In [12]:

```
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 [17]:

```
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 [8]:

```
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 [19]:

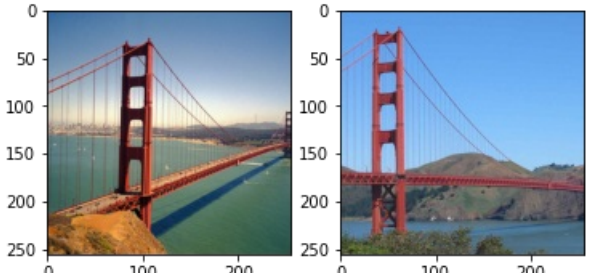
```
import matplotlib.pyplot as plt
```

In [20]:

```
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)
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x7fe3092bee50>
```



0 100 200 0 100 200

In [21]:

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

In [34]:

```
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 [35]:

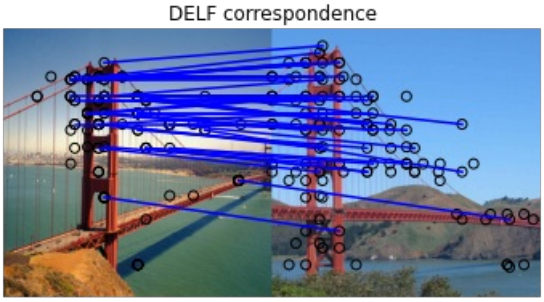
```
result1 = run_delf(image1)
result2 = run_delf(image2)
```

In [38]:

```
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 [ ]:

```
```