Readme

This the notebook is for Li Bin Song's MM805 Assignment.

The source code is located at github. Or you can unzip the zip from eclass.

Requirement to run this notebook

The commands shows here are based on Linux(Ubuntu)

1. Python 3.8
2. Setup Python virtual environment

```
python3.8 -m venv .venv
 . .venv/bin/activate
 pip install --upgrade pip
```

3. Install required packages `pip install -r requirements.txt`
4. Open `report.ipynb` , select python kernel and run

Q1. (40 points) Feature extraction and matching

1a. Harris Corner Point detection

The feature I implemented is Harris corner point. The logic is based on wikipedia

Answer:

The key function is `get_harris_points` , the detail code can be found in assignment_code.py ane extra function `display_corner_points` is written for display finding point on image as red point.

The main logic are:

1. convert image to grayscale color

2. calculate Ix, Iy spatial derivative
3. structure Matrix A
4. Harris response calculation(calcualte Det(A) and Trace(A))
5. Based on the response choose corner points

In [ ]:

```
import numpy as np
import cv2
from pprint import pprint
from scipy import ndimage
from skimage.feature import corner_peaks
from scipy import signal



def imfilter(I, filter)->np.ndarray:
    # I_f = ndimage.filters.correlate(I, weights=filter, mode='constant')
    I_ =  signal.convolve2d(I, filter, boundary='symm', mode='same')
    return I_


def harris_corner_points(I, k=0.05,top_n_points = 5000):

    # check image and convert to gray and normalize
    if len(I.shape) == 3 and I.shape[2] == 3:
        I = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
    if I.max() > 1.0:
        I = I / 255.0

    # Step 1 calcualte Axx, Axy and Ayy

    # Step 1.1 calculate Ix, Iy
    # apply soble filter for quick calculation
    filter_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) / \
        8.0  # sobel filter for x derivative
    filter_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]) / \
        8.0  # sobel filter for y derivative

    Ix = imfilter(I, filter_x)
    Iy = imfilter(I, filter_y)
    Ixx = Ix * Ix
    Iyy = Iy * Iy
    Ixy = Ix * Iy
```

```
    window = np.ones((3, 3))

    Axx = imfilter(Ixx, window)
    Axy = imfilter(Ixy, window)
    Ayy = imfilter(Iyy, window)

    # Step 2 calculate response
    # determinant
    detA = Axx * Ayy - Axy ** 2
    # trace
    traceA = Axx + Ayy
    # response
    response = detA - k * traceA ** 2

    # step 3. Get points location(x,y)
    # points = corner_peaks(response,min_distance=4)
    points = get_coordinate(response,top_n_points,I.shape)

    return points

def get_coordinate(response, alpha, image_shape):
    # sort with flattern mode
    sortedIndex = np.argsort(response,axis=None)[::-1]

    # maxIndes are index of flattern array
    maxIndexes = sortedIndex[0:alpha]

    # generate points coordinates
    # consider image as coordianate in place m x n
    # each point coordnate is (m', n')
    # m' * n + n' = flatterned index
    m = image_shape[0]
    n = image_shape[1]

    points = np.zeros((2,alpha),dtype=np.int32)
    x = np.floor(maxIndexes/n)
    y = np.mod(maxIndexes,n)
    points[0,:] = x
    points[1,:] = y
    points = points.T

    return points

def display_corner_points(org_img:np.ndarray, points, output_name):
    bool_arr = np.zeros(org_img.shape[:2],dtype=np.int8)
```

```
    bool_arr = np.bool8(bool_arr)

    # pprint(bool_arr)
    for (x,y) in points:
        bool_arr[x,y] = True

    # pprint(bool_arr)
    org_img[bool_arr]=[0,255,0]
    cv2.imwrite(output_name, org_img)
```

In [ ]:

```
# showing result
import matplotlib.pyplot as plt

image_name = "data/carmel/carmel-00.png"
image = cv2.imread(image_name,cv2.IMREAD_GRAYSCALE)
plt.figure(figsize=(20, 15))
plt.imshow(image, cmap='gray', vmin=0, vmax=255)
plt.title("original")
plt.show()


image_folder = "./data/carmel"
for i in range(10):
    I = cv2.imread(f"{image_folder}/carmel-0{i}.png")
    gray = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
    gray = gray / 255.0
    points = harris_corner_points(gray,top_n_points=50000)
    output_name = f"./output/a_harris_output.jpg"
    display_corner_points(I,points,output_name)
    image2 = cv2.imread(output_name)
    image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(20,15))
    plt.imshow(image2)
    plt.title(f"Image with Harris Points: {i}")
    plt.show()
```


