In [1]:

```
from absl import logging
import tensorflow as tf
import numpy as np
from skimage.feature import plot_matches
from PIL import Image,ImageOps
from scipy.spatial import cKDTree
from skimage.measure import ransac
from skimage.transform import AffineTransform
from six import BytesIO
import tensorflow_hub as hub
from six.moves.urllib.request import urlopen
```

In [2]:

```
images = 'Golden Gate'
if images == 'Bridges of sigh':
    Image1_url =
'https://upload.wikimedia.org/wikipedia/commons/2/28/Bridge_of_Sighs%2C_Oxford.jpg'
    Image2_url=
'https://upload.wikimedia.org/wikipedia/commons/c/c3/The_Bridge_of_Sighs_and_Sheldonain_Theatre%2c
_Oxford.jpg'
elif images == 'Golden Gate':
    Image1_url = 'https://upload.wikimedia.org/wikipedia/commons/1/1e/Golden_gate2.jpg'
    Image2_url  = 'https://upload.wikimedia.org/wikipedia/commons/3/3e/GoldenGateBridge.jpg'

elif images == 'Acropolis':
    Image1_url  =
'https://upload.wikimedia.org/wikipedia/commons/c/ce/2006_01_21_Ath%C3%A8nes_Parth%C3%A9non.JPG'
    Image2_url  = 'https://upload.wikimedia.org/wikipedia/commons/5/5c/ACROPILIS_1969_-
_paanoramio_-_jean_melis.jpg'
```

In [3]:

```
def download_and_resize(name,url,new_width=256,new_height=256):
    path  = tf.keras.utils.get_file(url.split('/')[-1],url)
    image = Image.open(path)
    image = ImageOps.fit(image,(new_width,new_height),Image.ANTIALIAS)
    return image
```

In [4]:

```
import matplotlib.pyplot as plt
```

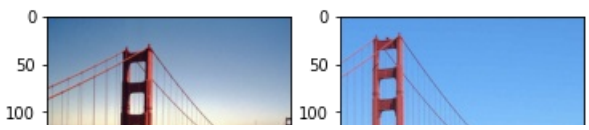
In [5]:

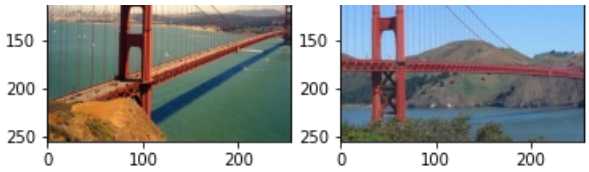
```
image1 = download_and_resize('image1.jpg',Image1_url)
image2 = download_and_resize('image2.jpg',Image2_url)

plt.subplot(1,2,1)
plt.imshow(image1)
plt.subplot(1,2,2)
plt.imshow(image2)
```

```
Downloading data from https://upload.wikimedia.org/wikipedia/commons/1/1e/Golden_gate2.jpg
237568/231506 [==============================] - 0s 1us/step
Downloading data from https://upload.wikimedia.org/wikipedia/commons/3/3e/GoldenGateBridge.jpg
98304/90709 [==============================] - 0s 1us/step
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x7ffa887c7110>
```





In [6]:

```
delif = hub.load('https://tfhub.dev/google/delf/1').signatures['default']
```

In [7]:

```
def run_delf(image):
    np_image = np.array(image)
    float_image = tf.image.convert_image_dtype(np_image,tf.float32)

    return delif(
        image = float_image,
        score_threshold = tf.constant(100.0),
        image_scales = tf.constant([0.25,0.3536,0.5,0.7071,1.0,1.4142,2.0]),
        max_feature_num = tf.constant(1000))
```

In [8]:

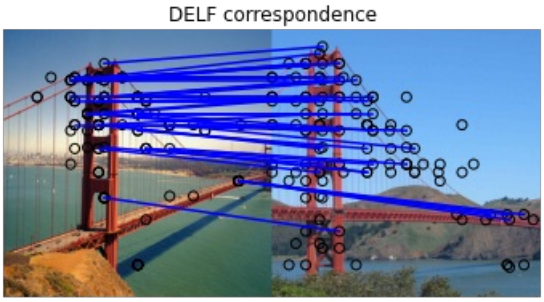
```
result1 = run_delf(image1)
result2 = run_delf(image2)
```

