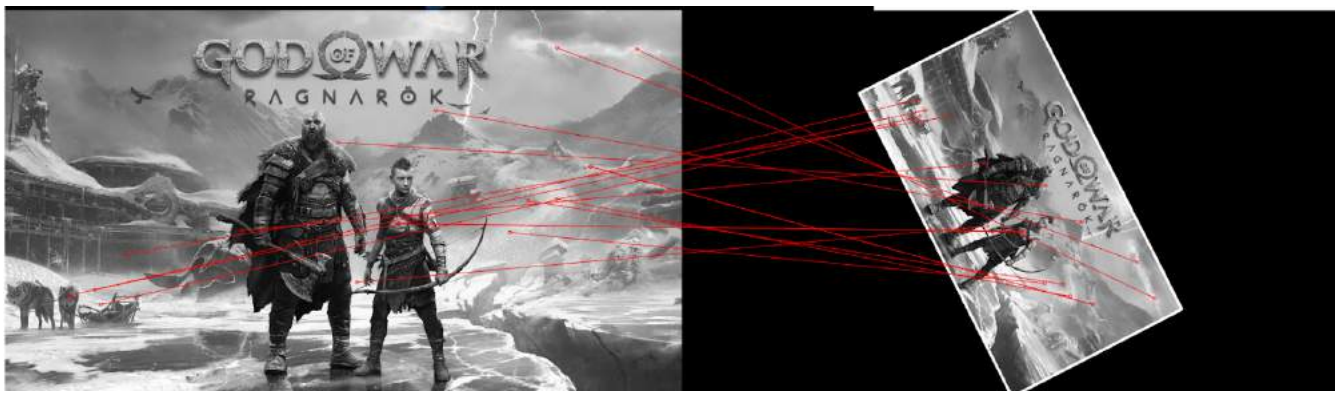
۳. برای حالت FREAK نیز تمام مراحل بالا ایجاد میشود و فقط تابع operate_FREAK به جای حالت SIFT استفاده میشود.

```python
def operate_FREAK(image , gray):
  # Applying FREAK detector
  freak = cv2.xfeatures2d.FREAK_create()
  fast = cv2.xfeatures2d.SIFT_create()
  # Find the keypoints
  keypoints = fast.detect(gray, None)
  kp , des = freak.compute(gray, keypoints)

  # Marking the keypoint on the image using circles
  img=cv2.drawKeypoints(gray ,
                        kp ,
                        image ,
                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

  return img , kp , des
```

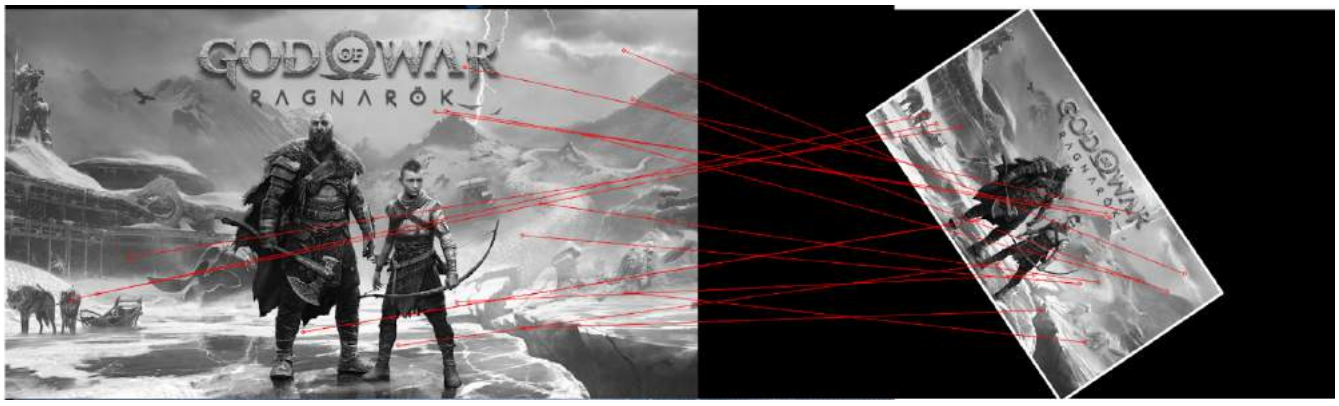
در این حالت تابع FREAK قابلیت استخراج نقاط کلیدی را ندارد پس با استفاده از تابع SIFT نقاط کلیدی را ایجاد میکنیم و با FREAK توصیف میکنیم.