original
0
100
200
300
400
500
600
0
200
400
600
800
1000

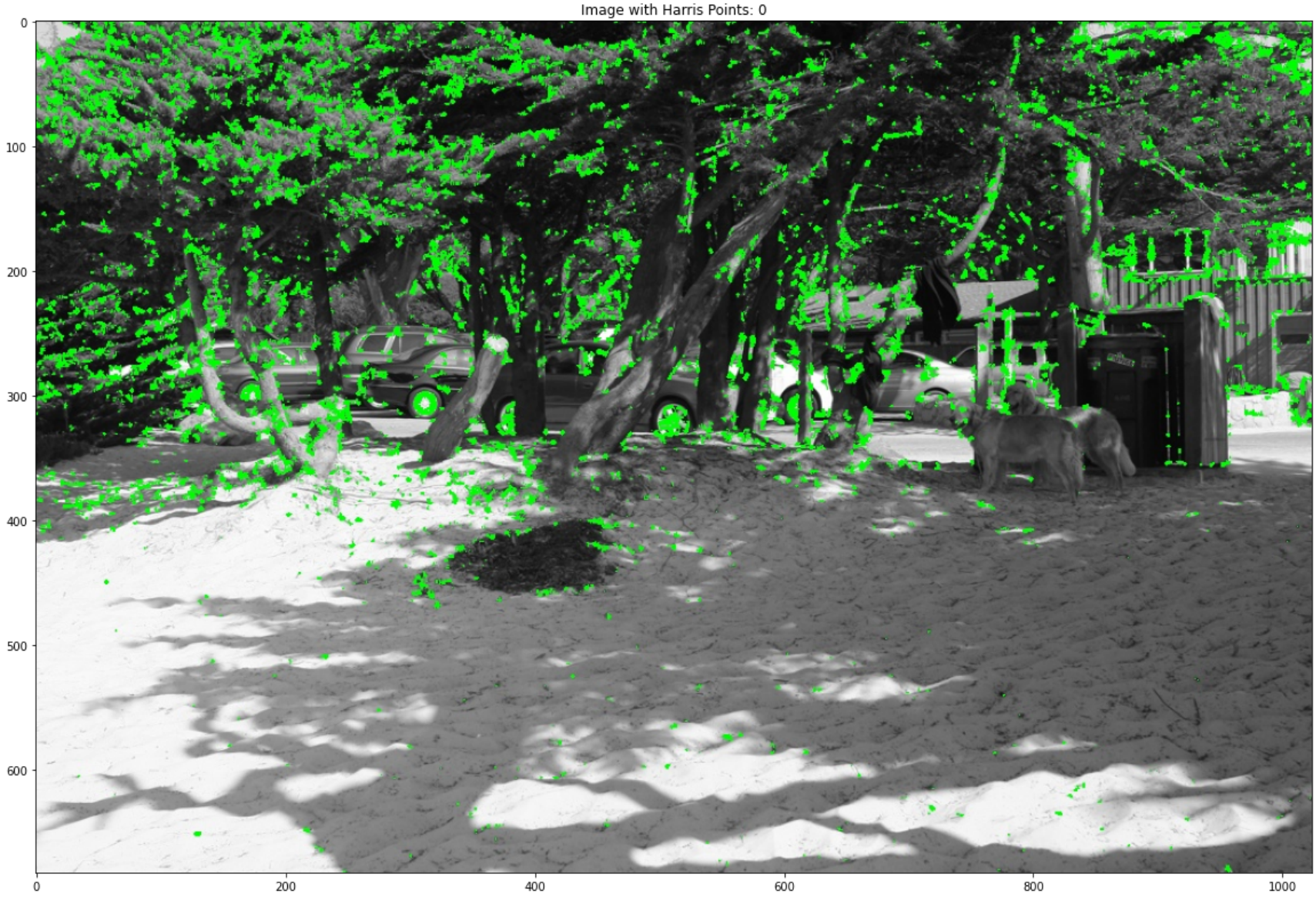


Image with Harris Points: 0
0
100
200
300
400
500
600
0
200
400
600
800
1000


Image with Harris Points: 1
0
100
200
300
400
500
600
0
200
400
600
800
1000

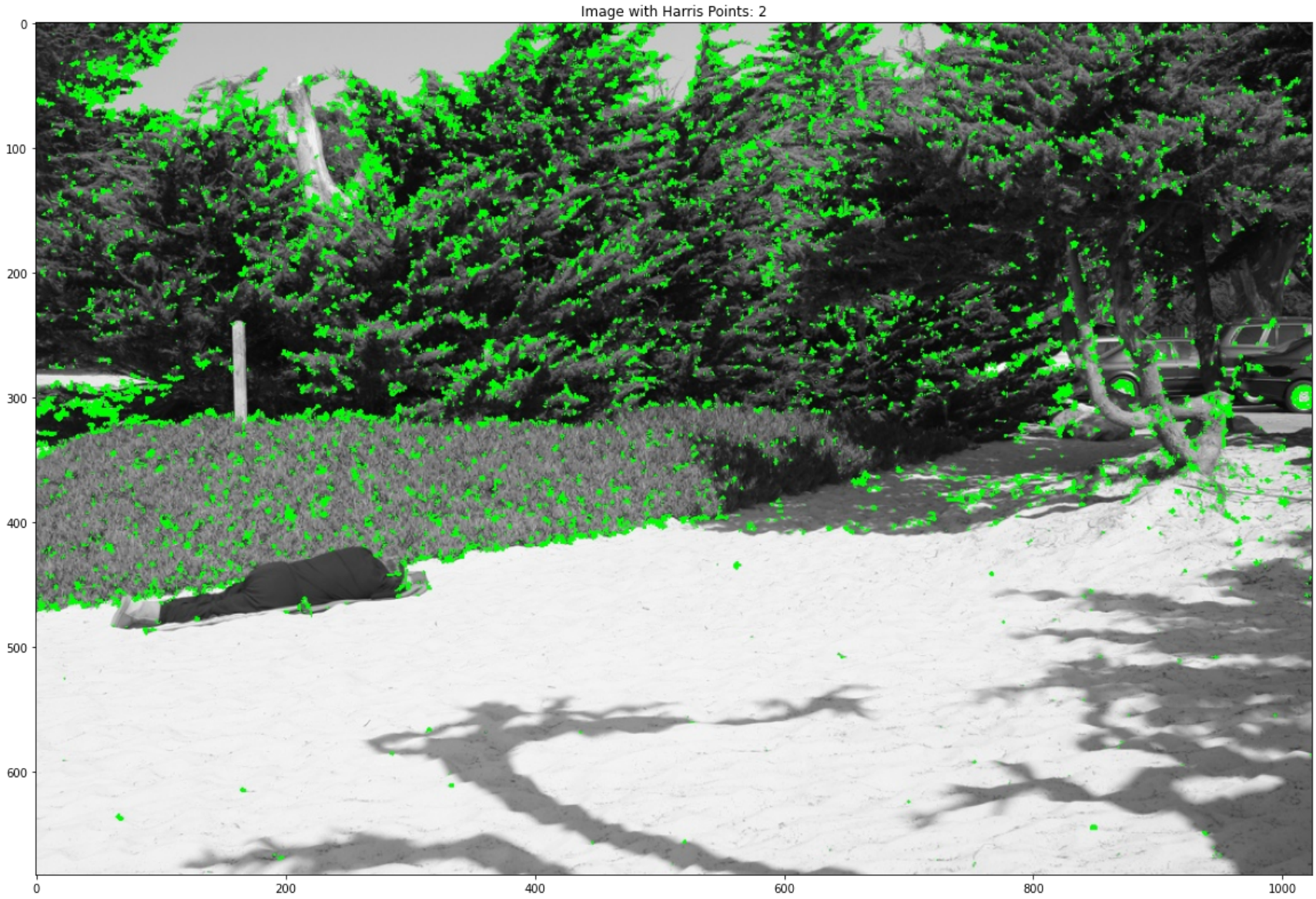


Image with Harris Points: 2
0
100
200
300
400
500
600
0
200
400
600
800
1000

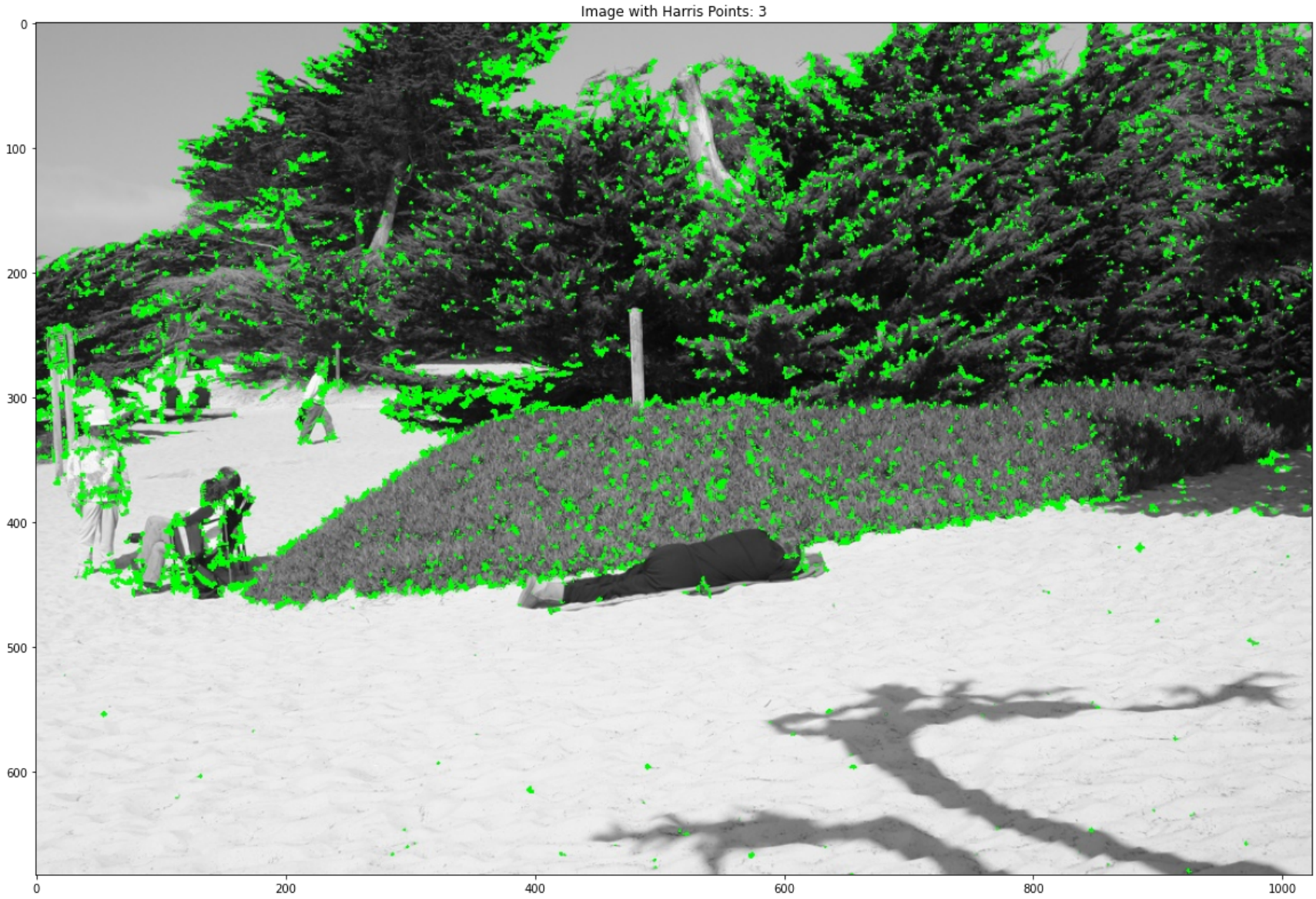


Image with Harris Points: 3
0
100
200
300
400
500
600
0
200
400
600
800
1000

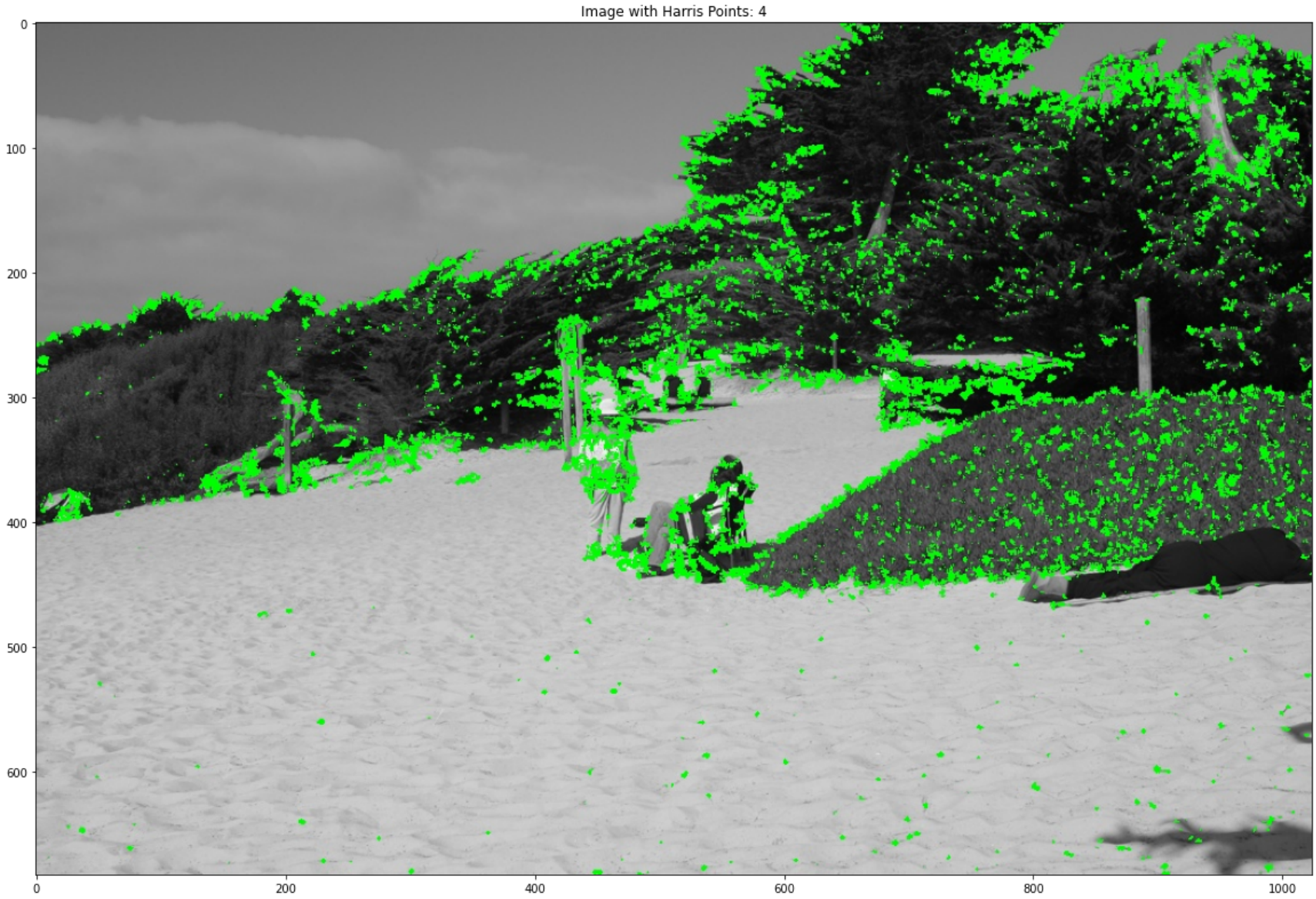


Image with Harris Points: 4
0
100
200
300
400
500
600
0
200
400
600
800
1000

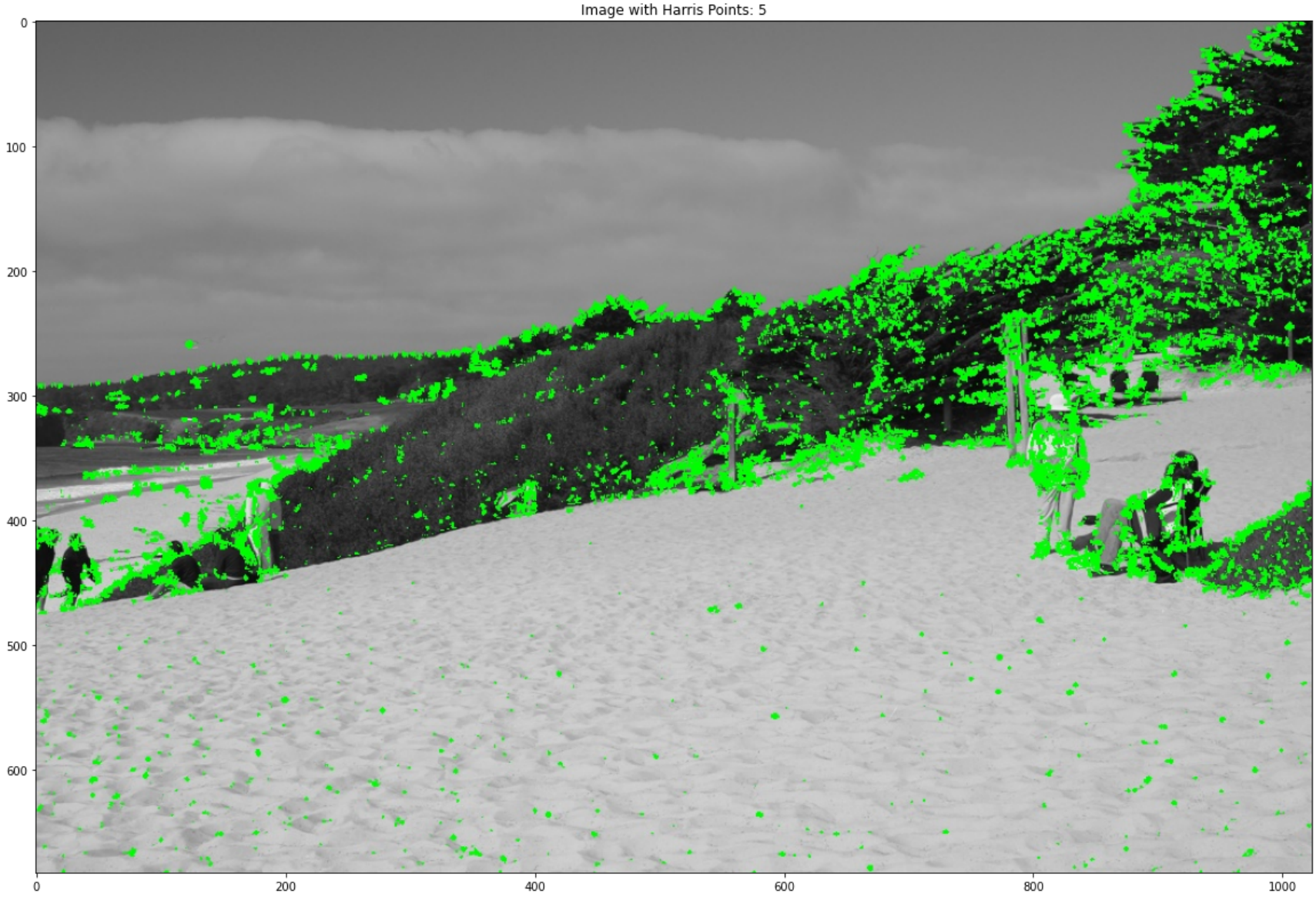


Image with Harris Points: 5
0
100
200
300
400
500
600
0
200
400
600
800
1000

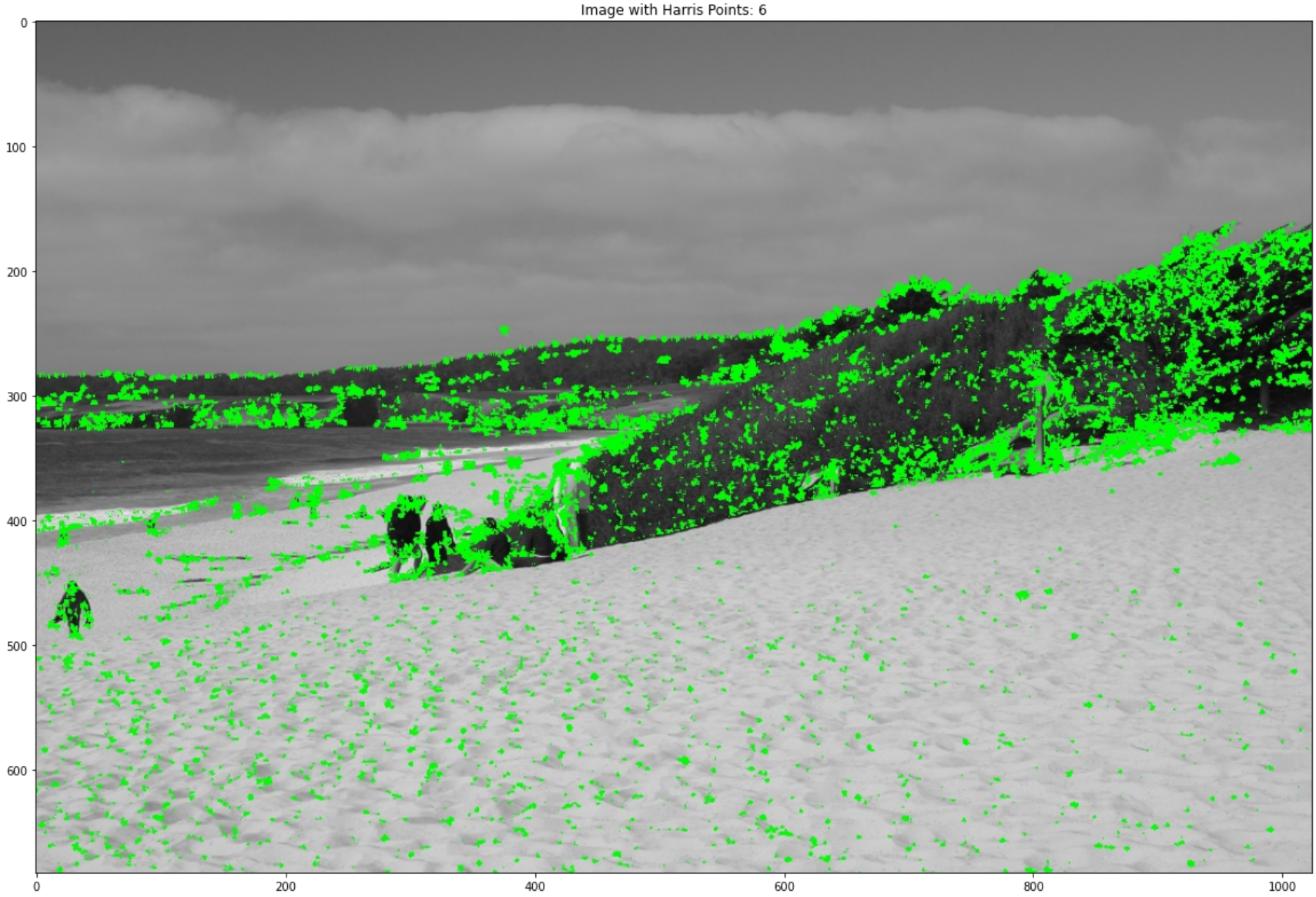


Image with Harris Points: 6
0
100
200
300
400
500
600
0
200
400
600
800
1000


Image with Harris Points: 7
0
100
200
300
400
500
600
0
200
400
600
800
1000


Image with Harris Points: 8
0
100
200
300
400
500
600
0
200
400
600
800
1000

1B. Feature Matching

Question: Implement a simple feature matching by using two feature descriptors of your choice (you can use the available feature descriptors in OpenCV or Matlab). Compare the two feature descriptors and the matching results on a few different images.

Answer:

The matching function is usign K near neighbour algorithm. The function accept descriptor name and display the result.

The first descriptor ORB feature descriptor and the second one is SIFT feature descriptor.

The SIFT gives better result. From the output you can see SIFT detects more good key points.

Accredit: the ploting function has refereced some logic from opencv example code.

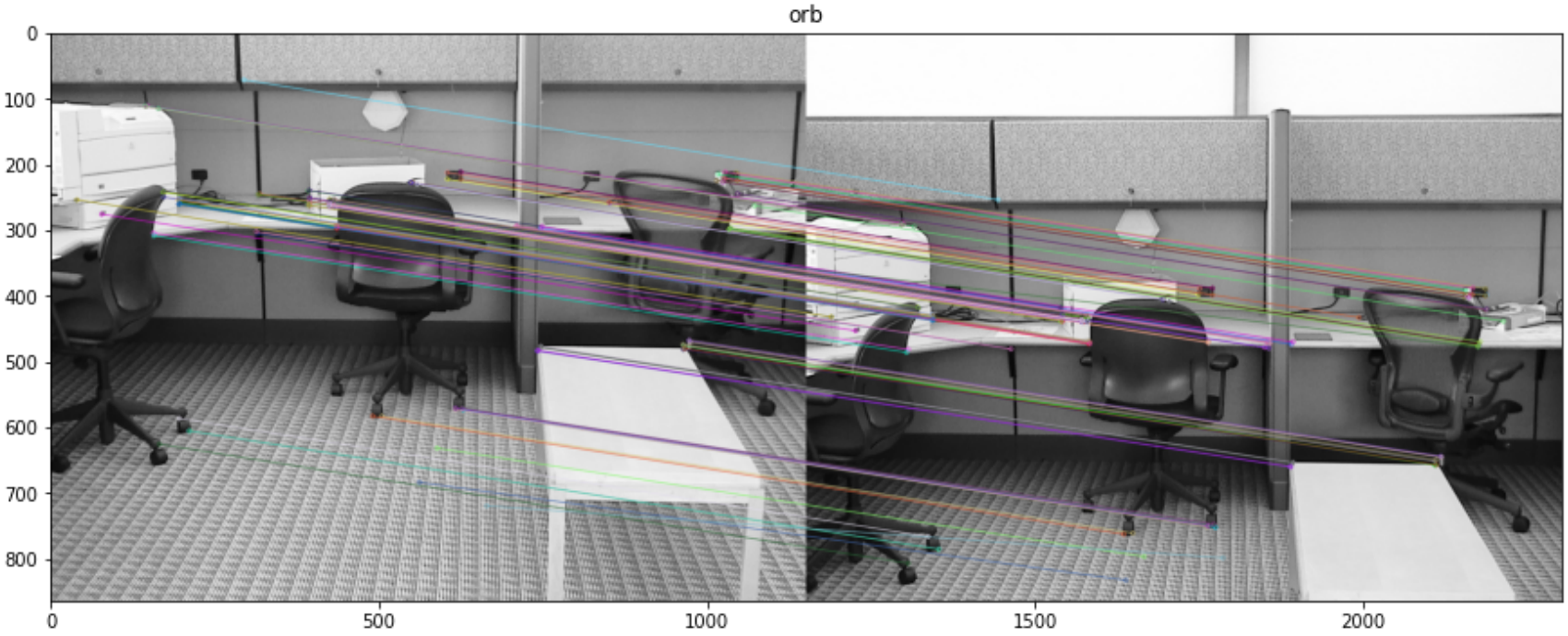
In [ ]:

```
import numpy as np
import cv2
from pprint import pprint
from scipy import ndimage
from skimage.feature import corner_peaks
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt

def knn_match(img1,img2,descriptor='sift',show_limit=100):
    '''Only support sift and orb as descriptor'''

    # Initiate SIFT detector
    if descriptor == 'sift':
        sift = cv2.SIFT_create()
        # find the keypoints and descriptors with SIFT
        # descriptors are 128 dimention histogram,
        # to compare descriptors using Knn
        kp1, des1 = sift.detectAndCompute(img1,None)
        kp2, des2 = sift.detectAndCompute(img2,None)
    elif descriptor == 'orb':
        # Initiate ORB detector
        orb = cv2.ORB_create()
        # find the keypoints and descriptors with ORB
        kp1, des1 = orb.detectAndCompute(img1,None)