In [9]:

```
def match_images(image1,image2,result1,result2):
    distance_threshold = 0.8

    num_features_1 = result1['locations'].shape[0]
    print("Loaded image 1's %d features" % num_features_1)

    num_features_2 = result2['locations'].shape[0]
    print("Loaded image 2's %d features" % num_features_2)

    d1_tree = cKDTree(result1['descriptors'])
    _, indices = d1_tree.query(
      result2['descriptors'],
      distance_upper_bound = distance_threshold)

    locations_2_to_use = np.array([
      result2['locations'][i,]
      for i in range(num_features_2)
      if indices[i] != num_features_1
    ])

    locations_1_to_use = np.array([
        result1['locations'][indices[i],]
        for i in range(num_features_2)
        if indices[i] != num_features_1
    ])

    _, inliners = ransac(
     (locations_1_to_use ,  locations_2_to_use),
     AffineTransform,
     min_samples = 3,
     residual_threshold = 20,
     max_trials=1000)

    print('Found %d inliners' % sum(inliners))

    _,ax = plt.subplots()
    inliner_idxs = np.nonzero(inliners)[0]
    plot_matches(
    ax,
    image1,
    image2,
    locations_1_to_use,
    locations_2_to_use,
```

```
    np.column_stack((inliner_idxs, inliner_idxs)),
    matches_color = 'b')
    ax.axis('off')
    ax.set_title('DELF correspondence')


match_images(image1,image2,result1,result2)
```

```
Loaded image 1's 227 features
Loaded image 2's 202 features
Found 49 inliners
```



In [10]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def showme(frame,title=None):
    #window_title = "map"
    cv2.namedWindow(title, cv2.WINDOW_NORMAL) #open a window
    cv2.imshow(title, frame)                  #show the image in that widow
    cv2.waitKey(0)                            #wait for any key
    cv2.destroyAllWindows()                   #close everything

def showplt(image, title=None, pltnative=False, custSize=[100,10]):
    plt.figure(figsize = (custSize))
    if pltnative:
        plt.imshow(image)
    else:
        plt.imshow(image[...,::-1])
    plt.title(title)
    plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
    plt.show()

#
def warpImages(img1, img2, H):
    rows1, cols1 = img1.shape[:2]
    rows2, cols2 = img2.shape[:2]

    list_of_points_1 = np.float32([
        [0,0],
        [0,rows1],
        [cols1,rows1],
        [cols1,0]
    ])
    list_of_points_1 = list_of_points_1.reshape(-1,1,2)

    temp_points = np.float32([
        [0,0],
        [0,rows2],
        [cols2,rows2],
        [cols2,0]
    ])
    temp_points = temp_points.reshape(-1,1,2)

    list_of_points_2 = cv2.perspectiveTransform(temp_points, H)

    list_of_points = np.concatenate((list_of_points_1, list_of_points_2), axis=0)

    ##Define boundaries:
    [x min, y min] = np.int32(list of points.min(axis=0).ravel() - 0.5)
```

```
    [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)

    translation_dist = [-x_min,-y_min]

    H_translation = np.array([[1, 0, translation_dist[0]], [0, 1, translation_dist[1]], [0,0,1]])

    output_img = cv2.warpPerspective(img2,
                                     H_translation.dot(H),
                                     (x_max - x_min, y_max - y_min))
    ## Paste the image:
    output_img[translation_dist[1]:rows1+translation_dist[1],
               translation_dist[0]:cols1+translation_dist[0]] = img1

    return output_img
#
def warp(img1, img2, min_match_count = 10):
    sift = cv2.SIFT_create()

    # Extract the keypoints and descriptors
    keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(img2, None)

    # Initialize parameters for Flann based matcher
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks = 50)

    # Initialize the Flann based matcher object
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # Compute the matches
    matches = flann.knnMatch(descriptors1, descriptors2, k=2)