نتیجه به دست آمده به صورت زیر است :

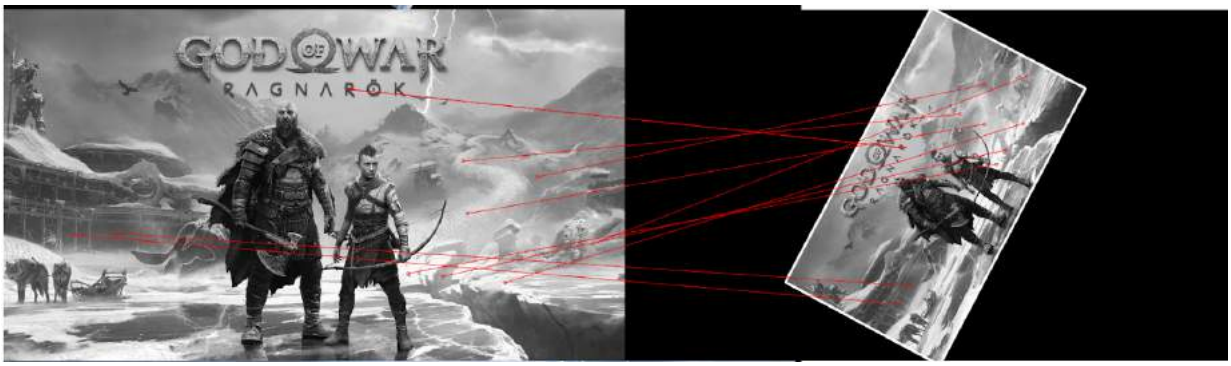
۱.



.۲



.۳




GOD OF WAR
RAGNARÖK

حال به بررسی سرعت و عملکرد ۲ مدل میپردازیم :
ابتدا مدل SIFT:
تصویر اول :
Size of all matches : 4589
Size of best matches : 119
SIFT time : 1.1664793491363525

تصویر دوم :
Size of all matches : 4589
Size of best matches : 101
SIFT time : 0.9552059173583984

تصویر سوم :
Size of all matches : 4589
Size of best matches : 116
SIFT time : 1.0316267013549805

و مدل FREAK :
تصویر اول :
Size of all matches : 4227
Size of best matches : 17
SIFT time : 0.7988717555999756

تصویر دوم :
Size of all matches : 4227
Size of best matches : 17
SIFT time : 0.7693140506744385

تصویر سوم :
Size of all matches : 4227
Size of best matches : 11
SIFT time : 0.7722818851470947

با توجه به نتایج به دست آمده از ۲ مدل ، قابل مشاهده است که مدل FREAK سرعت بهتری نسبت به مدل SIFT دارا میباشد. از نظر عملکرد مدل اول ۴۵۸۹ داده مچ پیدا کرده است در مقایسه با مدل دوم که دارای ۴۲۲۷ نقطه میباشد مدل بهتری است. در این حالت با محدود کردن هر دو مدل ، مدل اول را بسیار محدودتر کرده ایم اما نتایج به دست آمده برای آن نقاط بیشتری را دارد به این معنی که نقاطی که در این مدل پیدا شده اند از نظر شباهت بهتر از مدل دوم هستند. از نظر دقت هر دو مدل نقاط نظیر درستی را ایجاد کرده اند.
در مورد مقاومت نسبت به چرخش هر دو روش نتیجه مناسبی به دست آورده اند اما در حالت های مختلف و برای چرخش به حالت اولیه مدل اول یعنی SIFT بهتر از مدل دوم یعنی FREAK عمل کرده است به اینگونه که مدل اول حتما میتواند تصویر را به حالت اولیه برگرداند اما در مدل دوم گاهی در مورد چرخش اشتباه کرده و درست عمل نمیکند.