        kp2, des2 = orb.detectAndCompute(img2,None)
    else:
        print(f"Unsupported descriptor {descriptor}")
        return

    # KNN match
    k = 2
    m = des1.shape[0]
    n = des2.shape[0]

    matches = list()
```

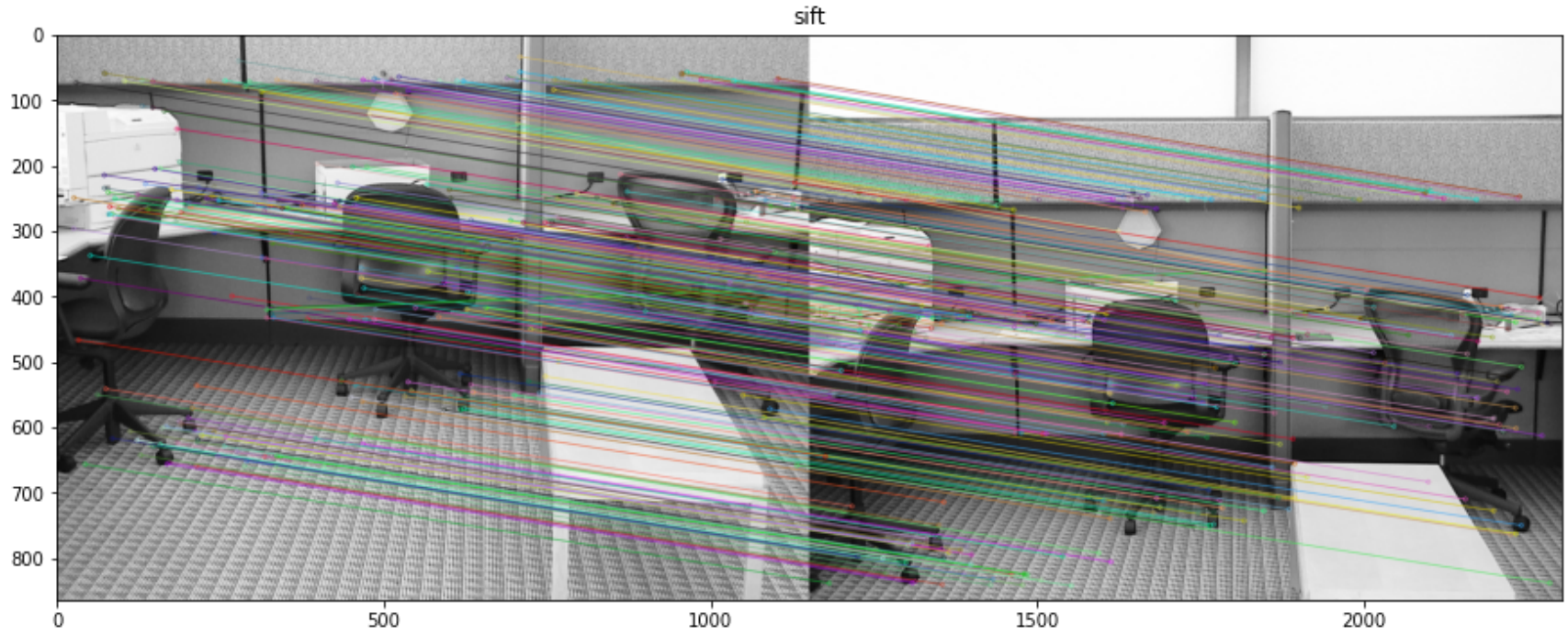
```
    dis_list = pairwise_distances(des1,des2,metric='euclidean')
    for i in range(m):
        d_list = list()
        arr = dis_list[i,:]
        arr = np.argsort(arr,axis=0)[0:k]
        for j in range(k):
            idx_j = arr[j]
            d = cv2.DMatch()
            d.distance = dis_list[i,idx_j]
            d.imgIdx = 0
            d.queryIdx = i
            d.trainIdx = idx_j
            d_list.append(d)

        # add to matches
        matches.append(d_list)

    matches = np.array(matches)
    r1 = matches.shape[0]


    good = []
    dis_values = list()
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])
            dis_values.append(m.distance)

    r2 = len(good)
    print(f"Descriptor {descriptor} detects {r1} points and {r2} points are good.")

    # sort good
    sorted_idx = np.argsort(dis_values)[:show_limit]
    good_show = []
    for idx in sorted_idx:
        good_show.append(good[idx])

    # Display result on plot
    img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good_show,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.figure(figsize=(15, 10))
    plt.imshow(img3)
    plt.title(descriptor)
    plt.show()
```

In [ ]:
```
# img_name1 = './data/office/office-00.png'
# img_name2 = './data/office/office-01.png'

images = [
    './data/office/office-00.png',
    './data/office/office-01.png',
    './data/shanghai/shanghai-00.png',
    './data/shanghai/shanghai-01.png',
    './data/carmel/carmel-00.png',
    './data/carmel/carmel-01.png',
]

n = len(images)//2
for i in range(n):
    img1 = cv2.imread(images[2*i],cv2.IMREAD_GRAYSCALE)          # queryImage
    img2 = cv2.imread(images[2*i + 1],cv2.IMREAD_GRAYSCALE) # trainImage
    knn_match(img1,img2,'orb',300)
    knn_match(img1,img2,'sift',300)
```

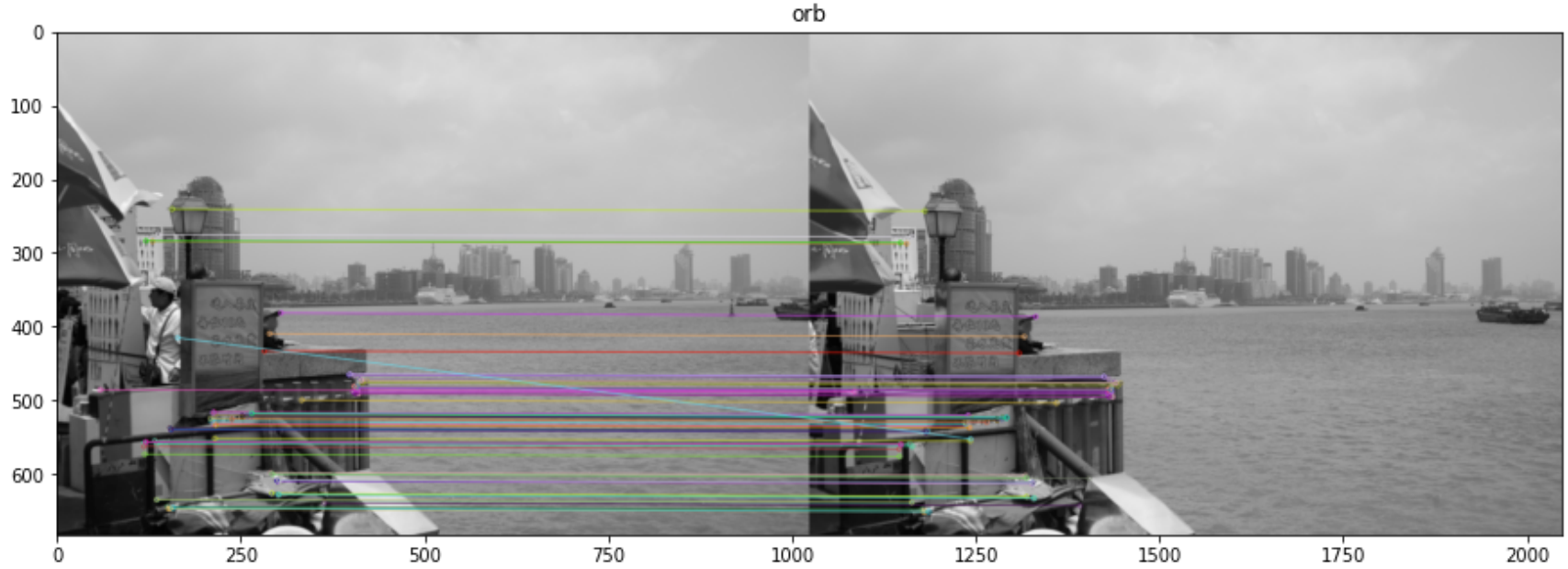
```
Descriptor orb detects 500 points and 132 points are good.
```



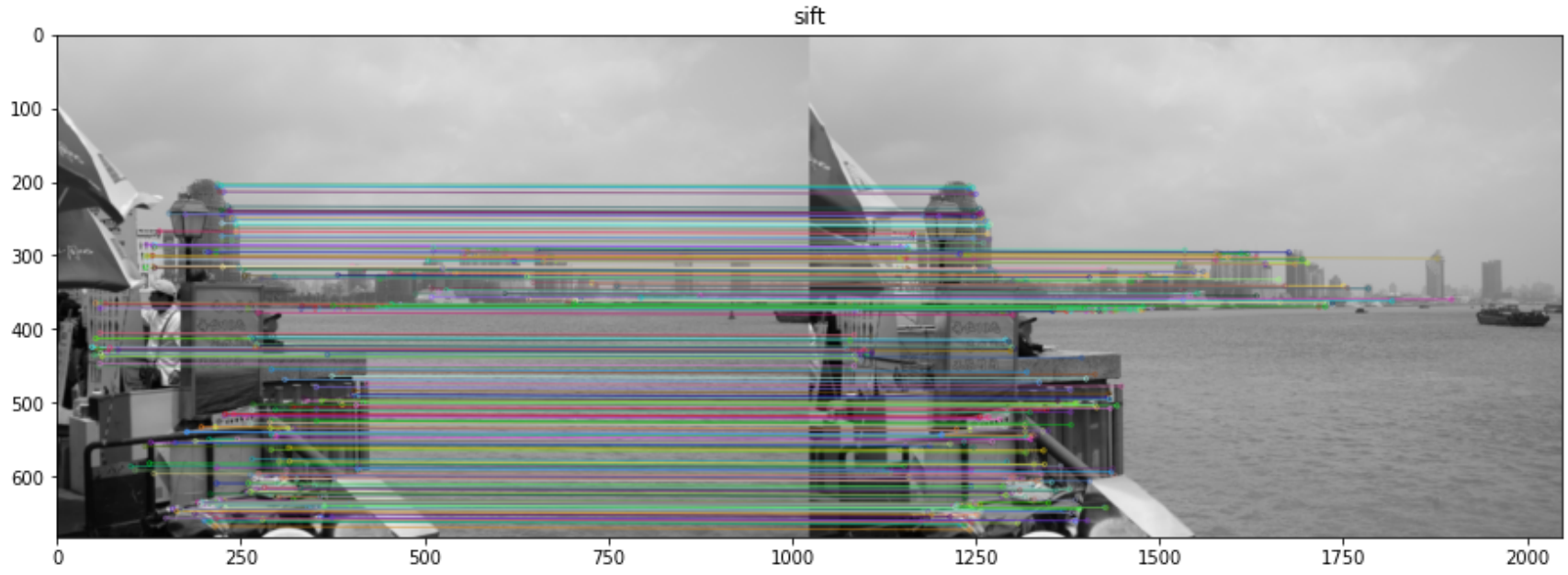

```
Descriptor sift detects 8726 points and 883 points are good.
```



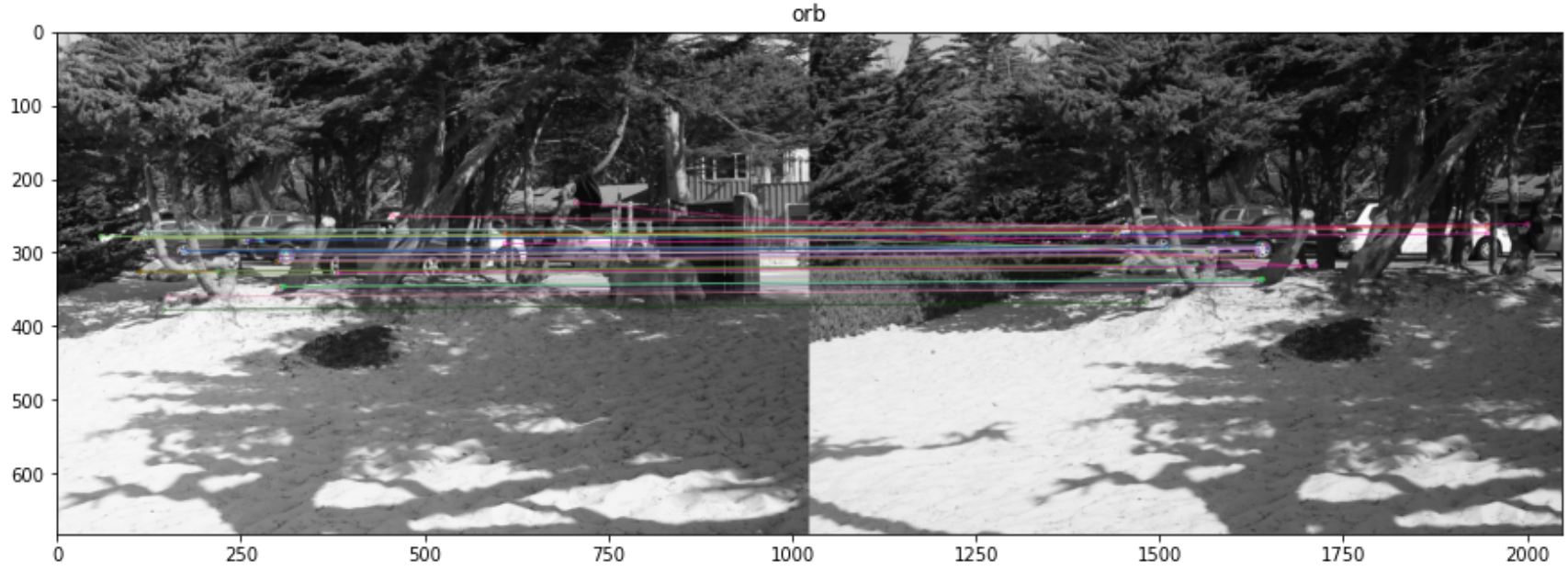

```
Descriptor orb detects 500 points and 76 points are good.
```



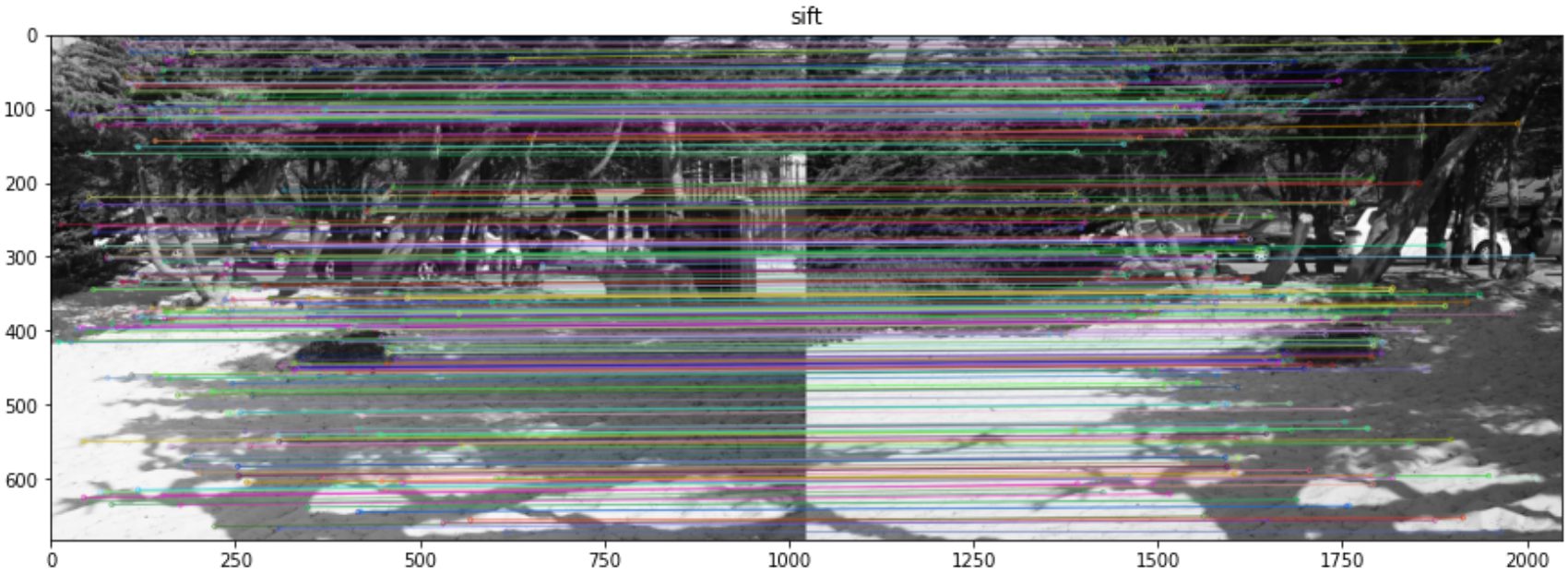

```
Descriptor sift detects 1659 points and 851 points are good.
```



Descriptor orb detects 500 points and 40 points are good.



