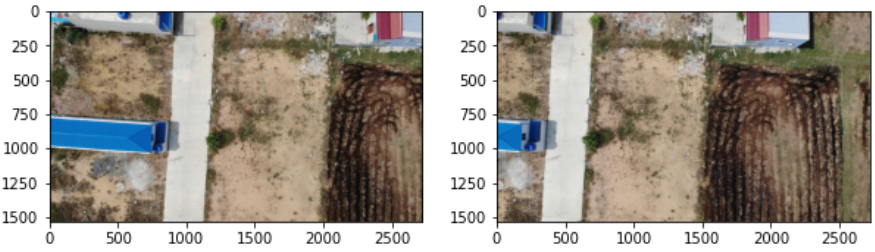
In [58]:
```
import numpy as np
import pandas as pd
from IPython.display import Image,display
from skimage import io
import matplotlib.pyplot as plt
```

In [59]:
```
import imutils
import cv2
import imageio
cv2.ocl.setUseOpenCL(False)
```

In [60]:
```
trainImg = imageio.imread('../input/image-stitching-from-drone-capture-opencv/drone/image_0041.jpg')
trainImg_gray = cv2.cvtColor(trainImg,cv2.COLOR_BGR2GRAY)

queryImg = imageio.imread('../input/image-stitching-from-drone-capture-opencv/drone/image_0061.jpg')
queryImg_gray = cv2.cvtColor(queryImg,cv2.COLOR_BGR2GRAY)

fig,(ax1,ax2) = plt.subplots(nrows=1,ncols=2, constrained_layout = False, figsize=(10,7))
ax1.imshow(queryImg,cmap='gray')

ax2.imshow(trainImg,cmap='gray')
plt.show()
```



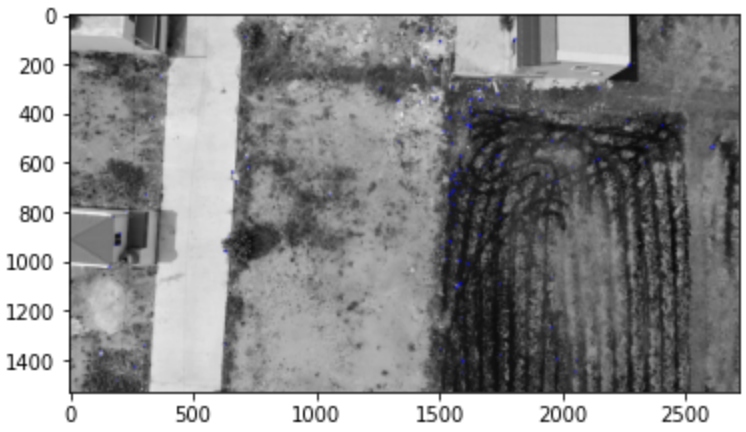
In [61]:
```
feature_extractor = 'orb'
feature_matching='bf'
```

In [62]:
```
def detectAndDescribe(image,method=None):
    assert method is not None

    if method == 'sift':
        descriptor = cv2.SIFT_create()
    elif method  == 'surf':
```

```
        descriptor = cv2.SURF_create()
    elif method == 'brisk':
        descriptor = cv2.BRISK_create()
    elif method == 'orb':
        descriptor = cv2.ORB_create()
    (kps,features) = descriptor.detectAndCompute(image,None)

    return (kps,features)
```

In [63]:
```
kpsA , featuresA = detectAndDescribe(trainImg_gray,  method=feature
_extractor)
kpsB, featuresB = detectAndDescribe(queryImg_gray, method = feature
_extractor)
```

In [64]:
```
plt.imshow(cv2.drawKeypoints(trainImg_gray,kpsA,None,(0,0,255)))
plt.show()
```