    # Store all the good matches as per Lowe's ratio test
    good_matches = []
    for m1,m2 in matches:
        if m1.distance < 0.7*m2.distance:
            good_matches.append(m1)

    if len(good_matches) > min_match_count:
        src_pts = np.float32([ keypoints1[good_match.queryIdx].pt
                               for good_match in good_matches ]).reshape(-1,1,2)

        dst_pts = np.float32([ keypoints2[good_match.trainIdx].pt
                               for good_match in good_matches ]).reshape(-1,1,2)

        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        result = warpImages(img2, img1, M)
        return result
        #cv2.imshow('Stitched output', result)
        #cv2.waitKey()
    else:
        print ("We don't have enough number of matches between the two images.")
        print ("Found only " + str(len(good_matches)) + " matches.")
        print ("We need at least " + str(min_match_count) + " matches.")
```

In [13]:

```
building1 = cv2.imread("../input/images/Building (2).jpg")
building2 = cv2.imread("../input/images/Building (3).jpg")
building3 = cv2.imread("../input/images/Building (4).jpg")
building4 = cv2.imread("../input/images/Building (1).jpg")
upBuilding1 = cv2.imread("../input/images/UP Building (1).jpg")
upBuilding2 = cv2.imread("../input/images/UP Building (2).jpg")

showplt(building1, title='building1', custSize=[10,2])
showplt(building2, title='building2', custSize=[10,2])
showplt(building3, title='building3', custSize=[10,2])
showplt(building4, title='building4', custSize=[10,2])
showplt(upBuilding1, title='upBuilding1', custSize=[10,2])
showplt(upBuilding2, title='upBuilding2', custSize=[10,2])
```

building2

building3

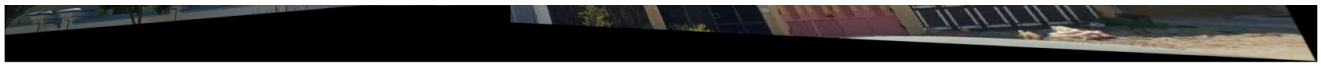
building4

upBuilding1

upBuilding2

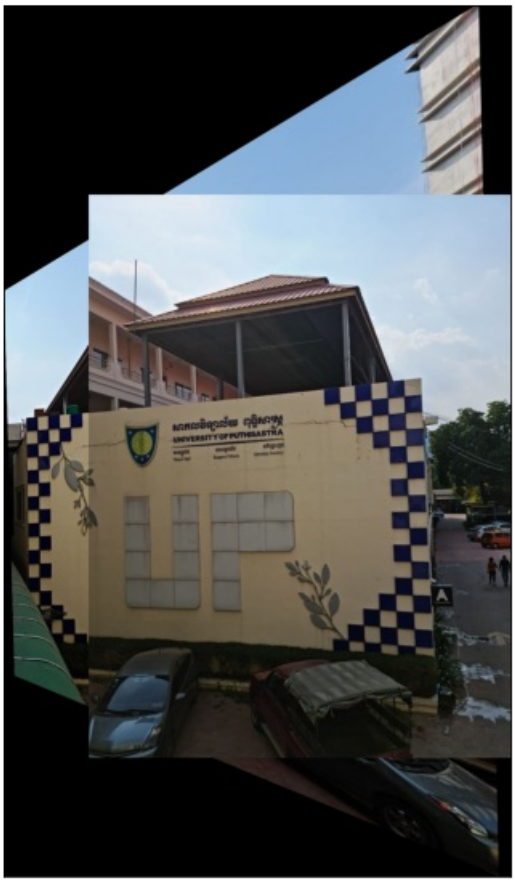
In [15]:

```
buildinCollage1 = warp(building2, building3)
buildinCollage2 = warp(building4, buildinCollage1)
showplt(buildinCollage2)
```



In [16]:

```
buildinCollage = warp(upBuilding2, upBuilding1)
showplt(buildinCollage)
```



In [ ]:

```

```