Descriptor sift detects 6963 points and 2184 points are good.



1C Feature tracking

Question: (10 points) Instead of finding feature points independently in multiple images and then matching them, find features in the first image of a video or image sequence and then re-locate the corresponding points in the next frames using search and gradient descent (Shi and Tomasi 1994). When the number of tracked points drops below a threshold or new regions in the image become visible, find additional points to track.

Answer:
The code is included below, here are some comments to gudie you through.

function `get_pre_points` will generate good features to track

function `get_new_points` will search new frame based on pre_points

function `feature_match_by_tracking` is main function, the input is image folder, which includes video frames download from internet. This can be changed to mp4, but for testing purpose some images frames are good enough. In this function, there is a logic

```
if m < min_threshold:
            # todo recalculate points
            # recalulate pre_points
```

```
        print(f"good points is lower than {min_threshold}, recalculating...")
        (pre_points, kp1) = get_pre_points(pre_gray,n)
        (new_points,kp2,matches) = get_new_points(pre_gray,new_gray,pre_points)
```

This logic will check good points left, if it is lower than threshold, will re-retrieve pre_points from good features to track.

The output shows current frame idex, good features left information and matching lines between the previous frame and current frame. The frame 36 shows the result it triggered the re-retrieve pre_points action.

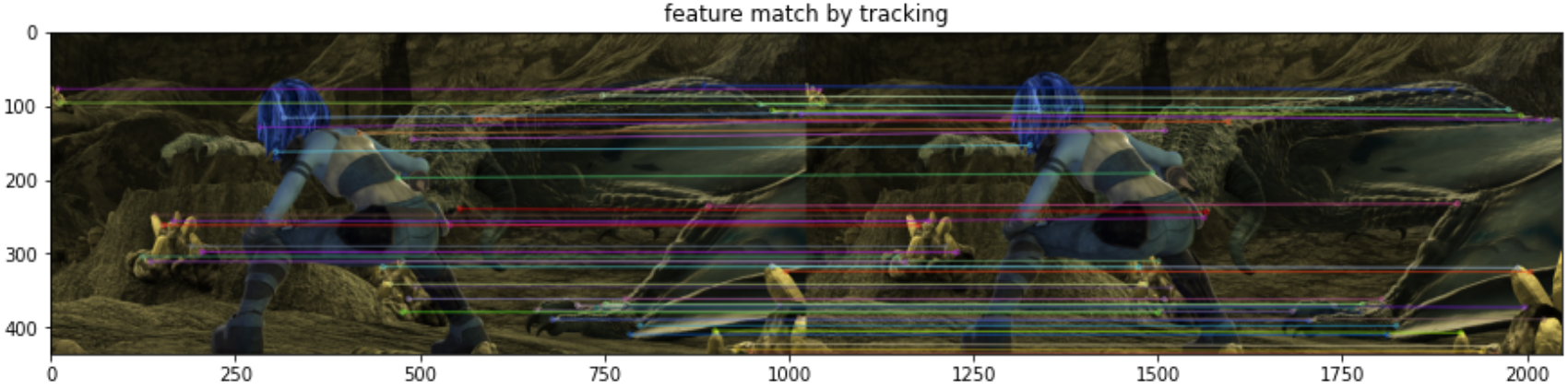
In [ ]:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
from imutils import paths


def get_pre_points(pre_gray,n):
    pre_points = cv2.goodFeaturesToTrack(pre_gray,n,0.01,10)
    pre_points = np.array(pre_points,dtype='float32')
    kp1 = list()
    for j in range(n):
        d = cv2.KeyPoint()
        d.pt = pre_points[j][0]
        d.size = 31.0
        kp1.append(d)
    return(pre_points,kp1)

def get_new_points(pre_gray,new_gray,pre_points):
    new_points, status, err = cv2.calcOpticalFlowPyrLK(pre_gray,
    new_gray,pre_points,None,maxLevel=2)

    n = pre_points.shape[0]
    # show matching
    # convert format to keypoints
    kp2 = list()
    matches = list()
    for j in range(n):
        good = status[j,0]

        d = cv2.KeyPoint()
        d.pt = new_points[j,0]
        d.size = 31
        kp2.append(d)

        if good != 0:
```

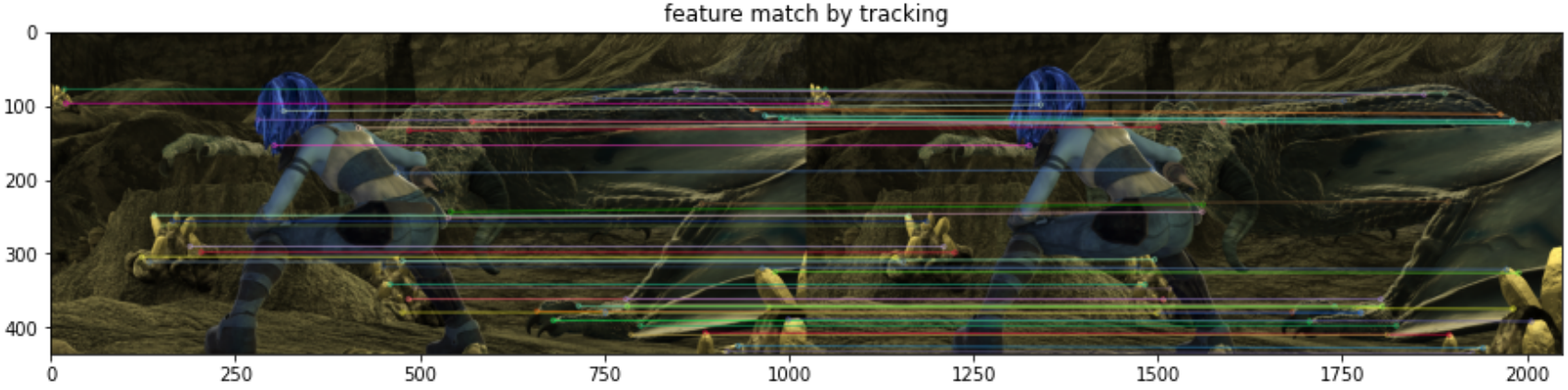
```
            m = cv2.DMatch()
            m.queryIdx = j
            m.trainIdx = j
            matches.append(m)

    return (new_points,kp2,matches)

def feature_match_by_tracking(image_folder,points_show=50,min_threshold=40):
    print("[INFO] loading images...")
    imagePaths = sorted(list(paths.list_images(image_folder)))
    images = []

        # images to stitch list
    for imagePath in imagePaths:
        image = cv2.imread(imagePath)
        images.append(image)

    # it can be chagned to read mp4 if needed
    n = 100
    pre_img  = images[0]
    pre_gray = cv2.cvtColor(pre_img,cv2.COLOR_BGR2GRAY)
    (pre_points, kp1) = get_pre_points(pre_gray,n)

    for i in range(1,len(images)):
        new_img  = images[i]
        new_gray = cv2.cvtColor(new_img,cv2.COLOR_BGR2GRAY)

        (new_points,kp2,matches) = get_new_points(pre_gray,new_gray,pre_points)
        m = len(matches)

        print(f"frame {i}: good points {m}")

        if m < min_threshold:
            # todo recalculate points
            # recalulate pre_points
            print(f"good points is lower than {min_threshold}, recalculating...")
            (pre_points, kp1) = get_pre_points(pre_gray,n)
            (new_points,kp2,matches) = get_new_points(pre_gray,new_gray,pre_points)


        img3 = cv2.drawMatches(pre_img,kp1,new_img,kp2,\
            matches[:points_show], None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

        plt.figure(figsize=(15, 10))
        plt.imshow(img3)
```

```
        plt.title('feature match by tracking')
        plt.show()

        pre_img = new_img.copy()
        pre_points = new_points.copy()
        pre_gray = new_gray.copy()
        kp1 = kp2.copy()

image_folder = './data/cave_3'
feature_match_by_tracking(image_folder,50,40)
```

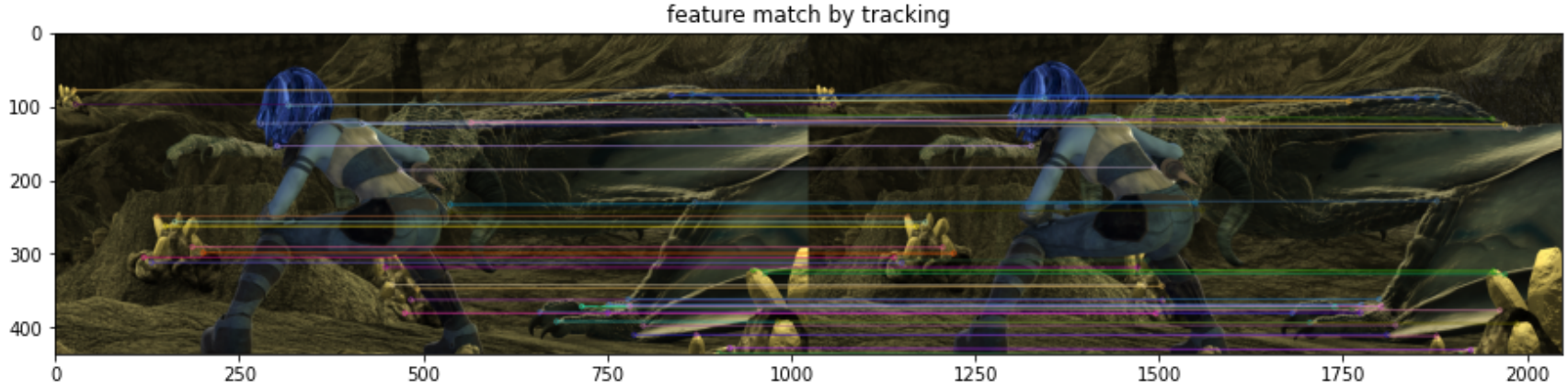
```
[INFO] loading images...
frame 1: good points 100
```



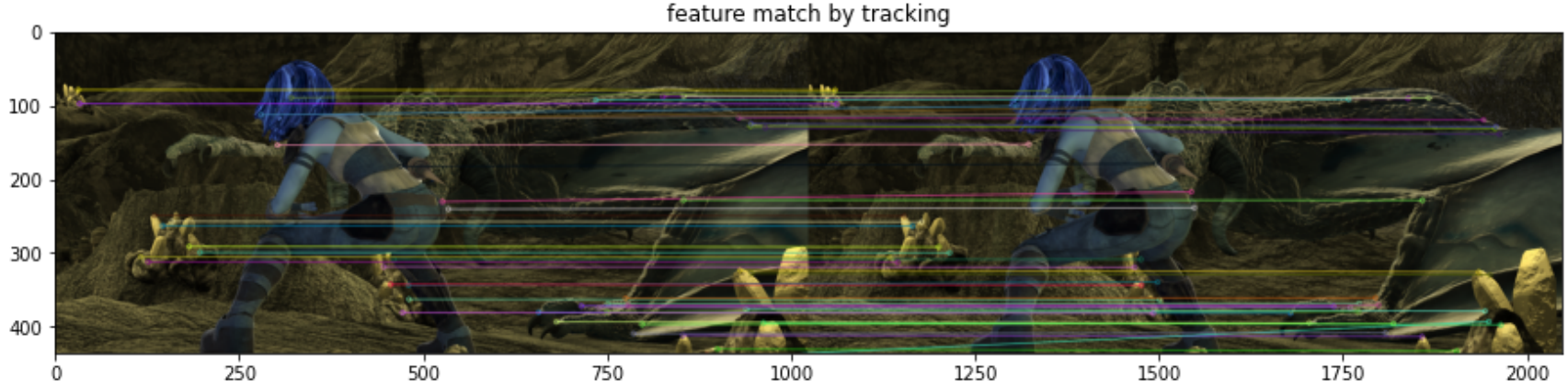