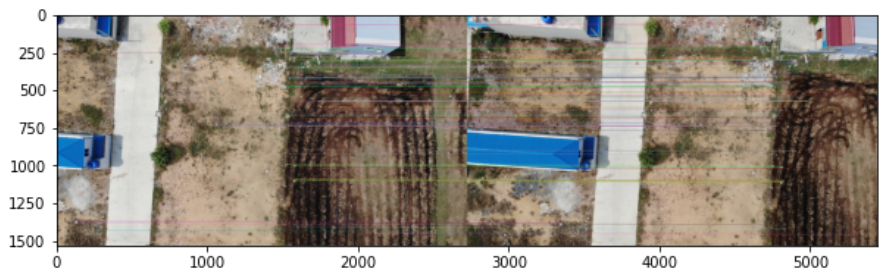
In [65]:
```
def createMatcher(method,crossCheck):
    if method=='sift'or method=='surf':
        bf = cv2.BFMatcher(cv2.NORM_L2,crossCheck=crossCheck)
    elif method == 'orb' or method=='brisk':
        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = crossChec
k)
    return bf
```

In [66]:
```
def MatchKeyPointsBF(featuresA,featuresB,method):
    bf  = createMatcher(method,crossCheck=True)

    best_matches = bf.match(featuresA,featuresB)

    rawMatches = sorted(best_matches, key = lambda x:x.distance)
    print('Raw matches (Brute Force):', len(rawMatches))
    return rawMatches
```

In [67]:
```
def matchKeyPointsKNN(featuresA,featuresB,ratio,method):
    bf = createMatcher(method,crossCheck=False)
    rawMatches = bf.knnMatch(featuresA,featuresB,2)
    print('Raw Matches(KNN):', len(rawMatches))
    matches = []

    for m,n in rawMatches:
        if m.distance < n.distance*ratio:
            matches.append(m)
    return matches
```

In [68]:
```
print('Using : {} feature matcher'.format(feature_matching))

fig = plt.figure(figsize=(10,7))

if feature_matching == 'bf':
    matches = MatchKeyPointsBF(featuresA,featuresB,method=feature_e
xtractor)
    img3 = cv2.drawMatches(trainImg,kpsA,queryImg,kpsB,matches[:100
],
                           None,flags = cv2.DrawMatchesFlags_NOT_DRA
W_SINGLE_POINTS)
elif feature_matching == 'knn':
    matches = matchKeyPointsKNN(featuresA,featuresB,ratio=0.75,meth
od=feature_extractor)
    img3 =  cv2.drawMatches(trainImg,kpsA,queryImg,kpsB,np.random.c
hoice(matches,100),
                            None,flags = cv2.DrawMatchesFlags_NOT_DR
AW_SINGLE_POINTS)

plt.imshow(img3)
plt.show()
```

```
Using : bf feature matcher
Raw matches (Brute Force): 287
```



In [69]:
```
def getHomography(kpsA,kpsB,featuresA,featuresB,matches,reprojThres
```

```
h):
    kpsA = np.float32([kp.pt for kp in kpsA])
    kpsB = np.float32([kp.pt for kp in kpsB])

    if len(matches) > 4:
        ptsA = np.float32([kpsA[m.queryIdx] for m in matches])
        ptsB = np.float32([kpsB[m.trainIdx] for m in matches])

        (H,status) = cv2.findHomography(ptsA,ptsB,cv2.RANSAC,reproj
Thresh)
        return (matches,H,status)
    else:
        return None
```

In [70]:
```
M = getHomography(kpsA,kpsB,featuresA,featuresB,matches,reprojThres
h=4)
if M is None:
    print('Error!')
(matches,H,status) = M
print(H)
```

```
[[ 1.30406116e+00  1.80267291e-02  2.53238367e+02]
 [ 4.14684463e-02  1.15118698e+00 -8.95723745e+01]
 [ 6.88435162e-05  7.53542949e-06  1.00000000e+00]]
```

In [71]:
```
width = trainImg.shape[1] + queryImg.shape[1]
height = trainImg.shape[0] + queryImg.shape[0]


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

plt.figure(figsize=(20,10))
plt.imshow(result)
plt.axis('off')
plt.show()
```