In [11]:
```
from absl import logging

import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageOps
from scipy.spatial import cKDTree
from skimage.feature import plot_matches
from skimage.measure import ransac
from skimage.transform import AffineTransform
from six import BytesIO

import tensorflow as tf

import tensorflow_hub as hub
from six.moves.urllib.request import urlopen
import cv2
import numpy as np
import matplotlib.pyplot as plt
import imageio
import imutils
cv2.ocl.setUseOpenCL(False)
feature_extractor = 'orb' # one of 'sift', 'surf', 'brisk', 'orb'
feature_matching = 'bf'
```

In [1]:
```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving IX-11-01917_0004_0004.JPG to IX-11-01917_0004_0004.JPG
Saving IX-11-01917_0004_0005.JPG to IX-11-01917_0004_0005.JPG
Saving IX-11-01917_0004_0006.JPG to IX-11-01917_0004_0006.JPG
```

In [4]:
```
image1 = ('IX-11-01917_0004_0004.JPG')
image2 = ('IX-11-01917_0004_0005.JPGg')
```

In [7]:
```
delf = hub.load('https://tfhub.dev/google/delf/1').signatures['default']
```

In [8]:
```
def run_delf(image):
  np_image = np.array(image)
  float_image = tf.image.convert_image_dtype(np_image, tf.float32)

  return delf(
      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 [12]:
```
Img1 = imageio.imread('IX-11-01917_0004_0005.JPG')
Img1_gray = cv2.cvtColor(Img1, cv2.COLOR_RGB2GRAY)

Img2 = imageio.imread('IX-11-01917_0004_0004.JPG')
# Opencv defines the color channel in the order BGR.
# Transform it to RGB to be compatible to matplotlib
Img2_gray = cv2.cvtColor(Img2, cv2.COLOR_RGB2GRAY)
```

In [13]:
```
def detectAndDescribe(image, method=None):
    """
    Compute key points and feature descriptors using an specific method
    """

    assert method is not None, "You need to define a feature detection method. Values are: 'sift', 'surf'"

    # detect and extract features from the image
    if method == 'sift':
        descriptor = cv2.xfeatures2d.SIFT_create()
    elif method == 'surf':
        descriptor = cv2.xfeatures2d.SURF_create()
    elif method == 'brisk':
        descriptor = cv2.BRISK_create()
    elif method == 'orb':
        descriptor = cv2.ORB_create()

    # get keypoints and descriptors
    (kps, features) = descriptor.detectAndCompute(image, None)

    return (kps, features)
```

In [14]:
```
kpsA, featuresA = detectAndDescribe(Img1_gray, method=feature_extractor)
```

```python
kpsB, featuresB = detectAndDescribe(Img2_gray, method=feature_extractor)
```

In [15]:
```python
def createMatcher(method,crossCheck):
    "Create and return a Matcher Object"

    if method == 'sift' or method == 'surf':
        bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=crossCheck)
    elif method == 'orb' or method == 'brisk':
        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=crossCheck)
    return bf
```

In [16]:
```python
def matchKeyPointsBF(featuresA, featuresB, method):
    bf = createMatcher(method, crossCheck=True)

    # Match descriptors.
    best_matches = bf.match(featuresA,featuresB)

    # Sort the features in order of distance.
    # The points with small distance (more similarity) are ordered first in the vector
    rawMatches = sorted(best_matches, key = lambda x:x.distance)
    print("Raw matches (Brute force):", len(rawMatches))
    return rawMatches
```

In [17]:
```python
def matchKeyPointsKNN(featuresA, featuresB, ratio, method):
    bf = createMatcher(method, crossCheck=False)
    # compute the raw matches and initialize the list of actual matches
    rawMatches = bf.knnMatch(featuresA, featuresB, 2)
    print("Raw matches (knn):", len(rawMatches))
    matches = []

    # loop over the raw matches
    for m,n in rawMatches:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if m.distance < n.distance * ratio:
            matches.append(m)
    return matches
```

In [18]:
```python
if feature_matching == 'bf':
    matches = matchKeyPointsBF(featuresA, featuresB, method=feature_extractor)
```

```python
    img3 = cv2.drawMatches(Img1,kpsA,Img2,kpsB,matches[:100],
                           None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
elif feature_matching == 'knn':
    matches = matchKeyPointsKNN(featuresA, featuresB, ratio=0.75, method=feature_extractor)
    img3 = cv2.drawMatches(Img1,kpsA,Img2,kpsB,np.random.choice(matches,100),
                           None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

```
Raw matches (Brute force): 197
```

In [19]:

```python
def getHomography(kpsA, kpsB, featuresA, featuresB, matches, reprojThresh):
    # convert the keypoints to numpy arrays
    kpsA = np.float32([kp.pt for kp in kpsA])
    kpsB = np.float32([kp.pt for kp in kpsB])

    if len(matches) > 4:

        # construct the two sets of points
        ptsA = np.float32([kpsA[m.queryIdx] for m in matches])
        ptsB = np.float32([kpsB[m.trainIdx] for m in matches])

        # estimate the homography between the sets of points
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
            reprojThresh)

        return (matches, H, status)
    else:
        return None
```