```
frame 2: good points 100
```



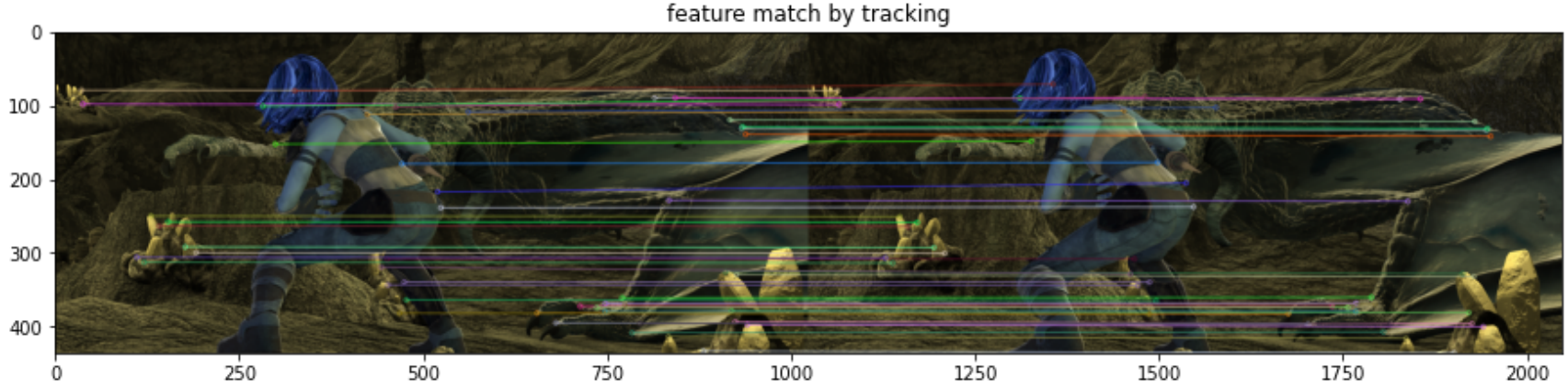

```
frame 3: good points 100
```



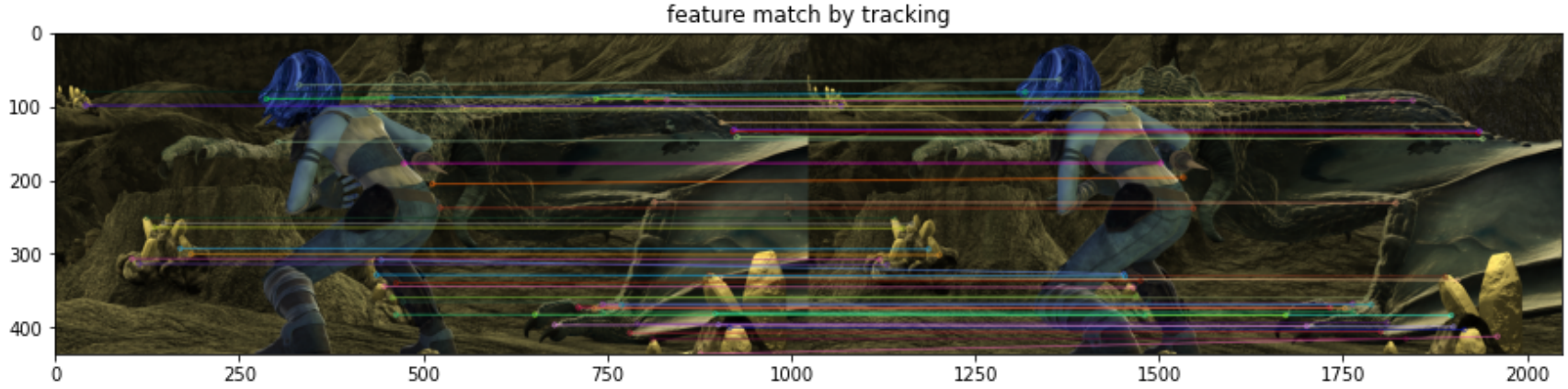

```
frame 4: good points 100
```



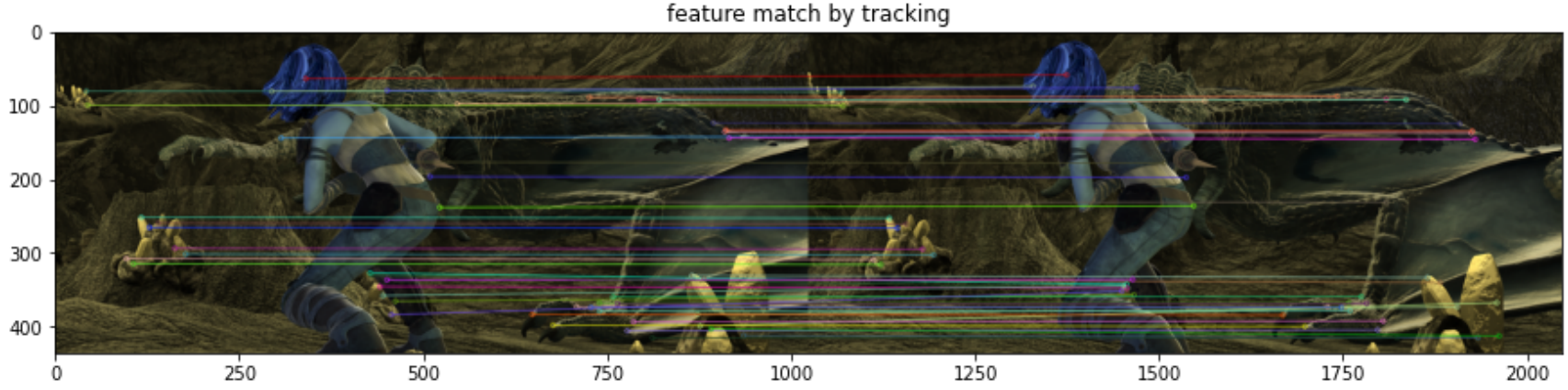

```
frame 5: good points 100
```



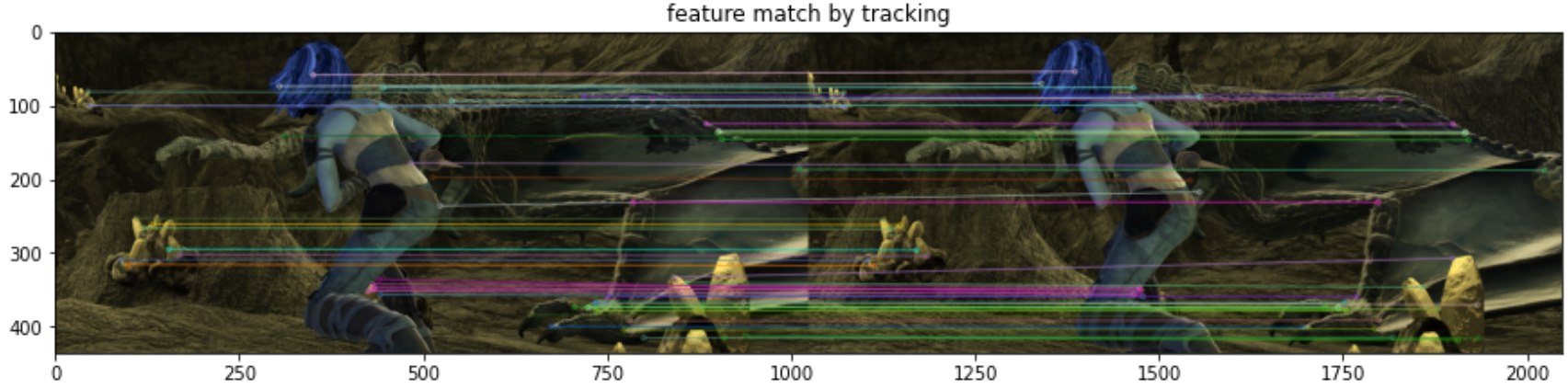

```
frame 6: good points 99
```



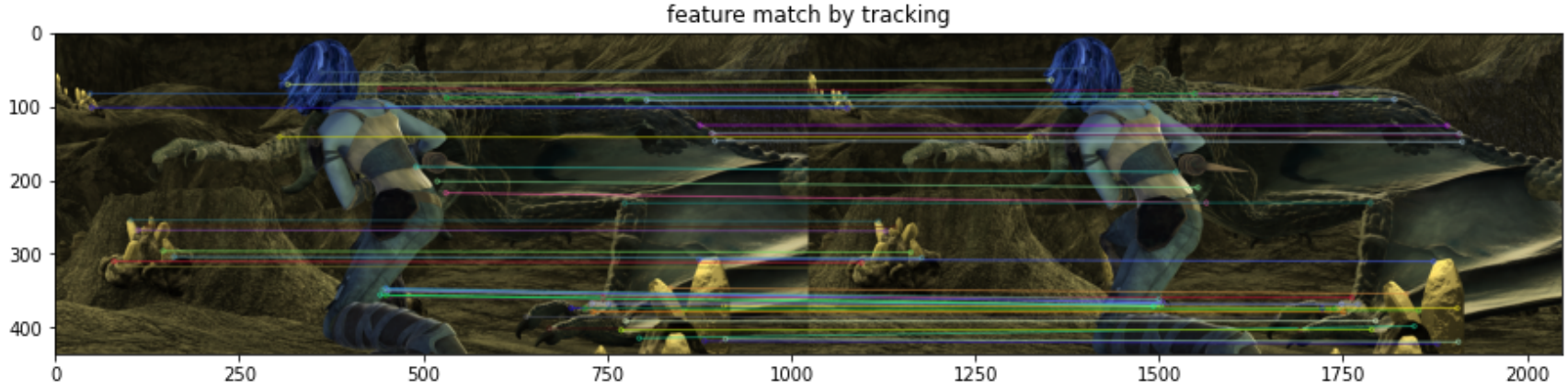

```
frame 7: good points 98
```



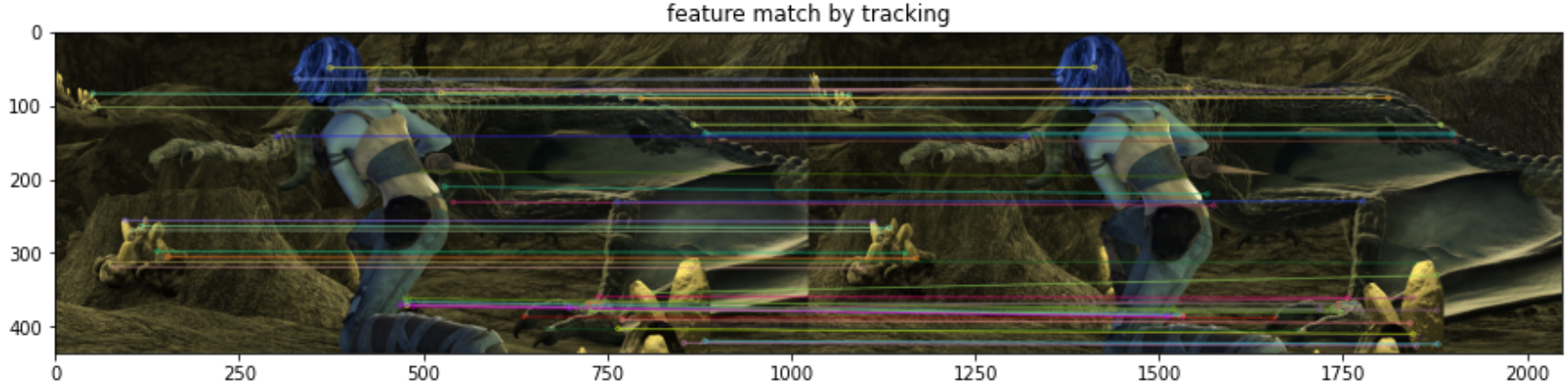

```
frame 8: good points 98
```



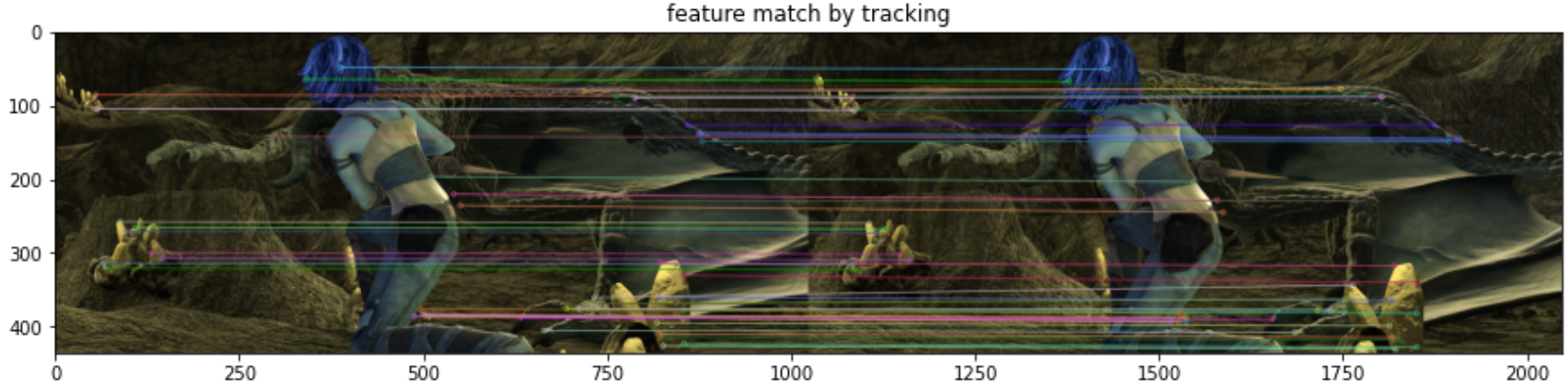

```
frame 9: good points 97
```



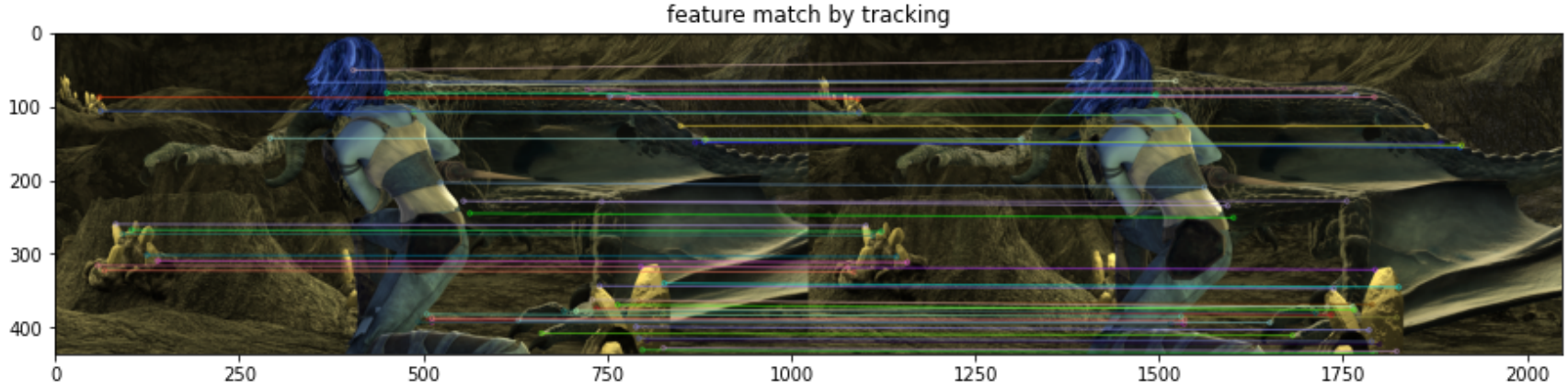

```
frame 10: good points 97
```



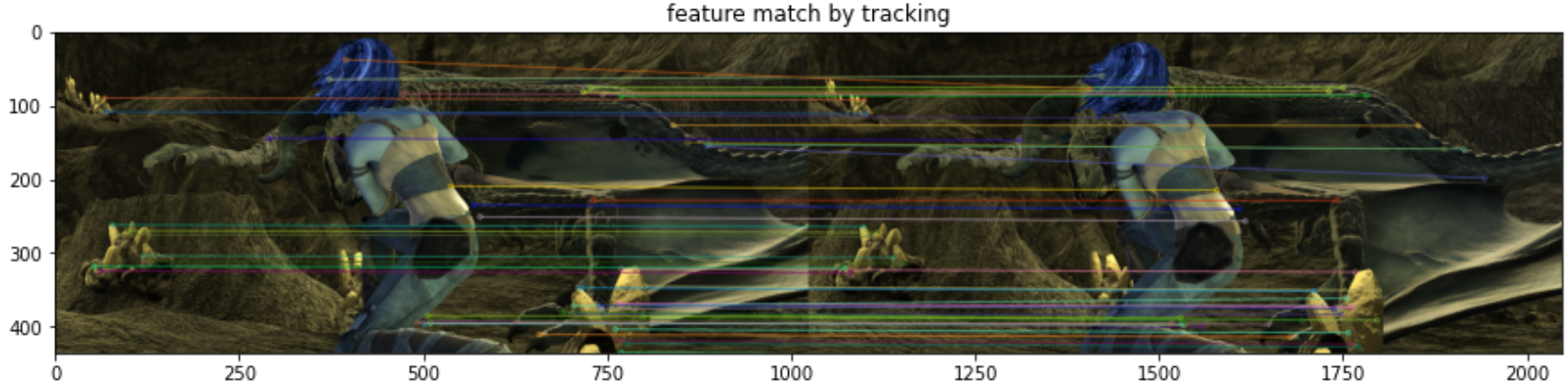

```
frame 11: good points 97
```



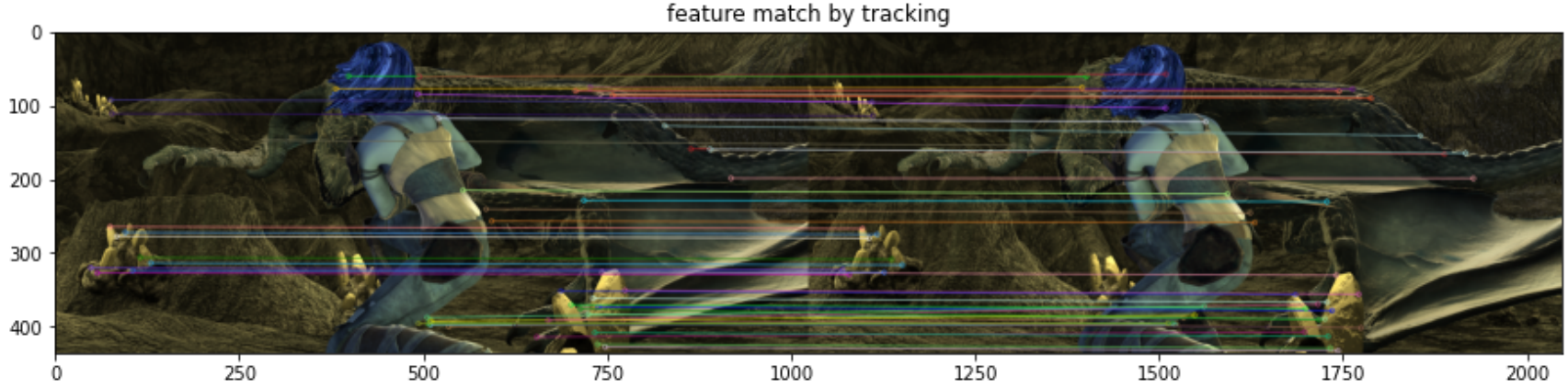

```
frame 12: good points 97
```



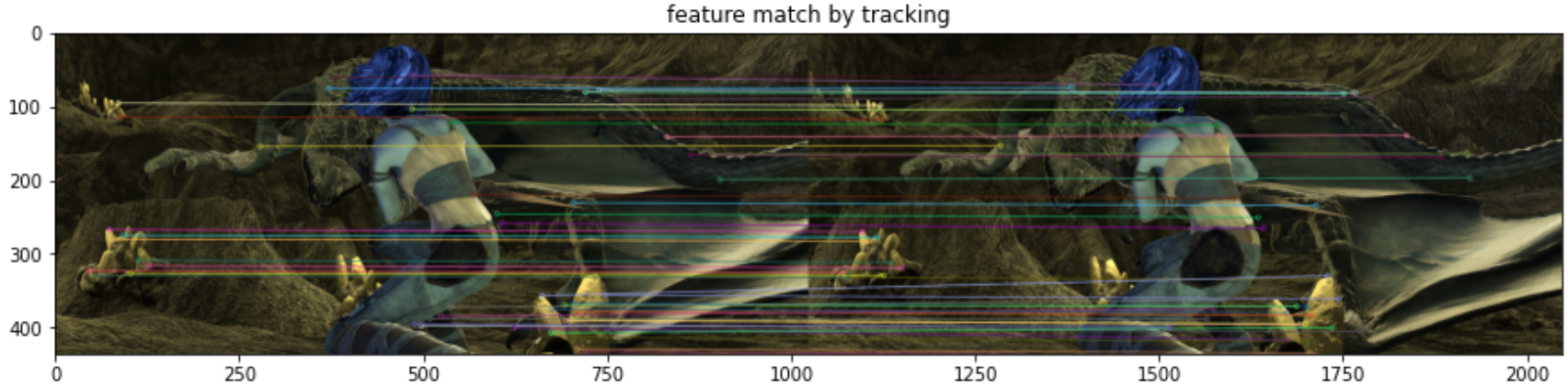

```
frame 13: good points 96
```



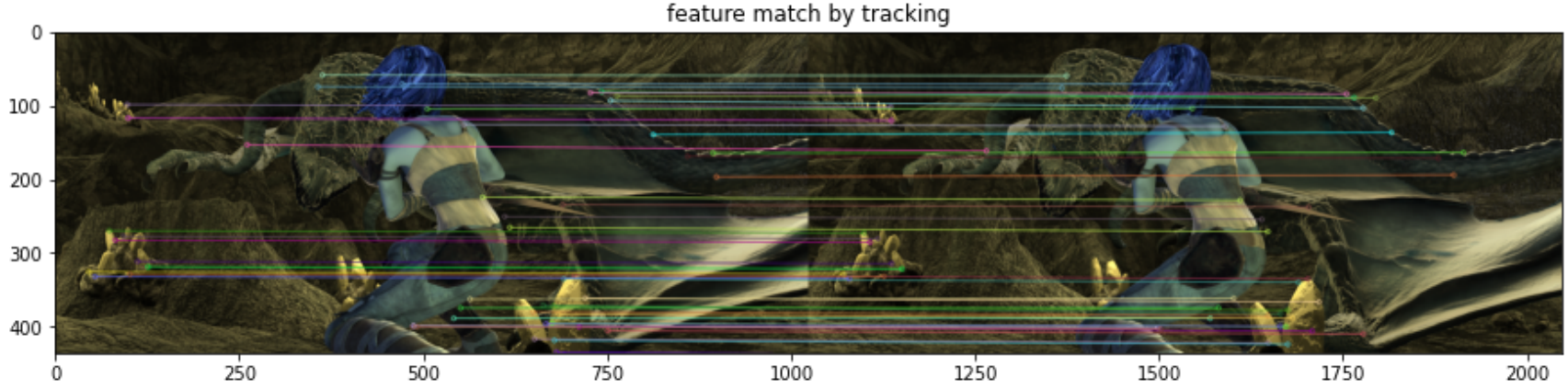

```
frame 14: good points 94
```



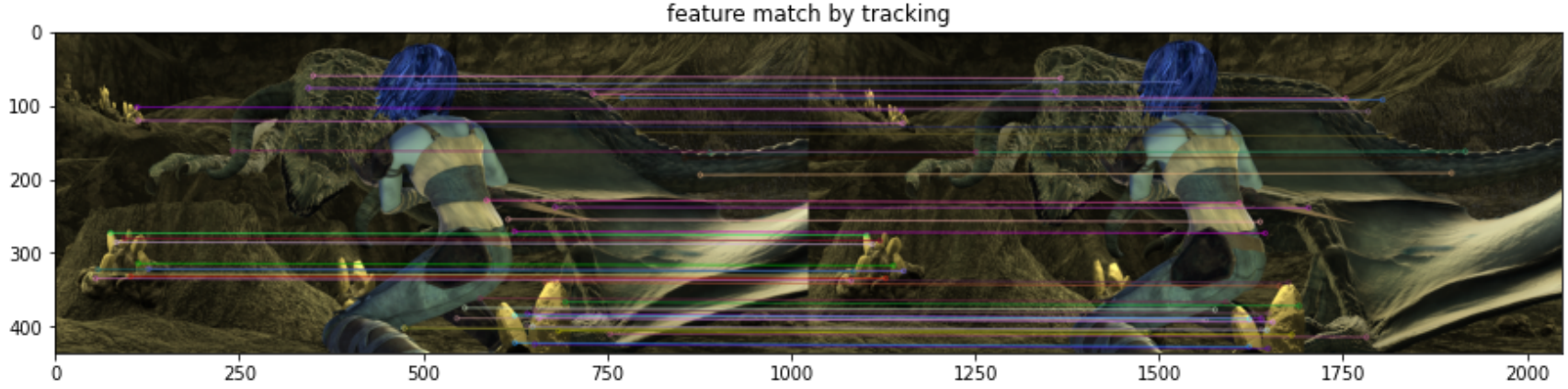

```
frame 15: good points 94
```



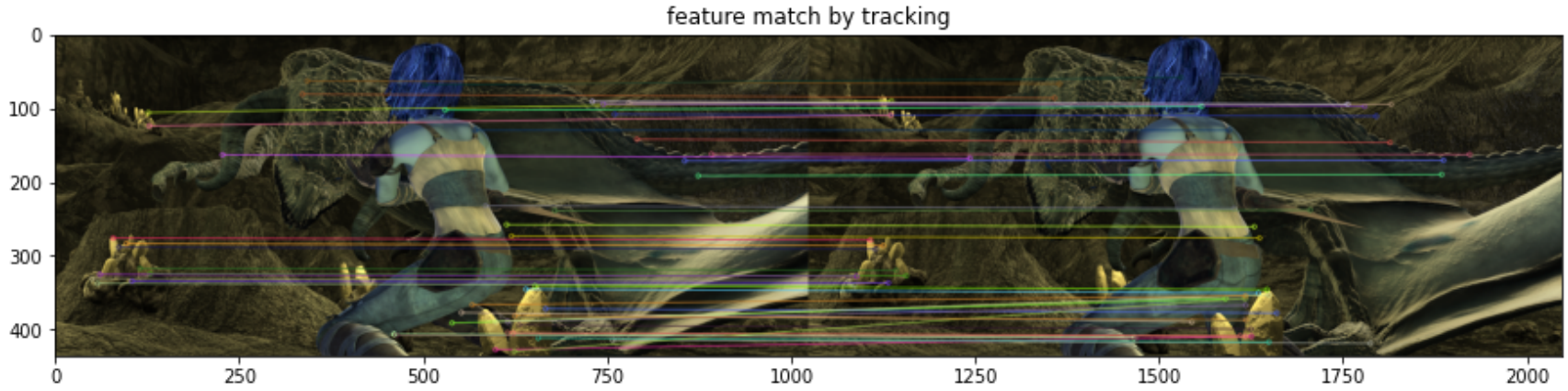

```
frame 16: good points 93
```



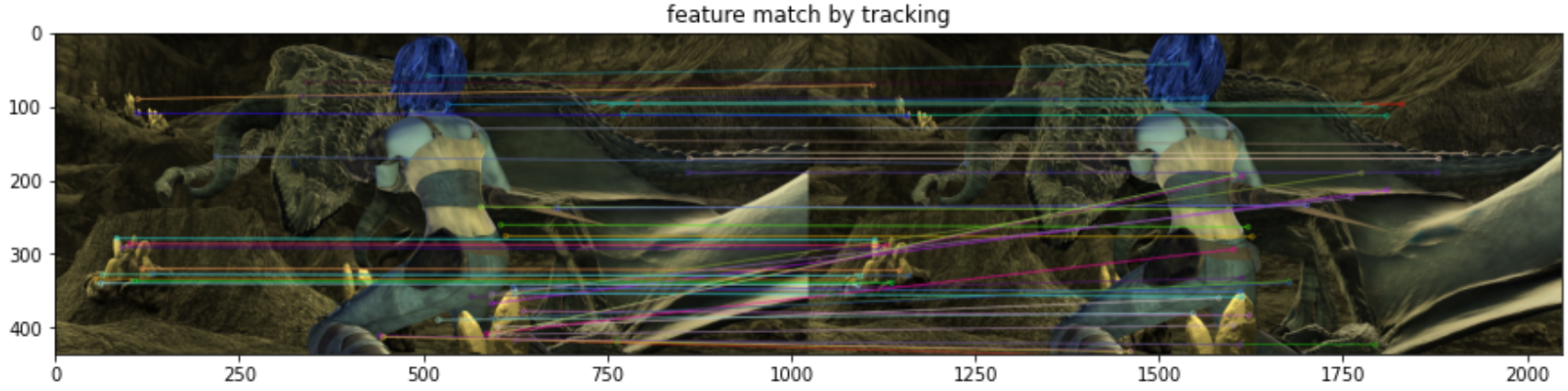

```
frame 17: good points 91
```



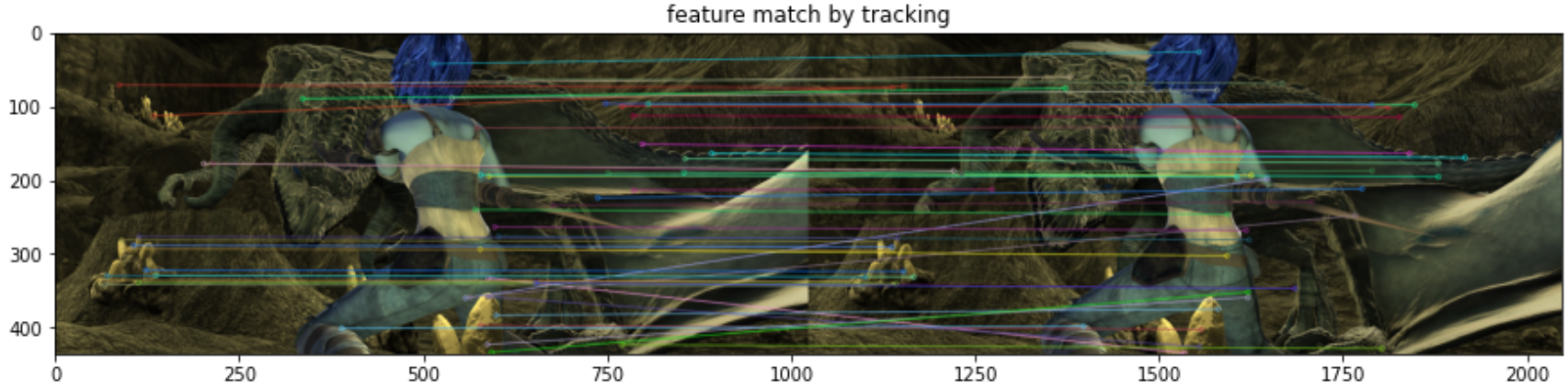

```
frame 18: good points 90
```



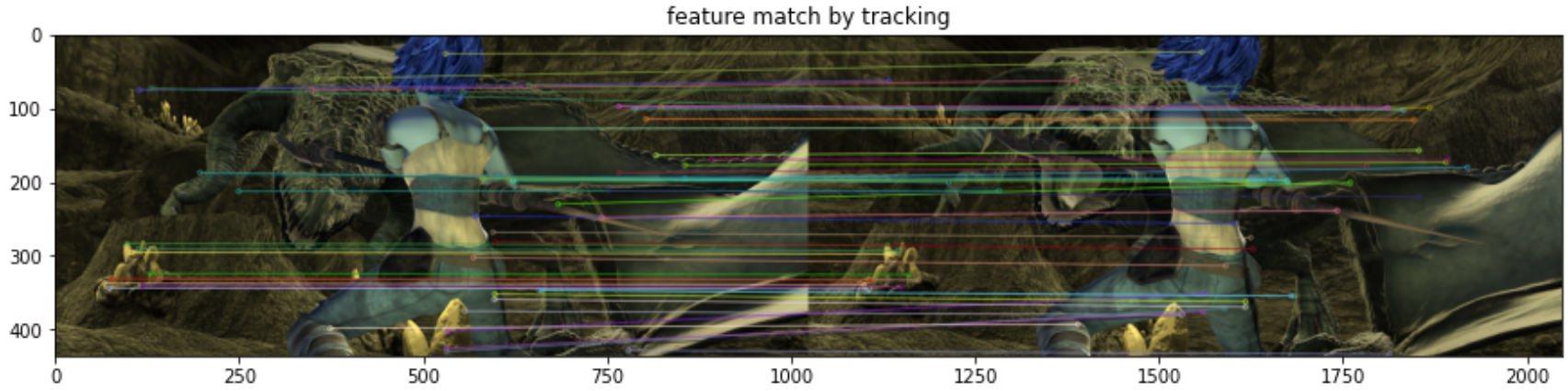

```
frame 19: good points 89
```



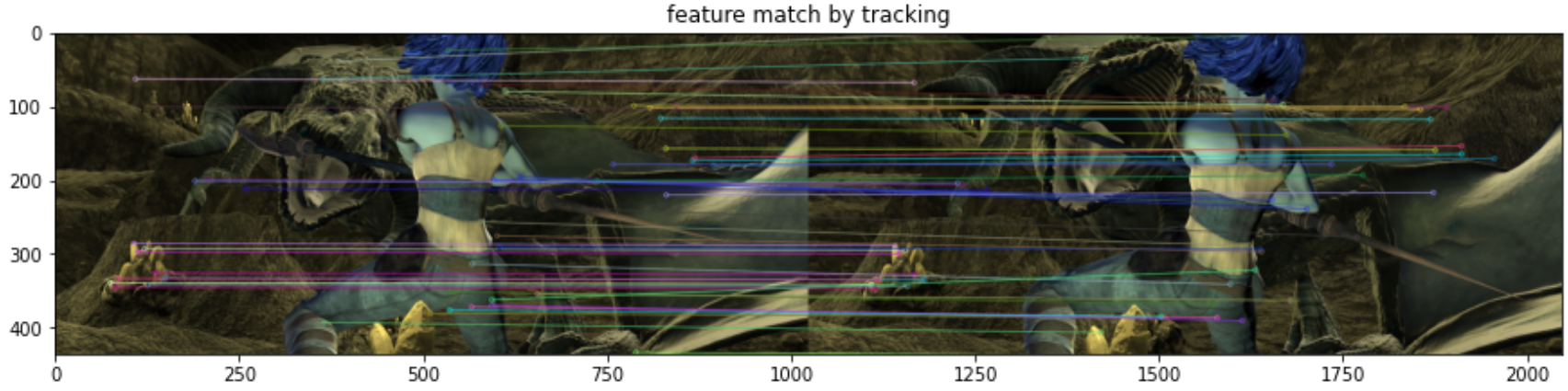

```
frame 20: good points 86
```



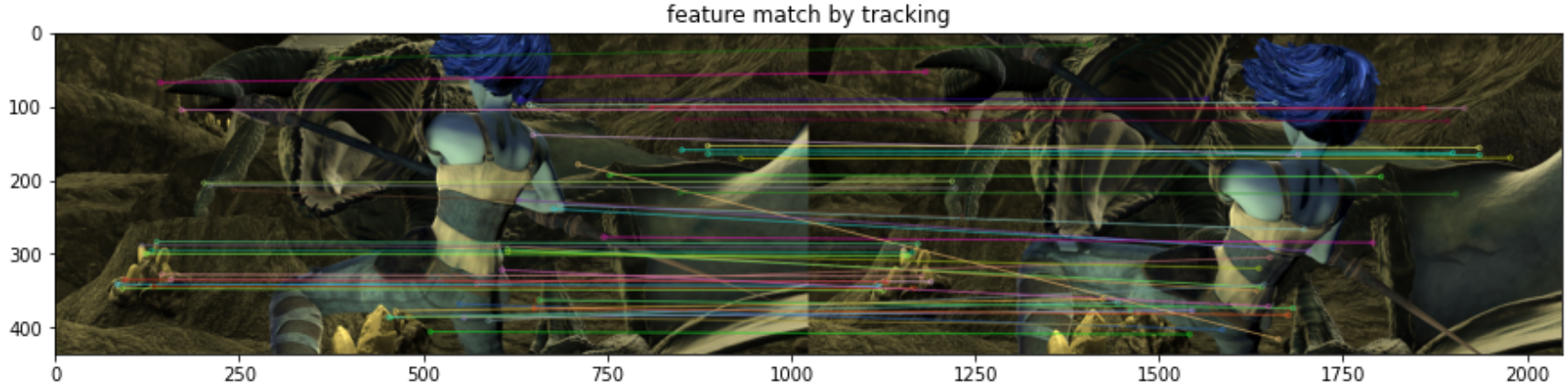

```
frame 21: good points 86
```



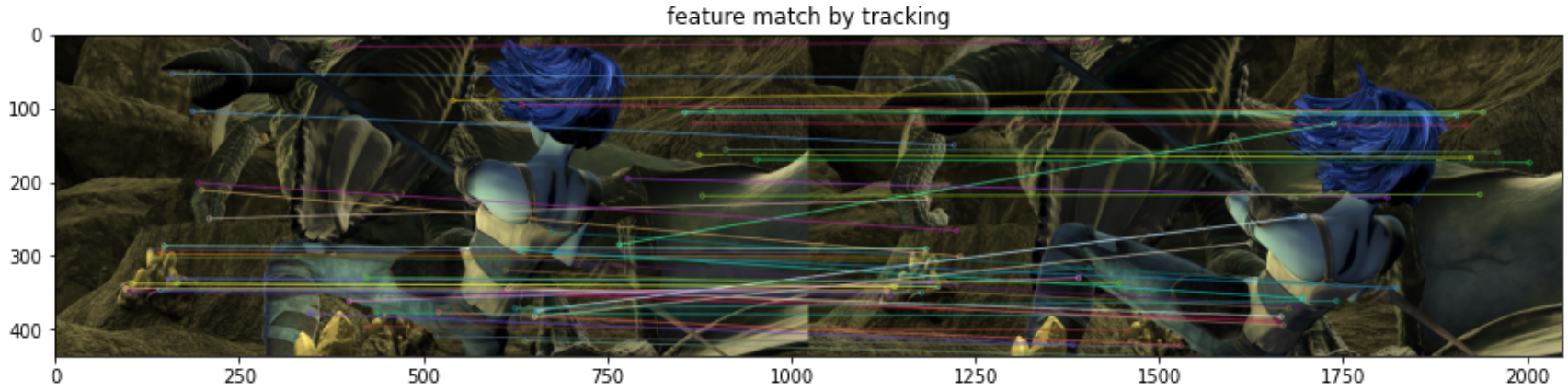

```
frame 22: good points 86
```



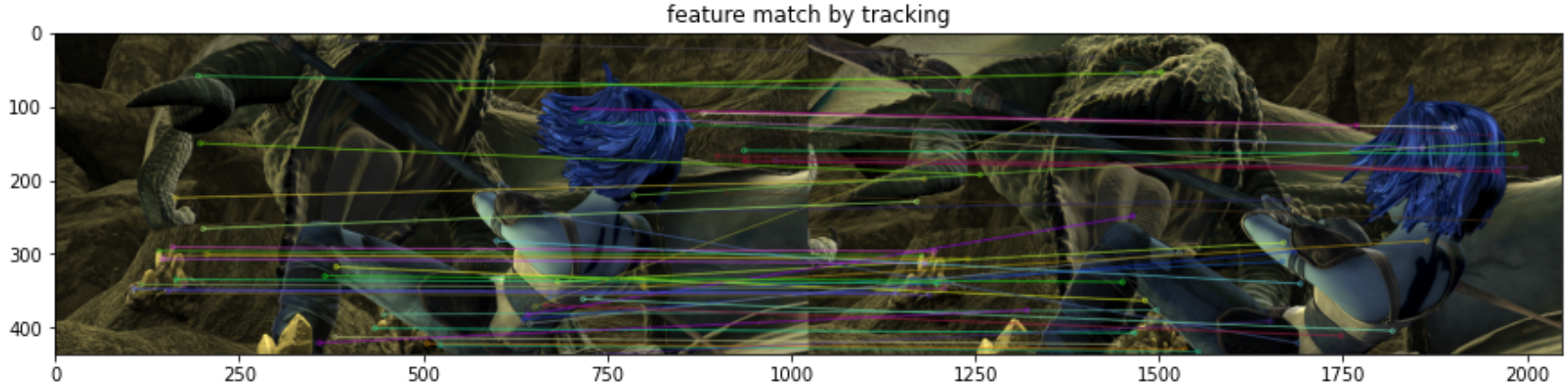

```
frame 23: good points 82
```



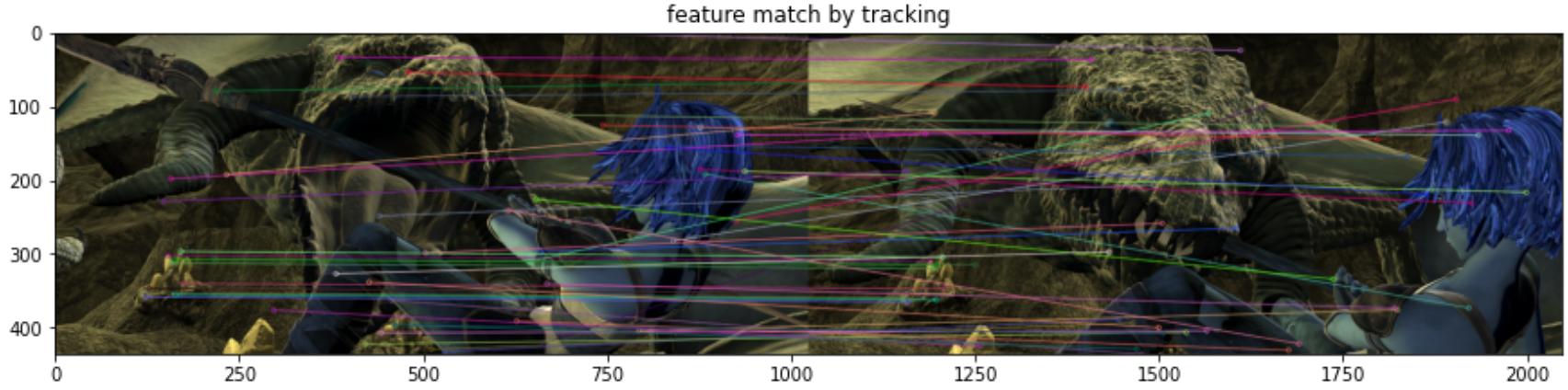

```
frame 24: good points 79
```



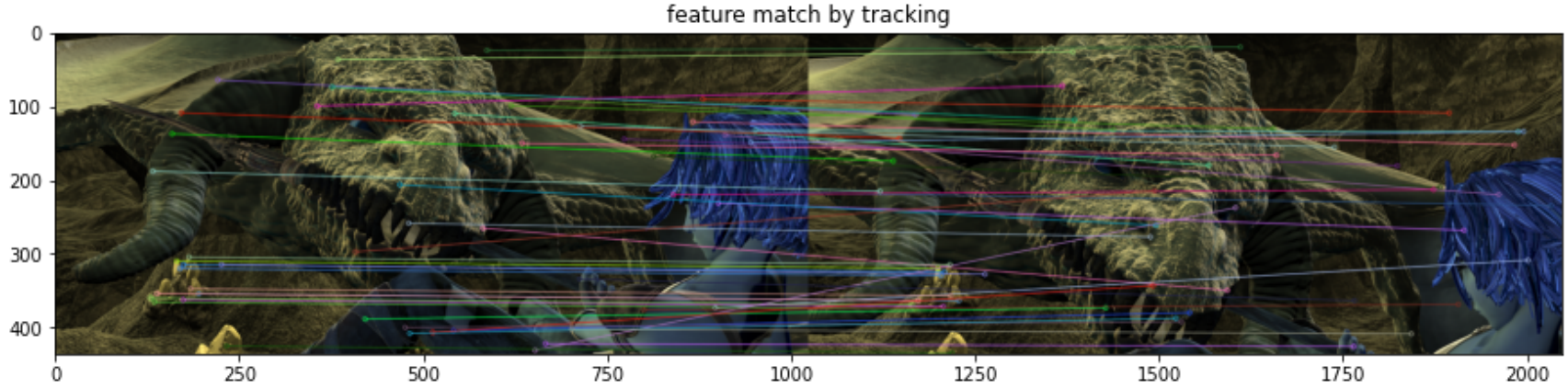

```
frame 25: good points 75
```



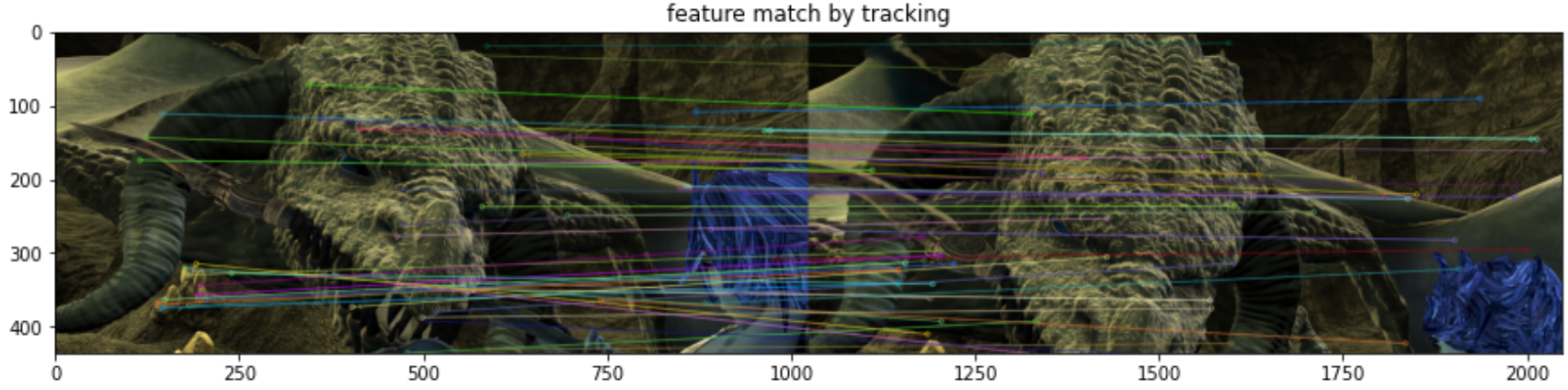

```
frame 26: good points 66
```



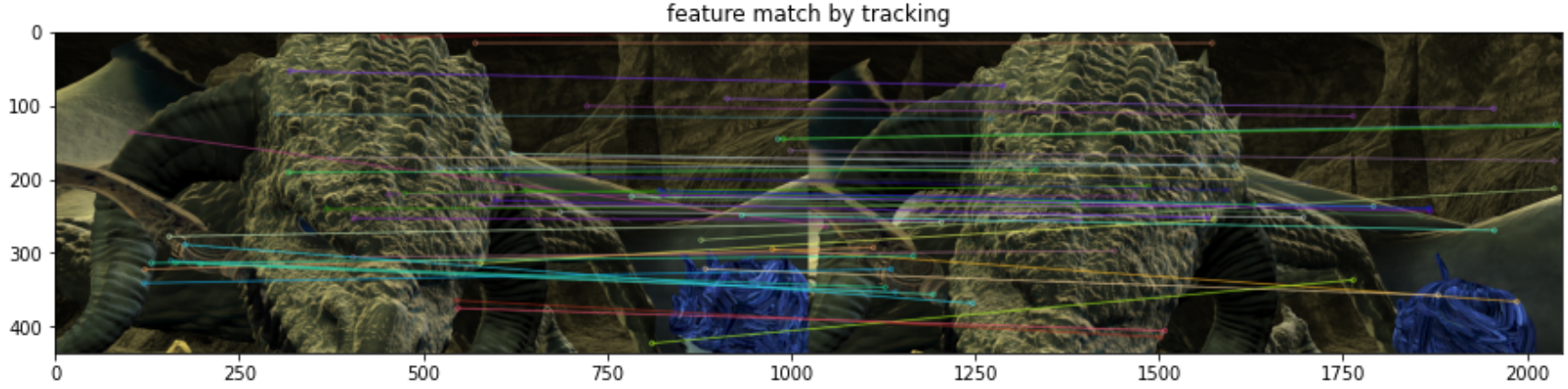

```
frame 27: good points 62
```



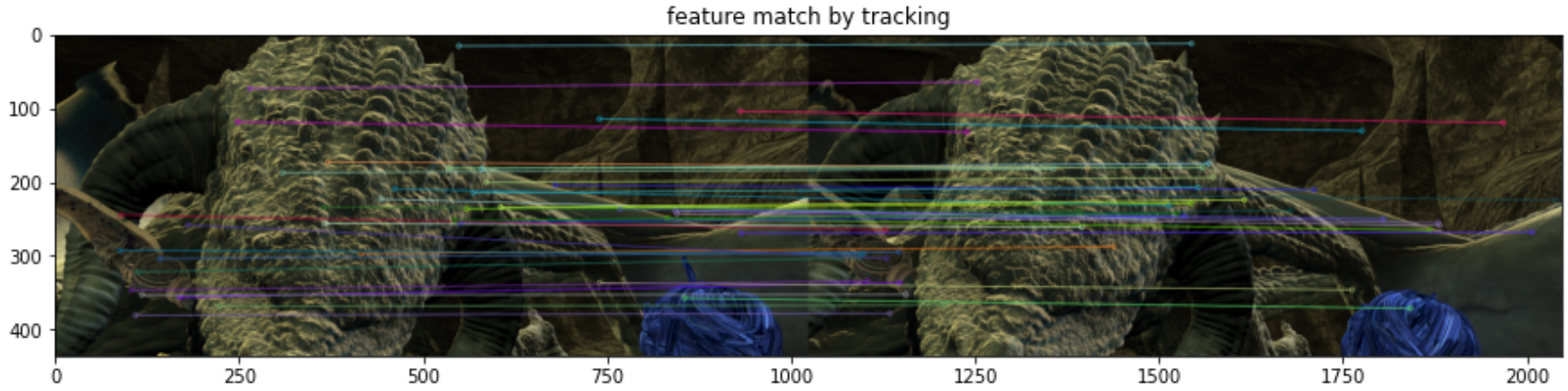

```
frame 28: good points 59
```



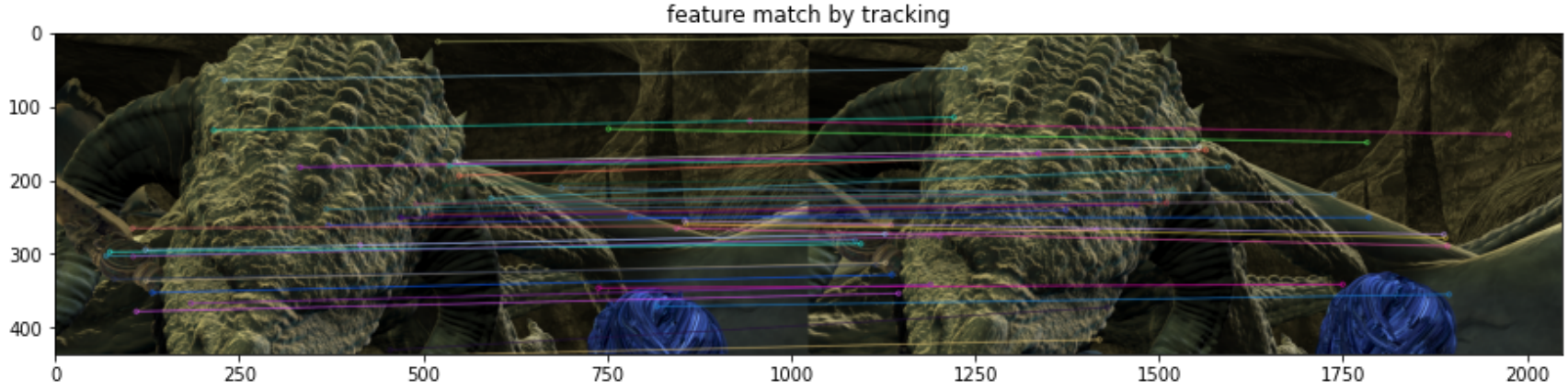

```
frame 29: good points 46
```



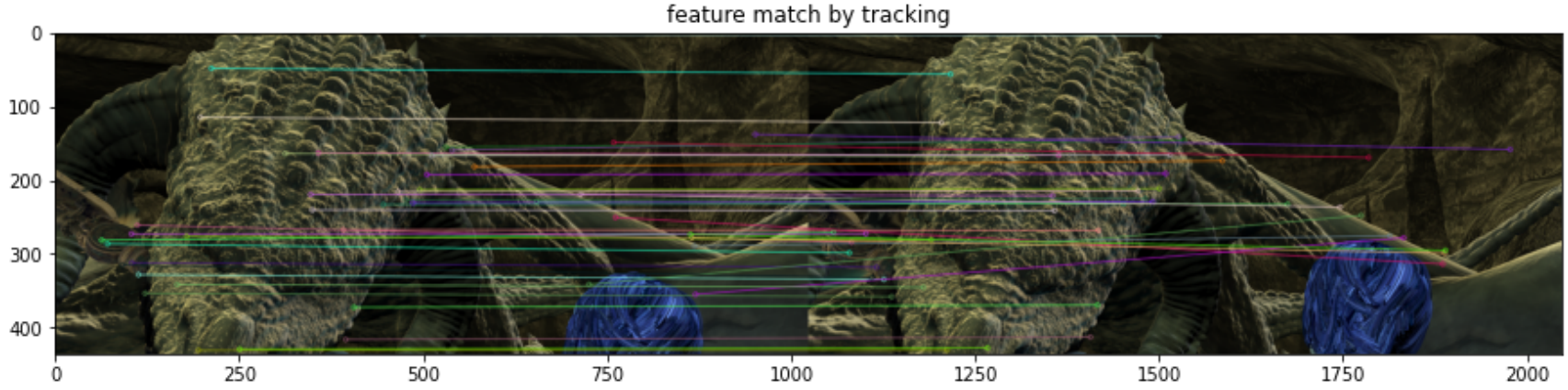

```
frame 30: good points 40
```



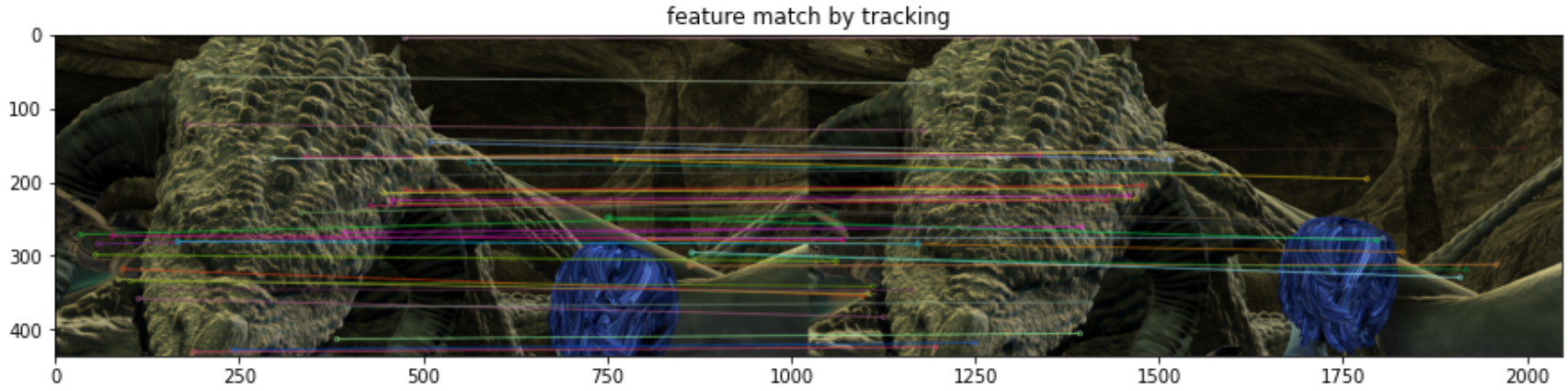

```
frame 31: good points 40
```



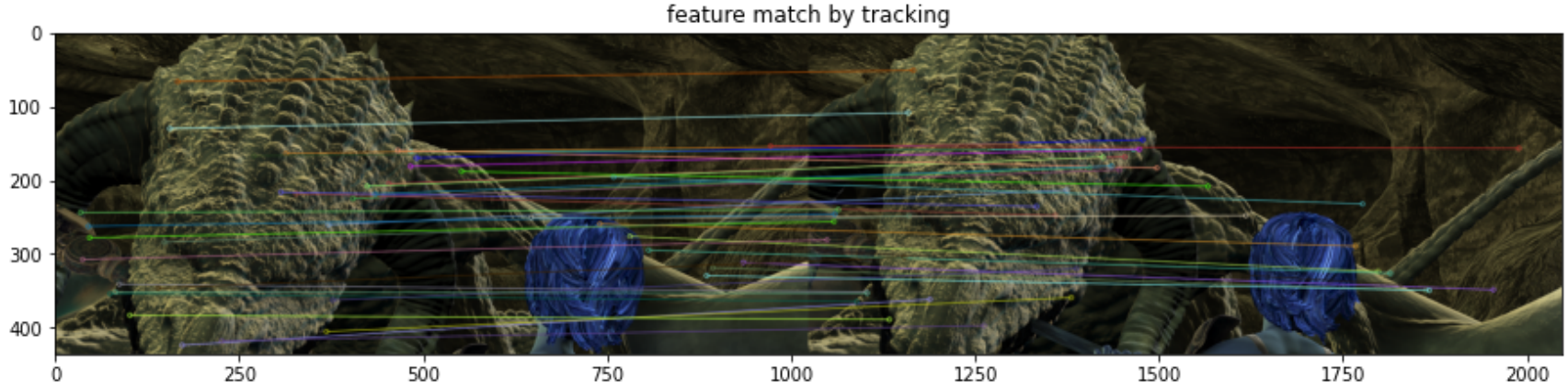

```
frame 32: good points 42
```



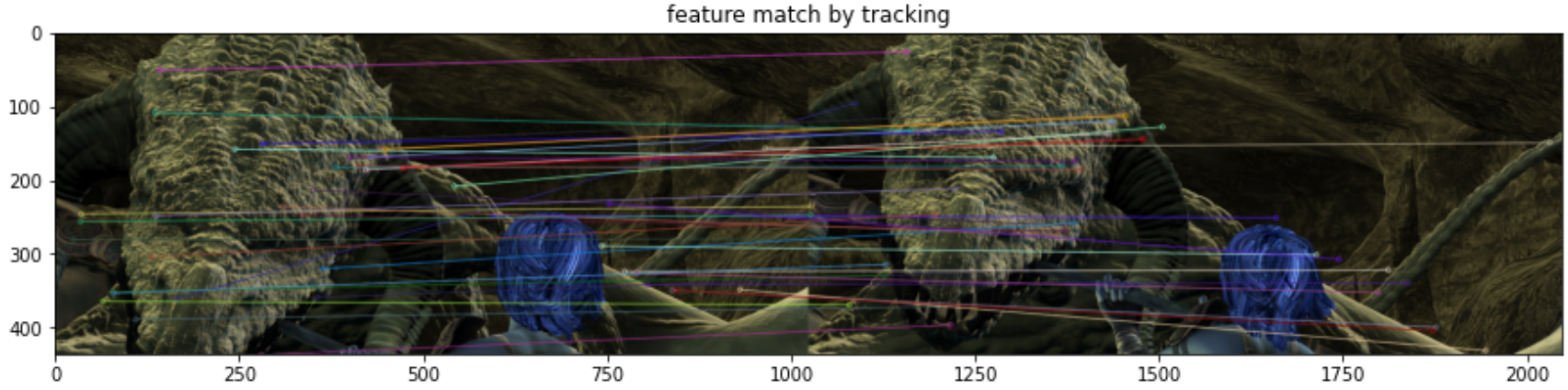

```
frame 33: good points 42
```



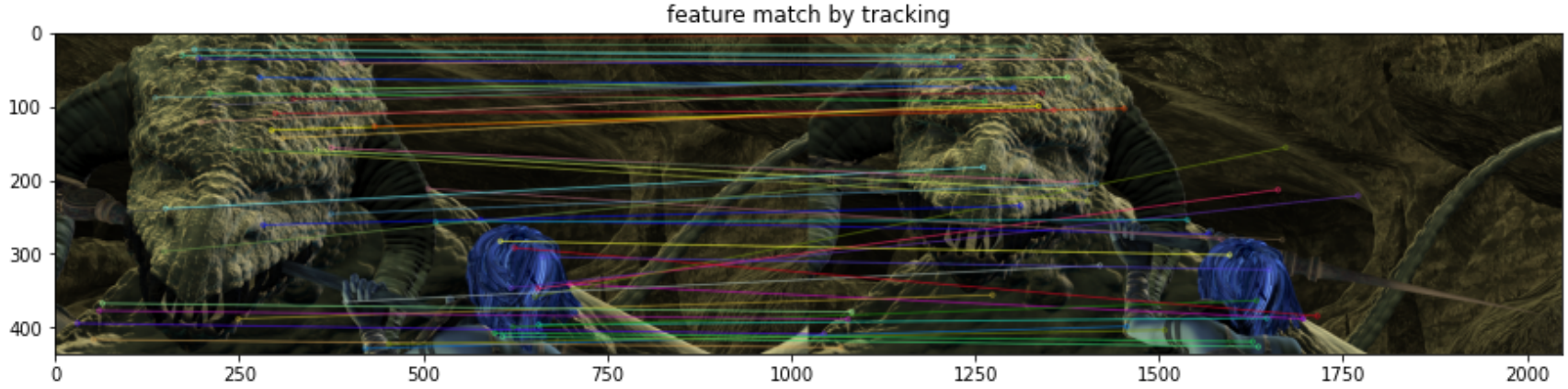

```
frame 34: good points 41
```



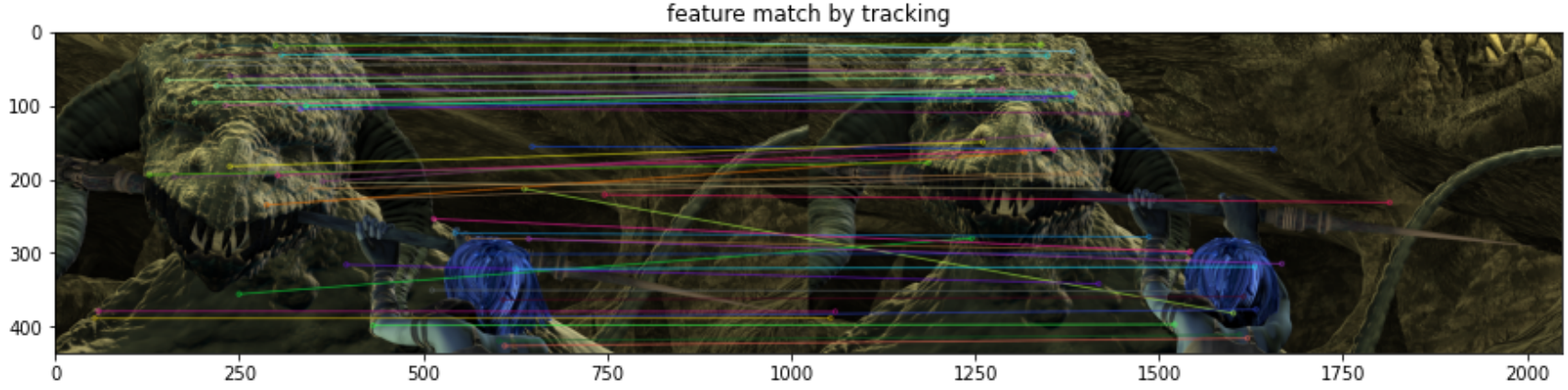

```
frame 35: good points 40
```



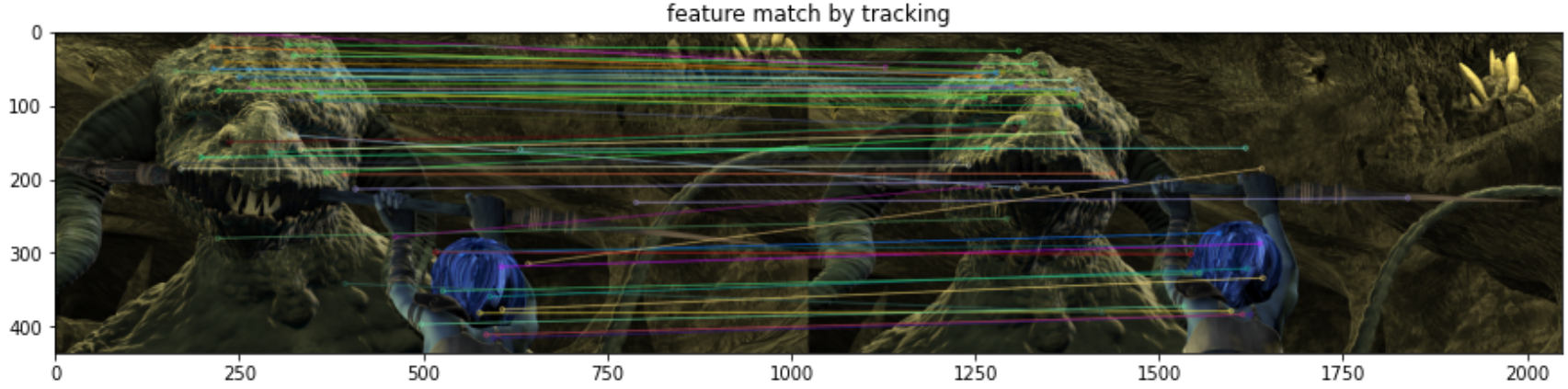

```
frame 36: good points 38
good points is lower than 40, recalculating...
```



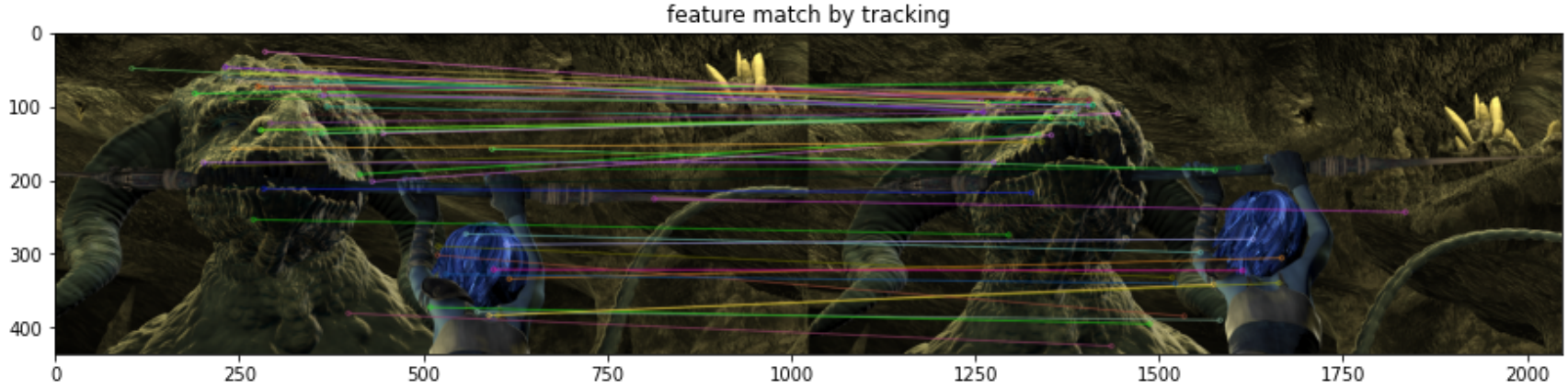