In [20]:

```python
M = getHomography(kpsA, kpsB, featuresA, featuresB, matches, reprojThresh=4)
if M is None:
    print("Error!")
(matches, H, status) = M
```

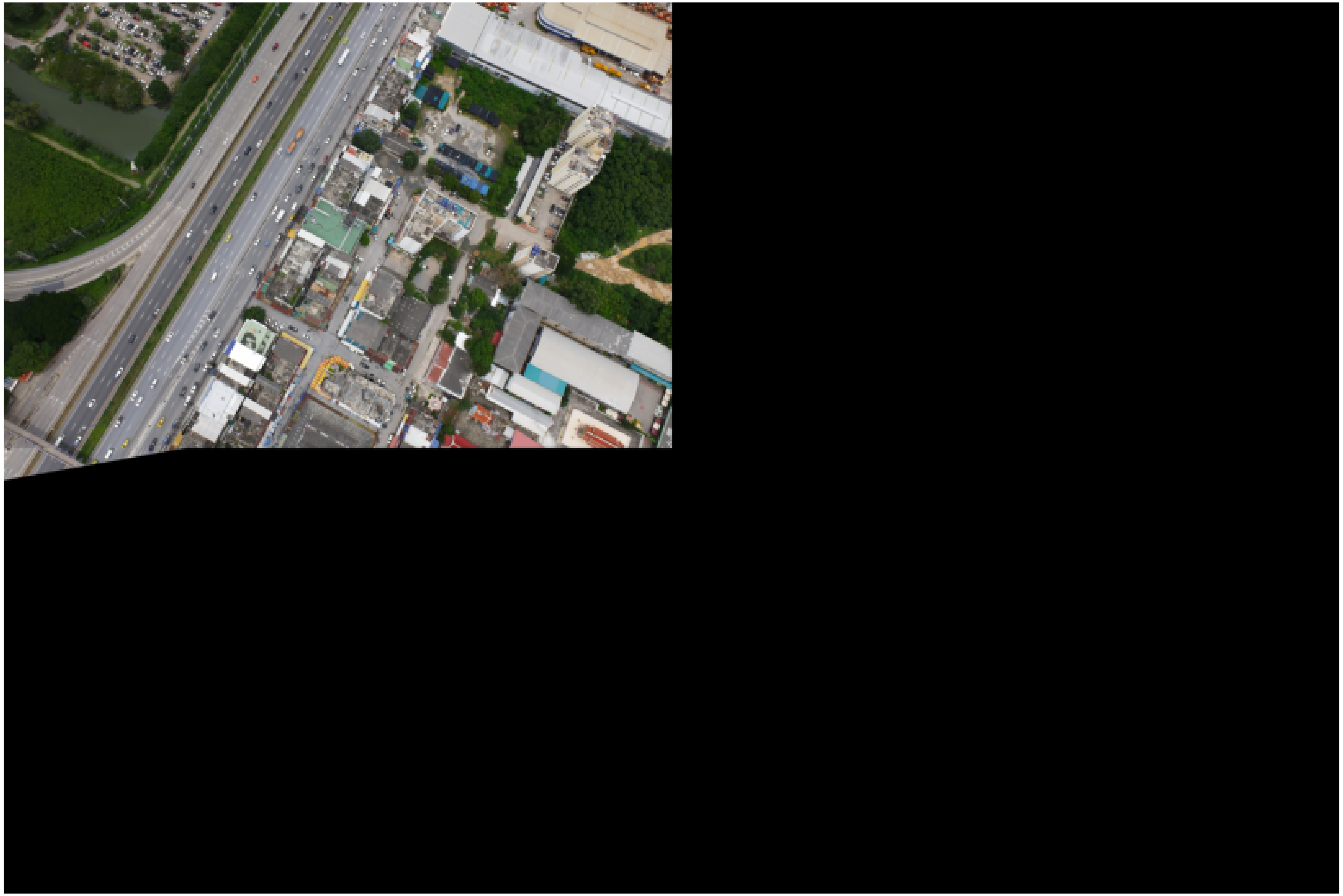
In [21]:

```python
# Apply panorama correction
width = Img1.shape[1] + Img2.shape[1]
height = Img2.shape[0] + Img2.shape[0]

result = cv2.warpPerspective(Img1, H, (width, height))
result[0:Img2.shape[0], 0:Img2.shape[1]] = Img2

plt.figure(figsize=(20,10))
plt.imshow(result)
```

```
plt.axis('off')
plt.show()
```



In [22]:
```
# transform the panorama image to grayscale and threshold it
gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
```

```python
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]

# Finds contours from the binary image
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# get the maximum contour area
c = max(cnts, key=cv2.contourArea)

# get a bbox from the contour area
(x, y, w, h) = cv2.boundingRect(c)

# crop the image to the bbox coordinates
result = result[y:y + h, x:x + w]

# show the cropped image
plt.figure(figsize=(20,10))
plt.imshow(result)
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x7f03c42f5a10>
```


0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
3000
4000
5000