```
frame 37: good points 90
```



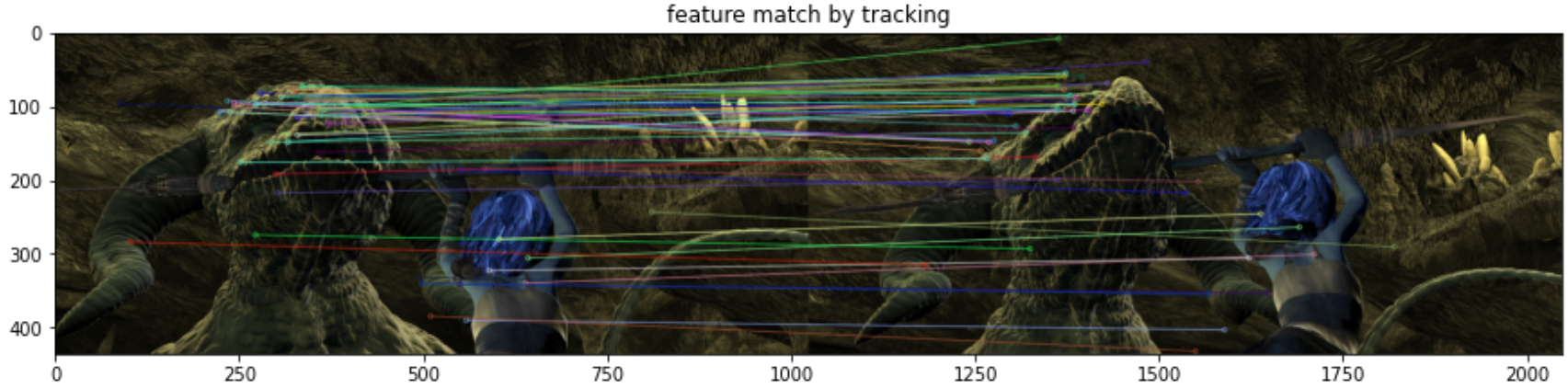

```
frame 38: good points 87
```



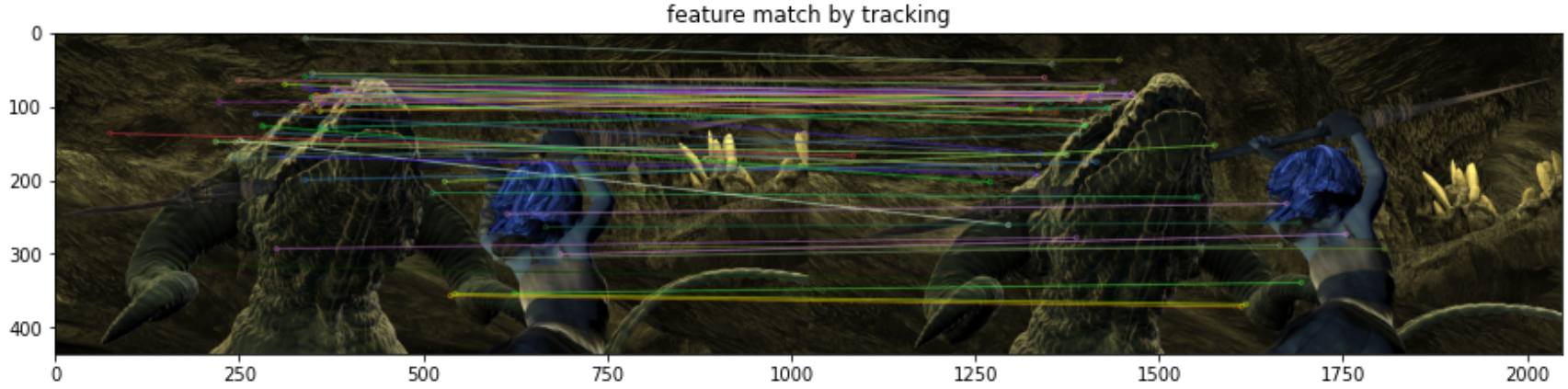

```
frame 39: good points 87
```



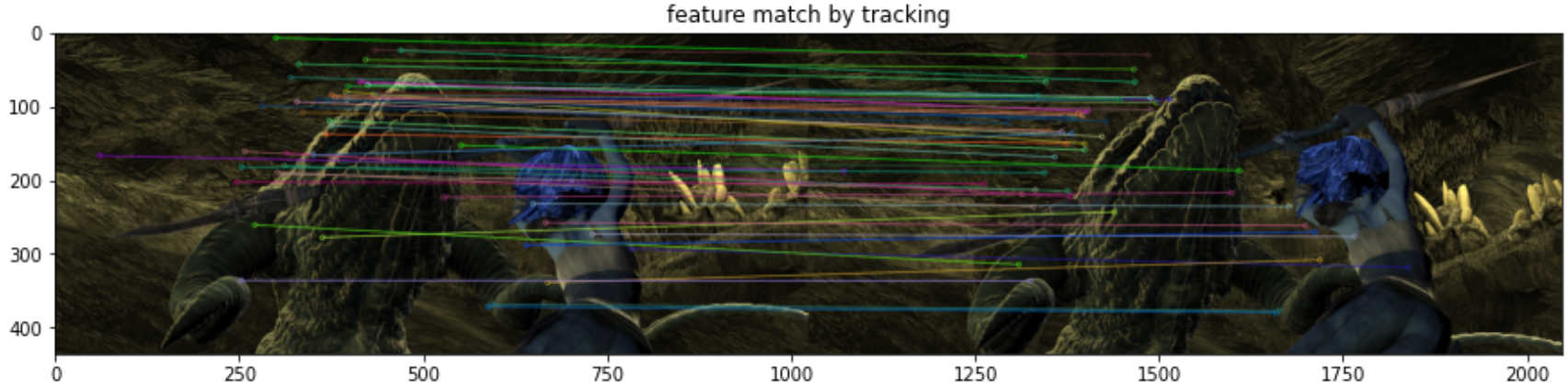

```
frame 40: good points 78
```



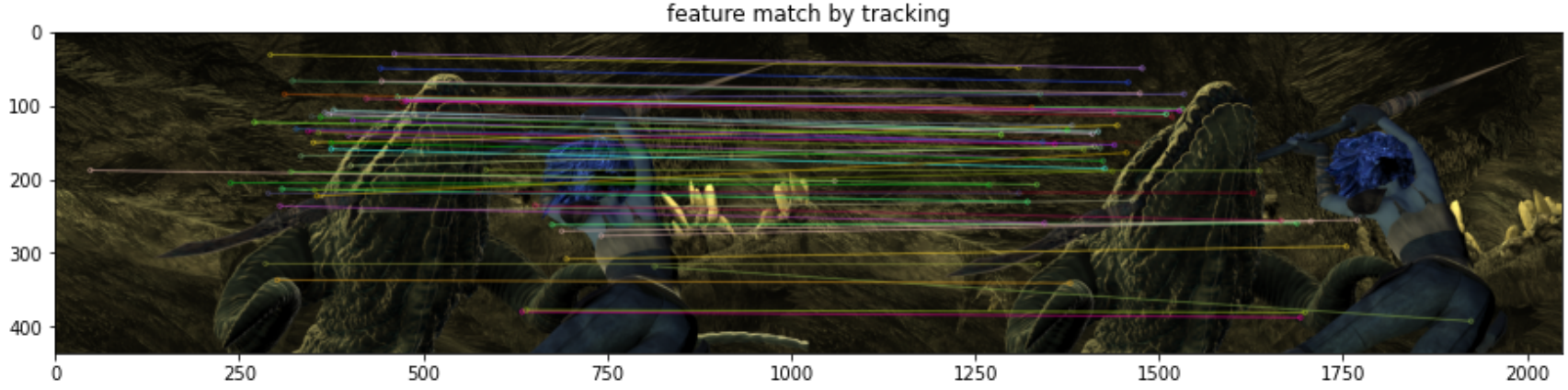

```
frame 41: good points 75
```



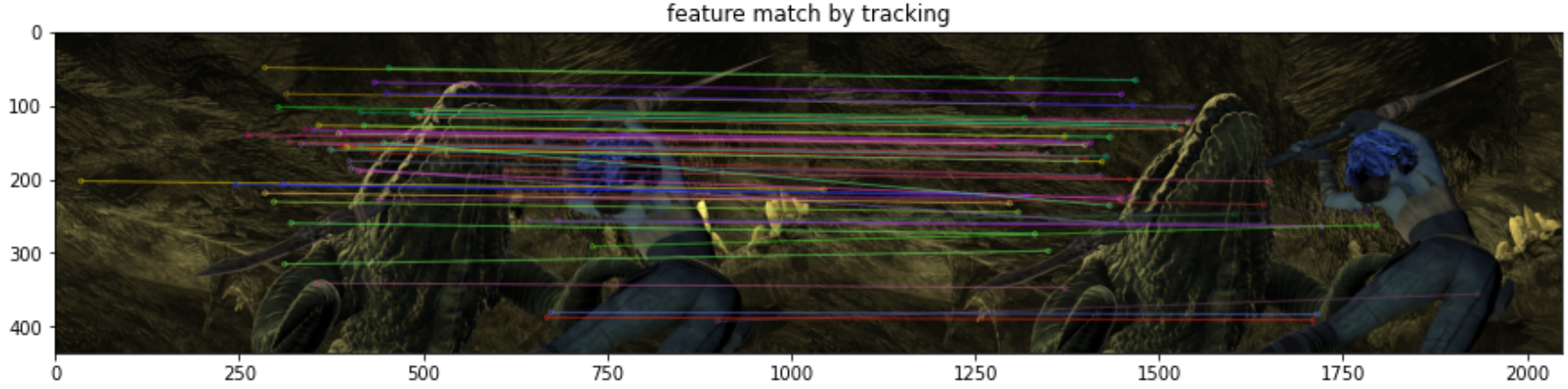

```
frame 42: good points 79
```



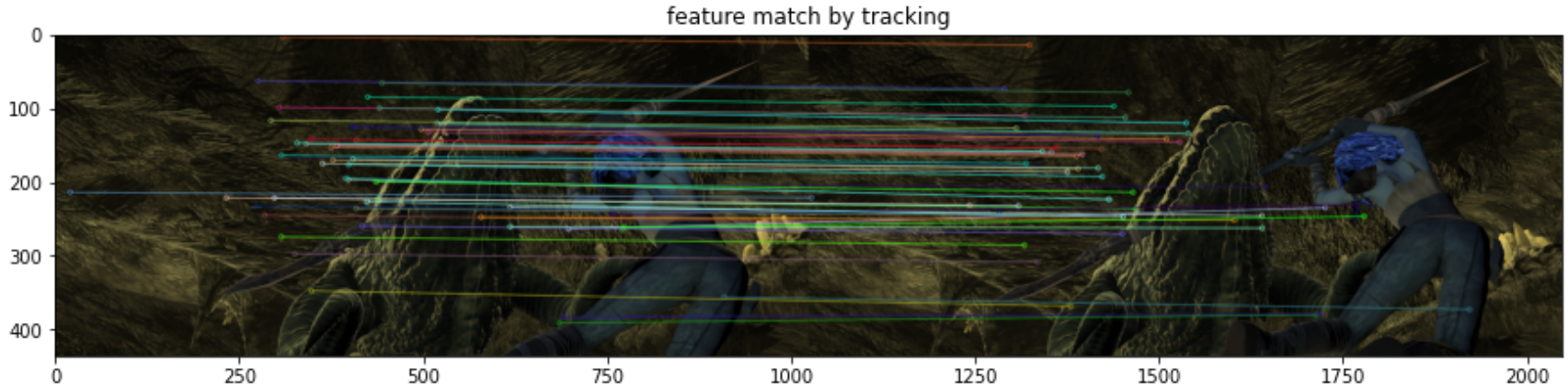

```
frame 43: good points 80
```



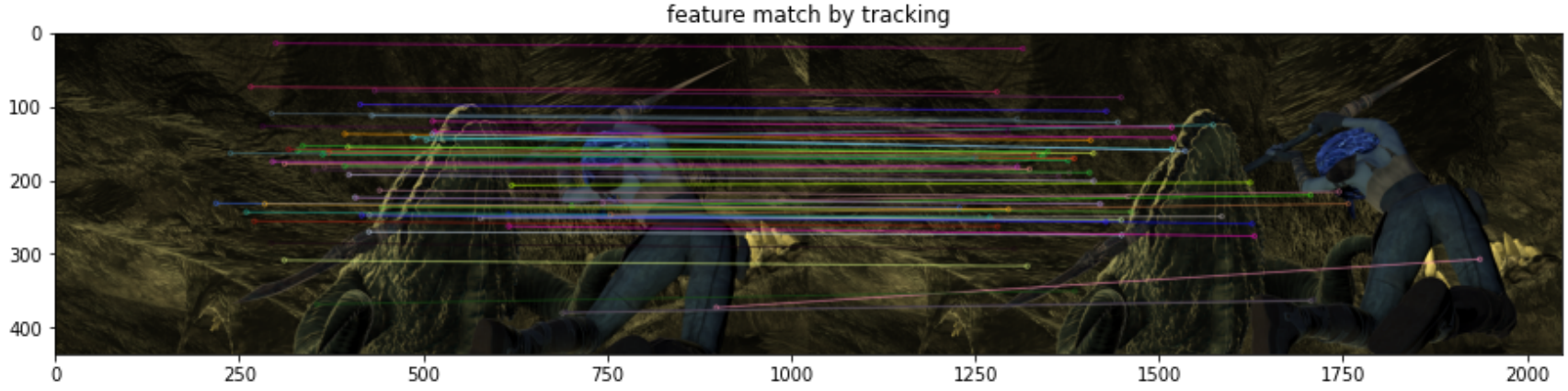

```
frame 44: good points 80
```



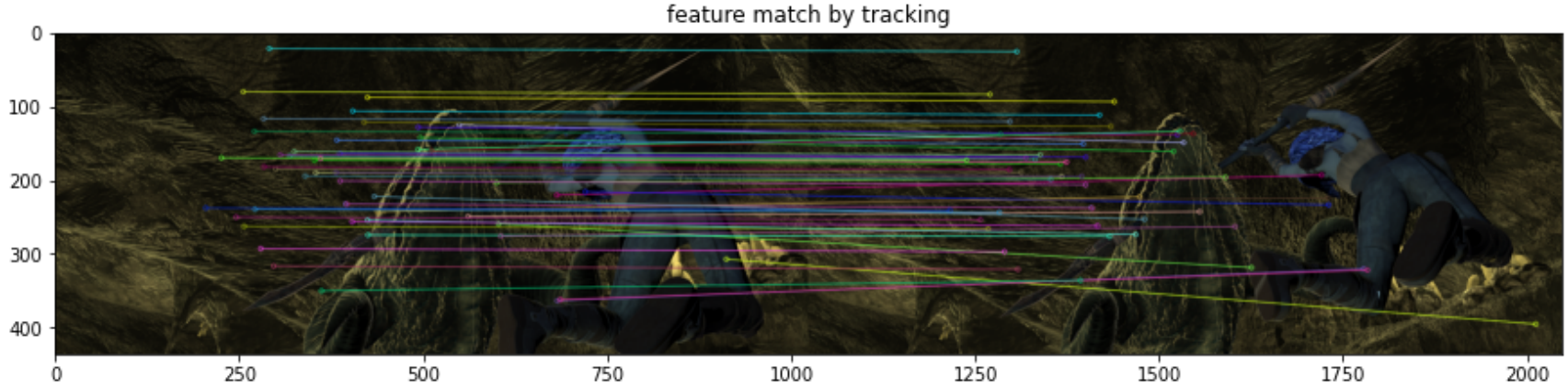

```
frame 45: good points 81
```



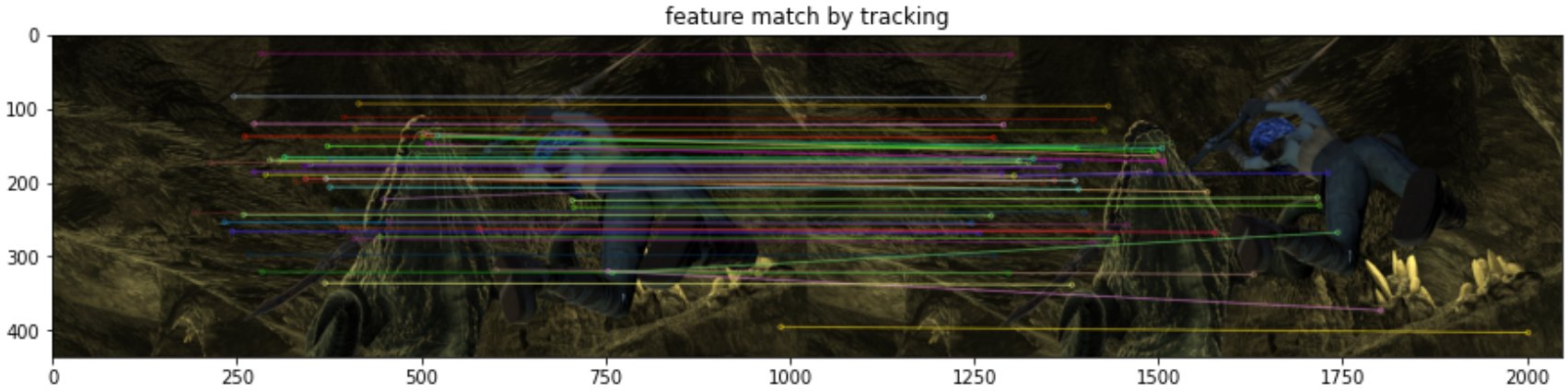

```
frame 46: good points 80
```



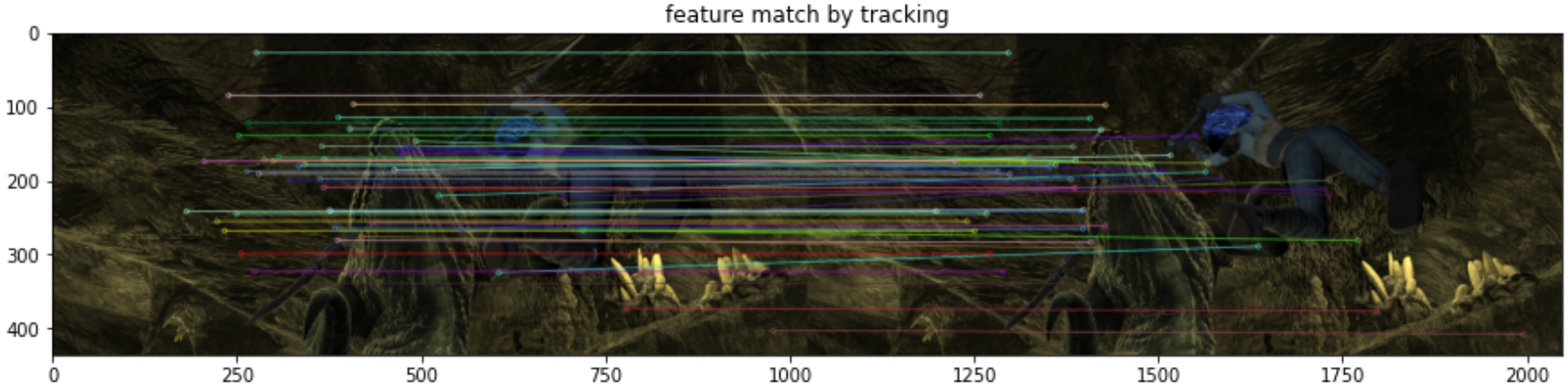

```
frame 47: good points 80
```




```
frame 48: good points 80
```




```
frame 49: good points 80
```



Question 2

Q2. (40 points) Optical flow: (use the motion sequences available at https://vision.middlebury.edu/flow or http://sintel.is.tue.mpg.de) Compute optical flow (spline-based or per-pixel) between two images, using one or more of the techniques described in the lecture.

a. (15 points) Implement the Lucas-Kanade algorithm (your code).

b. (25 points) Visualize the optical flow in two video sequences, provide some examples where Lucas-Kanade fails. Explain the reason in each case.

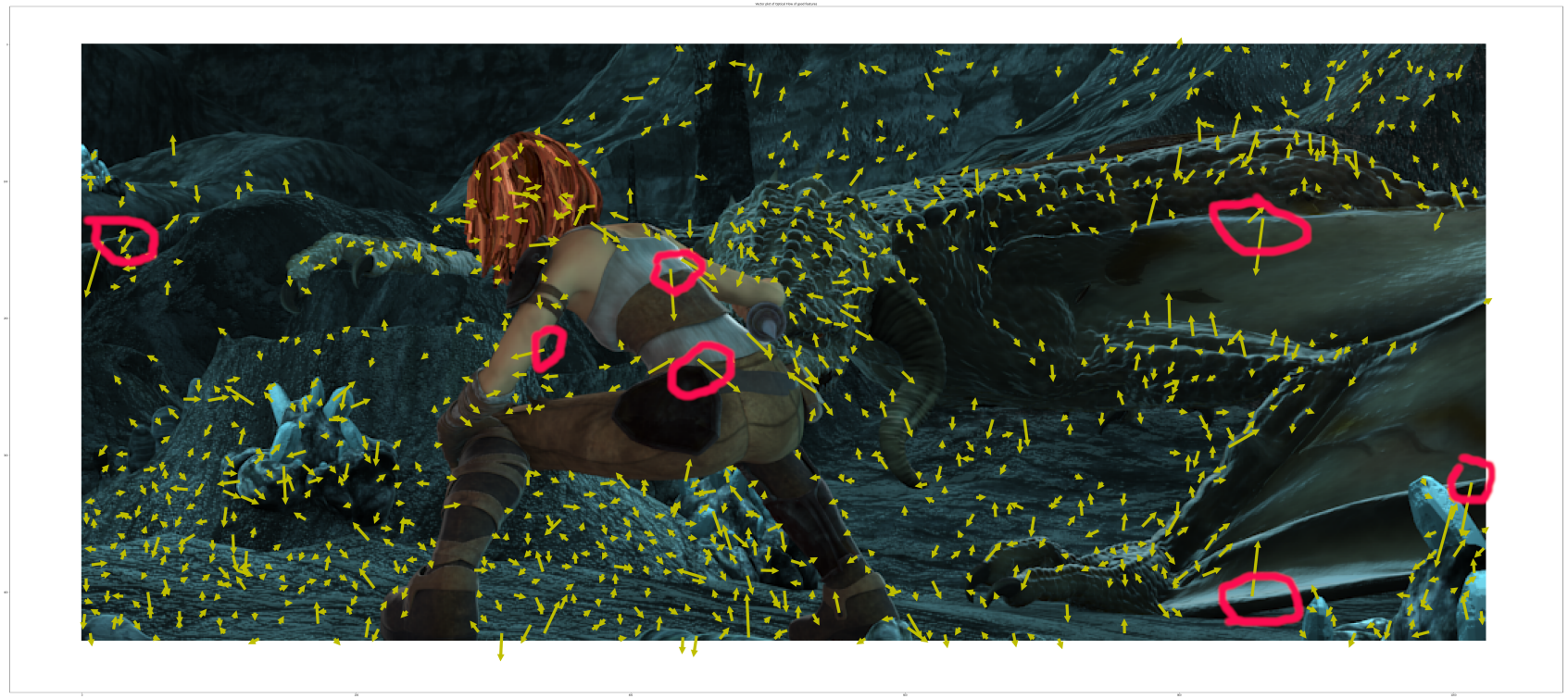
Answer:

a. function `LK_OpticalFlow` implement Lucas-Kande algorithm.

b. function `display_opitical_flow` shows 5 images(processed 6 frames) as the result. The output number can be adjusted in code.

The examples where Lucas-Kanade fails is shown in figure 1. The main reason is because of aperture problem. And this problem can be fixed by adding pyramid function to calcualte optical flow for each level.

figure 1



The algorithm may fail if the movement is too big, or the light indensity chagne too much. But it does not show the failure on this example frame, because it did not run into the condition.

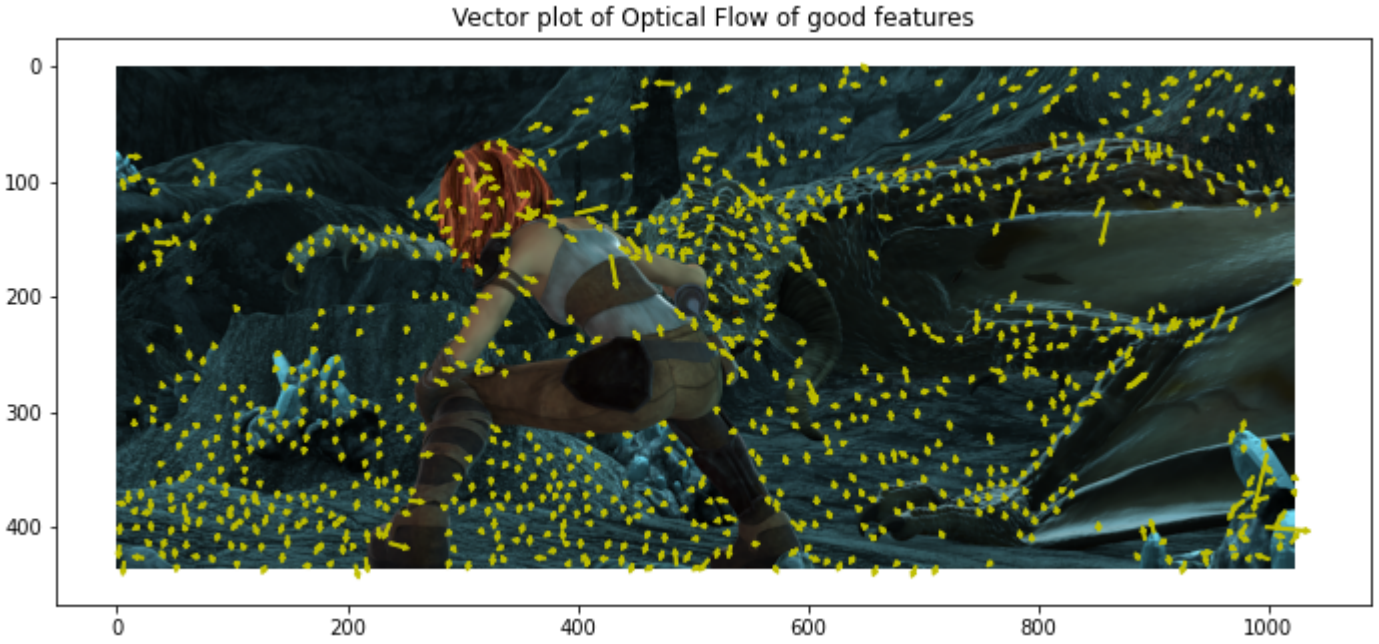
In [ ]:

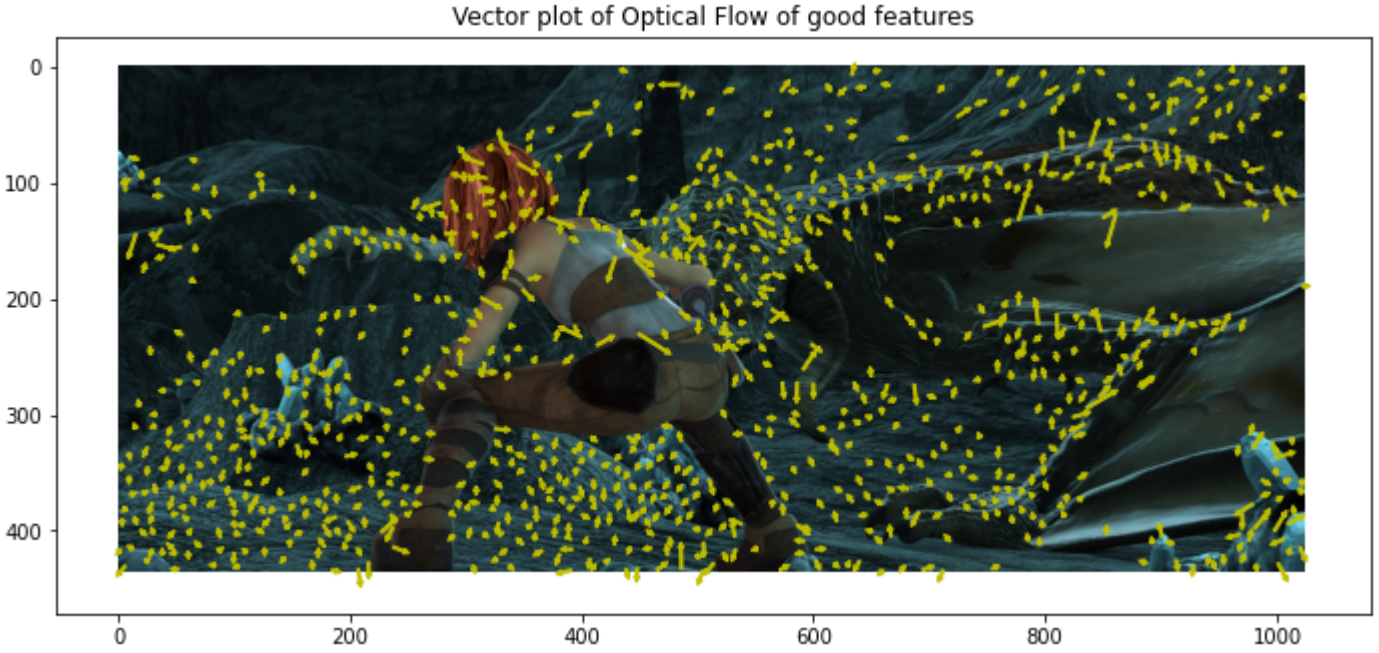
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
from imutils import paths

#2a Lucas-Kanade optical flow, return u, v (dx,dy)
def LK_OpticalFlow(gray1,gray2,win_size=(3,3)):
    '''
    This function implements the LK optical flow without pyramid or consider as level 1 pyramid.
    '''
```

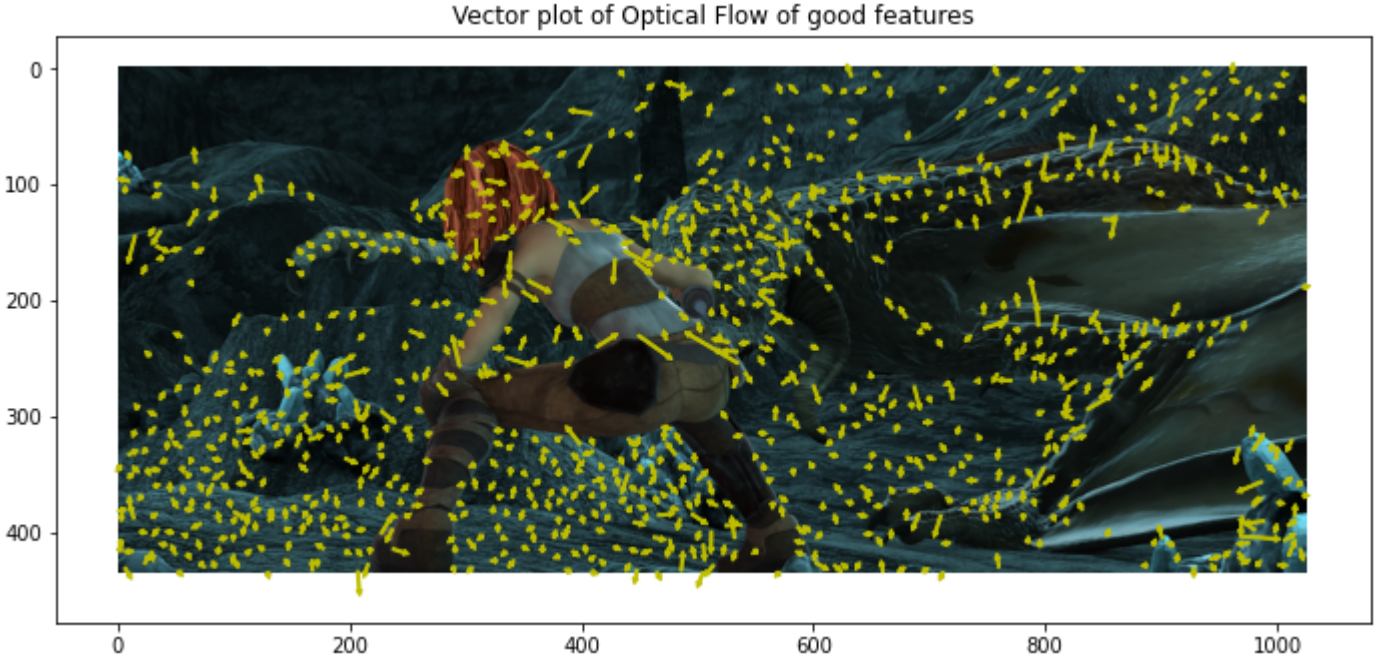
```
    # 1. apply Guassian filter to smooth the image in order to remove noise
    I1 = cv2.GaussianBlur(gray1, win_size, 0)
    I2 = cv2.GaussianBlur(gray2, win_size, 0)
    m,n = win_size
    m = m//2
    n = n//2

    # Calculate Ix, Iy, It
    # The filter multiple 1/4 for average result
    filter_x = np.array([[-1,1],[-1,1]]) * 0.25
    filter_y = np.array([[-1,-1],[1,1]]) * 0.25
    filter_t1 = np.array([[1,1],[1,1]]) * 0.25
    filter_t2 = np.array([[1,1],[1,1]]) * -1 * 0.25

    Ix = signal.convolve2d(I1,filter_x,'same') + signal.convolve2d(I2,filter_x,'same')
    Iy = signal.convolve2d(I1,filter_y,'same') + signal.convolve2d(I2,filter_y,'same')
    It = signal.convolve2d(I1,filter_t1,'same') + signal.convolve2d(I2,filter_t2,'same')

    # retrieve good features to track
    points = cv2.goodFeaturesToTrack(I1,1000, 0.01,10)
    points = np.int0(points)

    #init u, v
    u = np.zeros(gray1.shape)
    v = np.zeros(gray1.shape)

    # todo:
    # Calculating the u and v arrays for the good features obtained n the previous step.
    kp = list()
    status = list()
    err = list()
    for p in points:
        # nparray image format is h, w(y,x)
        y,x = p.ravel()

        # calculating the derivatives for the neighbouring pixels
        # windows size mxn

        IX = Ix[x-m:x+m+1,y-n:y+n+1].flatten()
        IY = Iy[x-m:x+m+1,y-n:y+n+1].flatten()
        IT = It[x-m:x+m+1,y-n:y+n+1].flatten()

        # Resolve minimum least squares equation
        # x = inverse(At dot A) dot (At dot b)
```

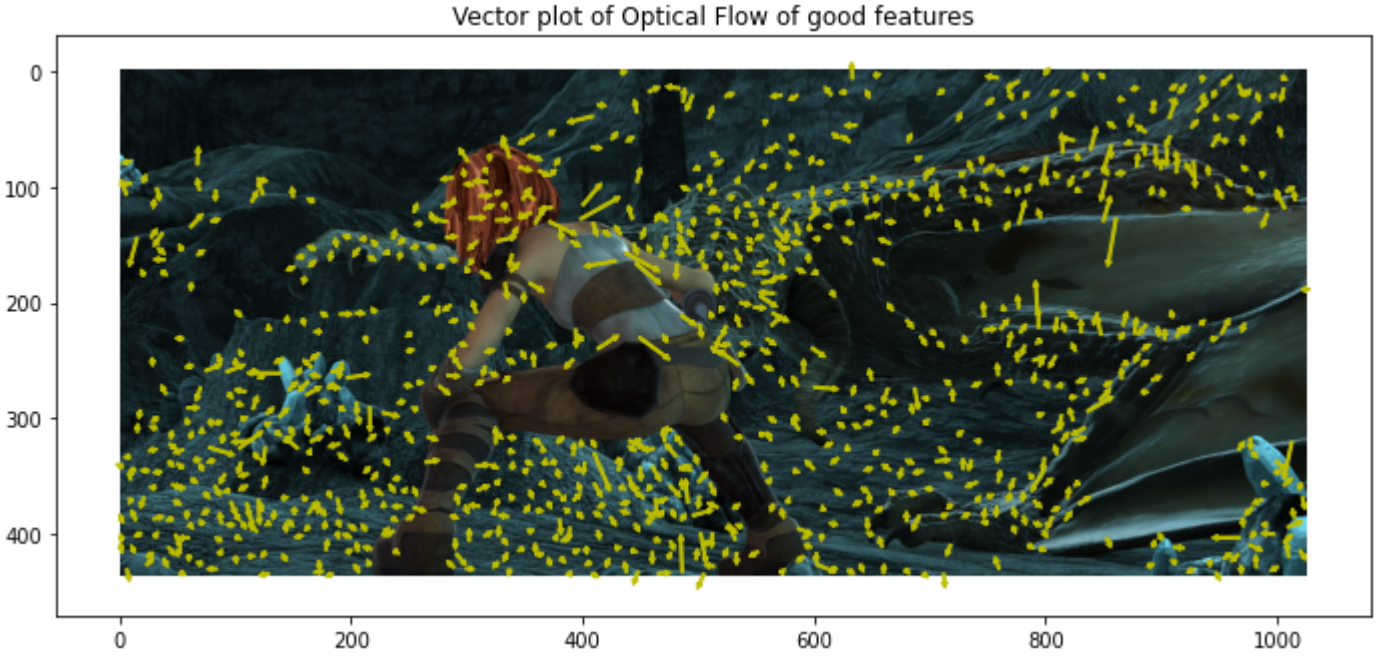
```
        # np.concatenate((IX,IY),axis=1)
        At = np.matrix((IX,IY))
        A = np.matrix.transpose(At)
        A = np.array(A)
        At = np.array(At)
        b = np.transpose(np.array(IT))

        At_dot_A = np.dot(At,A)
        A_inv = np.linalg.pinv(At_dot_A)
        B = np.dot(At,b)

        X = np.dot(A_inv,B)
        u[x,y] = X[0]
        v[x,y] = X[1]
        if X[0] != 0 or X[1] != 0:
            status.append(1)
        else:
            status.append(0)
        kp.append(p)
        kp.append(0)

        # print(f"({x,y}): {X[0], X[1]}")

    return (u, v, points, status, err)

# 2b Visualize result
def display_opitical_flow(image_folder, show_n=10, threshold = 0.5):
    imagePaths = sorted(list(paths.list_images(image_folder)))
    images = []

    # images to stitch list
    for imagePath in imagePaths:
        image = cv2.imread(imagePath)
        images.append(image)

    img1 = images[0]
    gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    n = min(len(images),show_n)+1
    for i in range(1,n):
        img2 = images[i]
        gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
        (u, v, points, status, err) = LK_OpticalFlow(gray1,gray2, win_size=(3,3))

        # display result
```

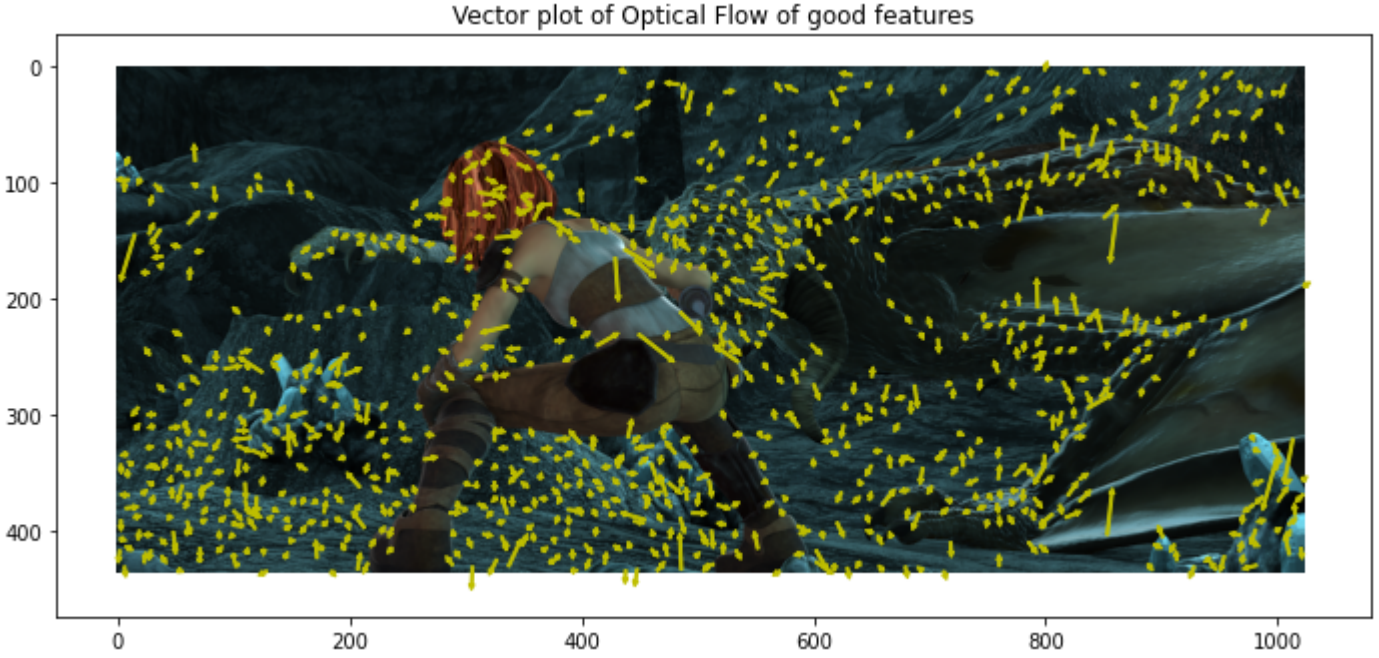
```
        img = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
        plt.figure(figsize=(40,30))
        plt.subplot(1,3,3)
        plt.title('Vector plot of Optical Flow of good features')
        plt.imshow(img)
        for i in range(len(points)):
            p = points[i]
            y,x = p.ravel()
            # only show points value > threshold
            if status[i] == 1 and(abs(u[x,y])>threshold or abs(v[x,y])>threshold):
                plt.arrow(y,x, v[x,y], u[x,y], width=2,head_width = 5, head_length = 5, color = 'y')

        plt.show()

        img1 = img1
        gray1 = gray1

# 2 Results
image_folder = "./data/cave_3"
display_opitical_flow(image_folder,5,0.5)
```




Vector plot of Optical Flow of good features
0
100
200
300
400
0
200
400
600
800
1000


Vector plot of Optical Flow of good features
0
100
200
300
400
0
200
400
600
800
1000





Q3. (20 points) Head detection:

You are asked to implement an algorithm that can detect head in images and videos. The algorithm should detect head regardless of viewing direction of the camera. What is your suggestion? Try to eliminate false positives as much as you can.

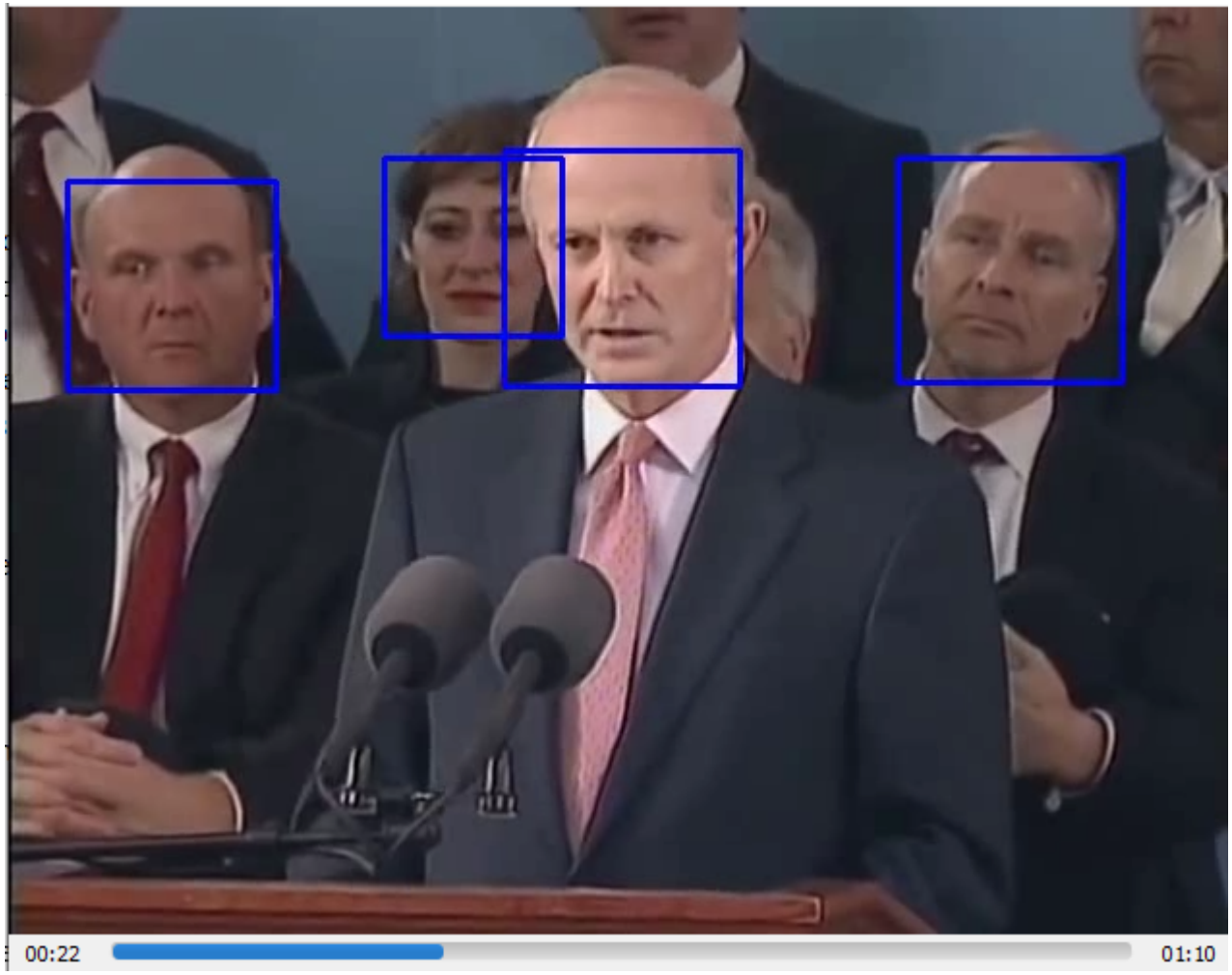
Answer

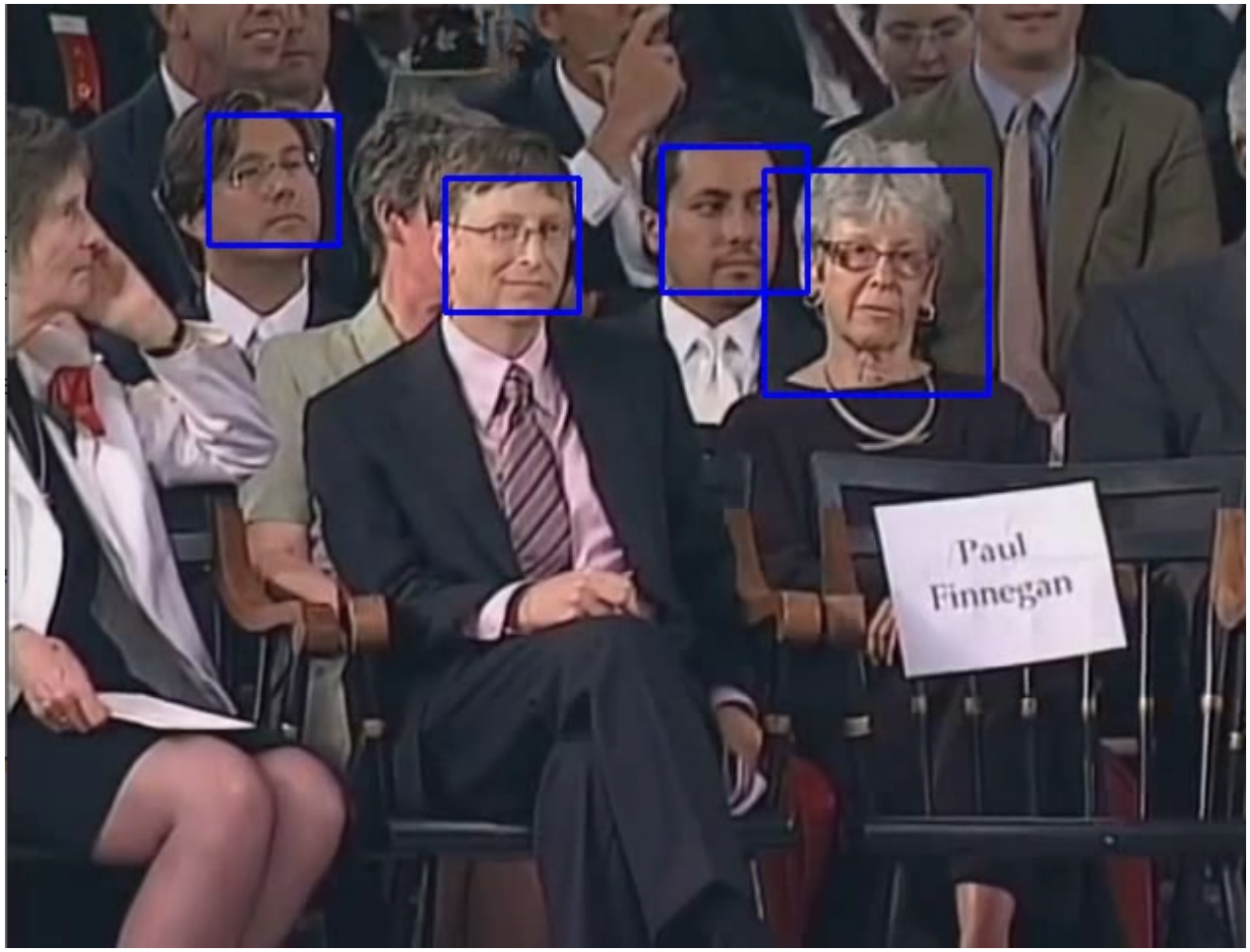
1) The first option I thought about is to contour detection. The assumption is that the head is oval shape. If I can get contour first, then I can filter oval shape and I will get the head mask. I wrote a test program to detect the contour by using `CV2.findContours` . The code is in `head.py` and the function name is `contour_detect` . But the result does not work well, because the intensity are changing in the video and it could not easily get contour of a head. I gave up this option. 2) The next option I used is `Haar cascade object detection` method. Haar feature-based cascade classifiers is proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. There are three types of Harr features are defined and the whole process requires a training in order to do object detection. In the code I am using pre-trained `haarcascade_frontalface_default.xml` features for the detection. The code is shown in next python code cell.

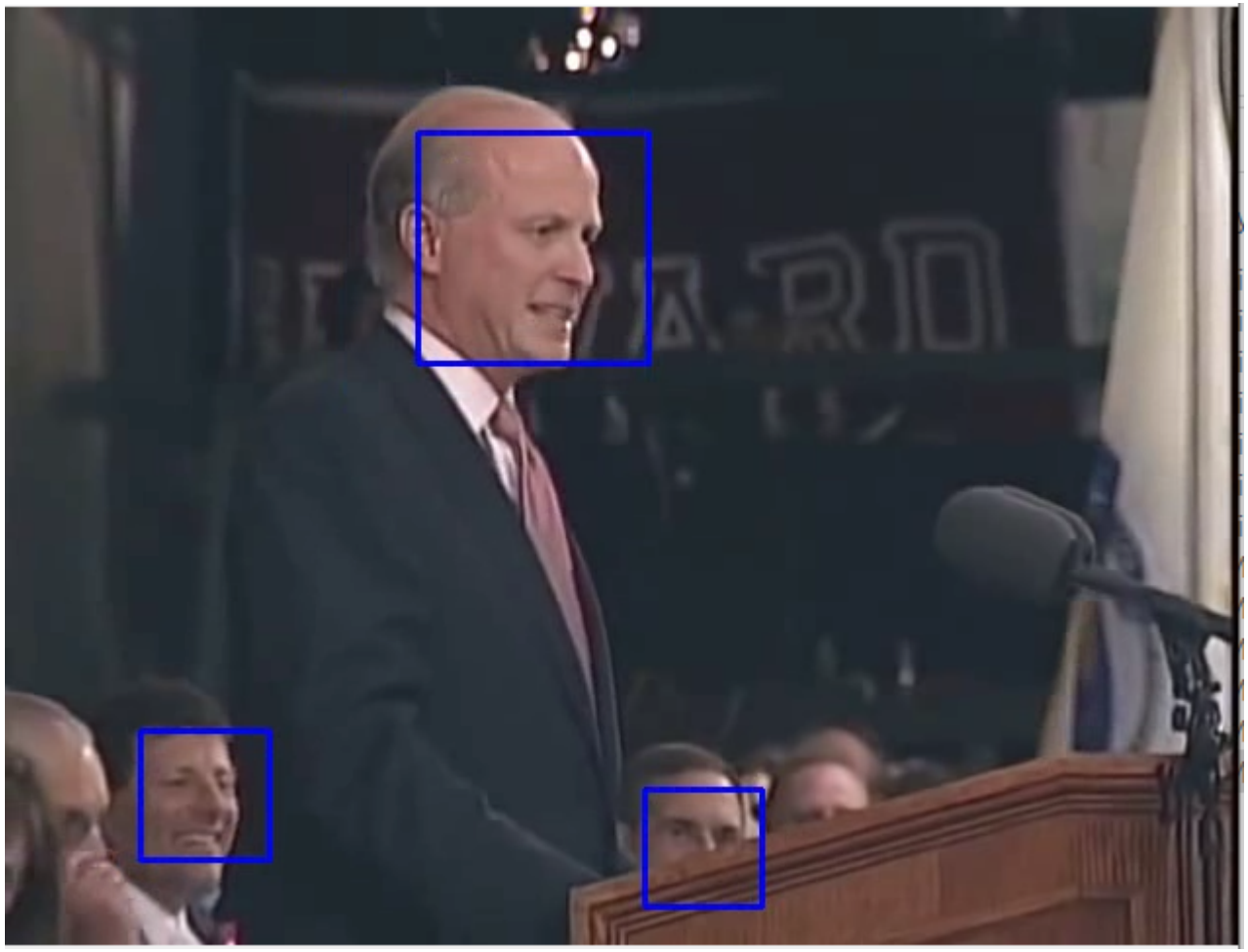
Result 1

1) The result from default parameters can be watched from Youtube. The url is https://youtu.be/EjcGXcOtiMw.

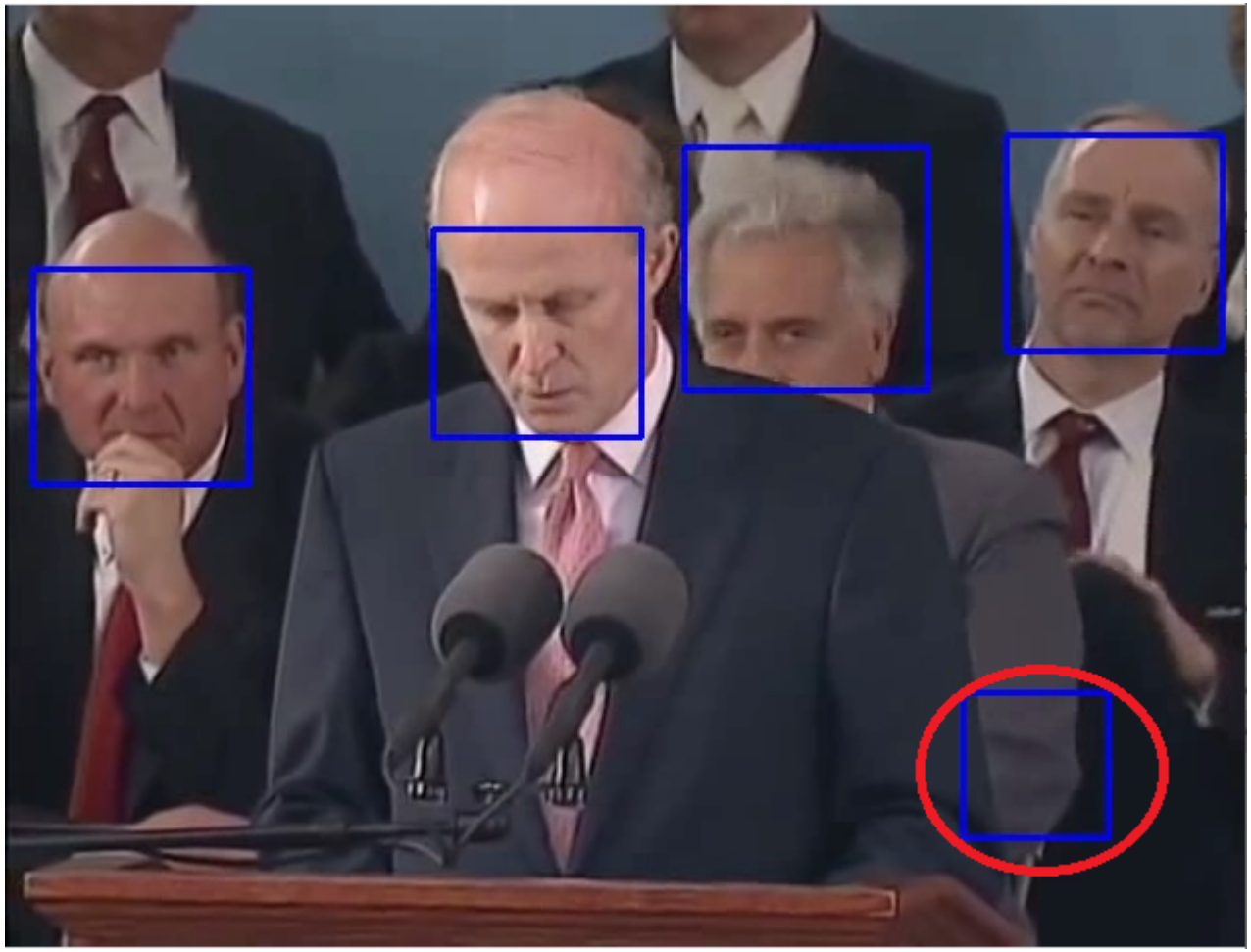
True Positive results:




Paul
Finnegan



False Positive results:



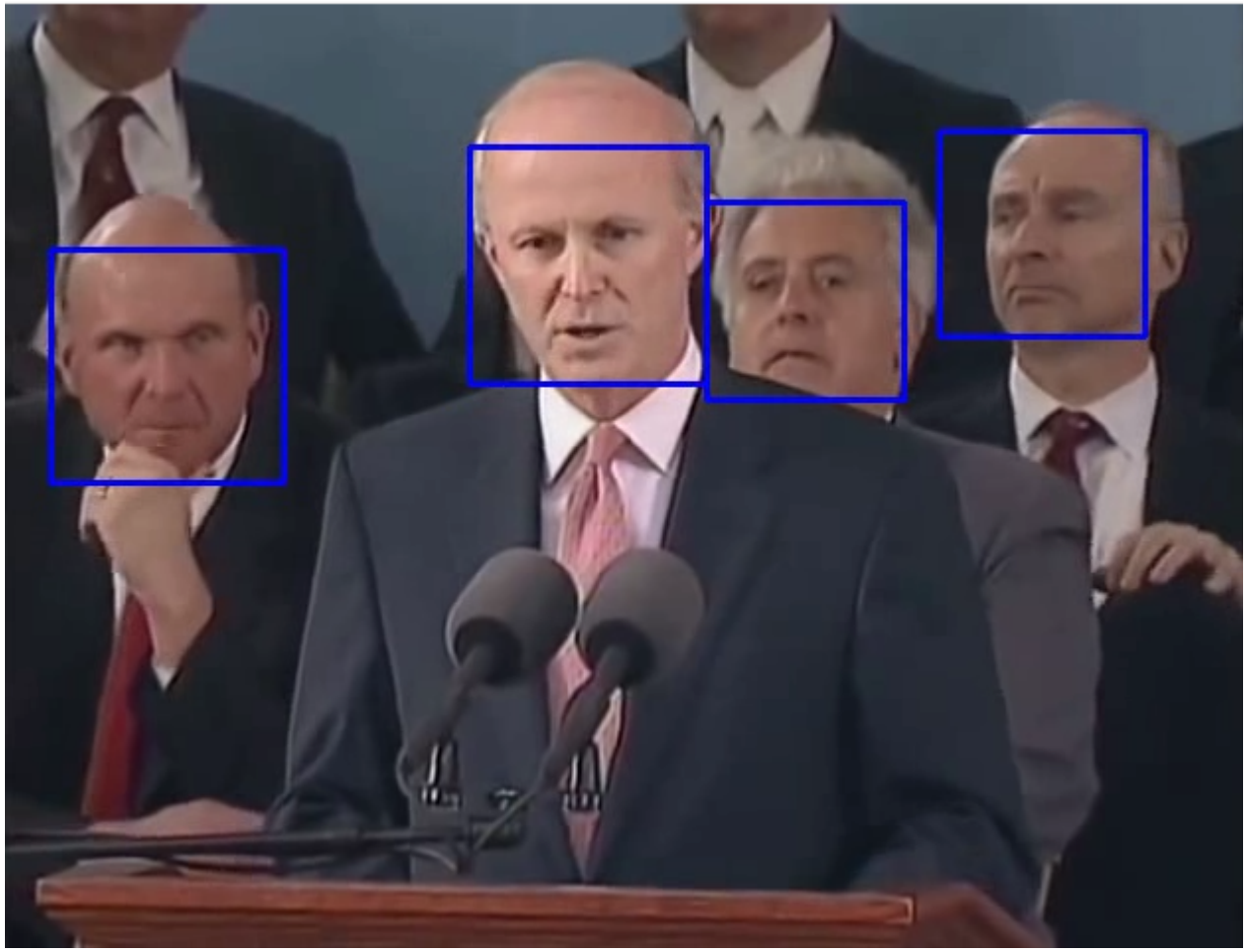
Reduce false positive

After testing there are a few things impact on false positive results.

1) Harr features are capturing features changes patterns. For example, left eye, nose, right eye has a pattern darker,bright, darker. These patterns togethers represents the object features. In our case it is face,part of head. This is fundamental assumptions and it will happen if any pattern matches with the trained classifier. 2) There are two parameters used in detecting stage, scaleFactor=1.1 and minNeighbors=3. The minNeighbors impacts false positive and scaleFactor impact both searching performance and false positive result. After tuned, the minNeighbors is set to 5(default 3) and scaleFactor is set to 1.3(default 1.1). 3) Another behavior is that the image size,

also has impact on the searching performance and false positive. In my code I resize the frame size to 1/3 of original to acquire better performance and lower false positive.

The video can be seen from YouTube and the url is https://youtu.be/-FnyaVMCaUk. And two screenshots is shown here.



Q3 Code

To run the code please make sure you have the video files comes with this project. The video file is under data folder.

In [ ]:

```
import numpy as np
import cv2

# org video is from https://www.youtube.com/watch?v=zPx5N6Lh3sw&t=1276s
# ffmpeg -i BillGates.mp4 -ss 00:00:05 -t 00:10:00 -async 1 cut.mp4
# ffmpeg -i cut.mp4 -vf scale=320:240 small.mp4

def harr_cascade_detect(harr_cascade,frame):
```

```
    # detection, inside the detection the frame data is changed with
    # bounding box and returned
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    a = 2
    gray = cv2.resize(gray, dsize=(0, 0), fx=1/a, fy=1/a)
    heads = harr_cascade.detectMultiScale(gray,scaleFactor=1.3,minNeighbors=5)

    for (x,y,w,h) in heads:
        cv2.rectangle(frame,(a*x,a*y),(a*x+a*w,a*y+a*h),(255,0,0),2)

    return frame

def harr_casscade_mp4():
    xml = 'haarcascade_frontalface_default.xml'
    # xml = 'haarcascade_upperbody.xml'

    harr_cascade = cv2.CascadeClassifier(f'./data/cv2/{xml}')
    video_name = "cut1.mp4"
    # cut2 does not work.
    vid_capture = cv2.VideoCapture(f'./data/{video_name}')

    if (vid_capture.isOpened() == False):
        print("Error opening the video file")
        return

    # get video meta info
    fps = vid_capture.get(5)
    frame_count = vid_capture.get(7)
    frame_width = int(vid_capture.get(3))
    frame_height = int(vid_capture.get(4))
    frame_size = (frame_width,frame_height)
    # fps = 20
    print(f"Video Info:")
    print(f"width:{frame_width} height:{frame_height} FPS: {fps} Count: {frame_count}")

    # output to mp4
    output = cv2.VideoWriter(f'./data/output_{video_name}', \
        cv2.VideoWriter_fourcc(*'mp4v'), fps, frame_size)

    while(vid_capture.isOpened()):
        # vid_capture.read() methods returns a tuple, first element is a bool
        # and the second is frame
        ret, frame = vid_capture.read()
        if ret == True:
```

```
            # detection, inside the detection the frame data is changed with
            # bounding box and returned
            frame = harr_cascade_detect(harr_cascade,frame)
            # frame = contour_detect(frame)

            cv2.imshow('Frame',frame)
            output.write(frame)

            # press q to quit
            key = cv2.waitKey(20)
            if key == ord('q'):
                break
        else:
            break

    vid_capture.release()
    output.release()
    cv2.destroyAllWindows()

harr_casscade_mp4()
```

```
Video Info:
width:632 height:480 FPS: 29.97002997002997 Count: 5005.0
```