Assignment 1 (part II): Automatic Panorama Mosaicing

```
In [2]:  import numpy as np
         import matplotlib
         import matplotlib.image as image
         import matplotlib.pyplot as plt
         from skimage.feature import (corner_harris, corner_peaks, plot_matches, BRIEF, match_descriptors)
         from skimage.transform import warp, ProjectiveTransform
         from skimage.color import rgb2gray
         from skimage.measure import ransac
```

The cell below demonstrates feature detection (e.g. corners) and matching (e.g. BRIEF descriptor)

```
In [504…  imL = image.imread("images/CMU_left.jpg")
          imR = image.imread("images/CMU_right.jpg")
          imLgray = rgb2gray(imL)
          imRgray = rgb2gray(imR)


          # NOTE: corner_peaks and many other feature extraction functions return point coordinates as (y,x), that is (rows,cols)
          keypointsL = corner_peaks(corner_harris(imLgray), threshold_rel=0.0005, min_distance=5)
          keypointsR = corner_peaks(corner_harris(imRgray), threshold_rel=0.0005, min_distance=5)

          extractor = BRIEF()

          extractor.extract(imLgray, keypointsL)
          keypointsL = keypointsL[extractor.mask]
          descriptorsL = extractor.descriptors

          extractor.extract(imRgray, keypointsR)
          keypointsR = keypointsR[extractor.mask]
          descriptorsR = extractor.descriptors

          matchesLR = match_descriptors(descriptorsL, descriptorsR, cross_check=True)

          print ('the number of matches is {:2d}'.format(matchesLR.shape[0]))

          fig = plt.figure(1,figsize = (12, 4))
          axA = plt.subplot(111)
          plt.gray()
          plot_matches(axA, imL, imR, keypointsL, keypointsR, matchesLR) #, matches_color = 'r')
          axA.axis('off')

          plt.show()
```

```
the number of matches is 56
```

C:\Users\韩佳芮\AppData\Local\Temp\ipykernel_42356\1034223736.py:28: FutureWarning: `plot_matches` is deprecated since version 0.23 and will be removed in version 0.25. Use `skimage.feature.plot_matched_features` instead.
  plot_matches(axA, imL, imR, keypointsL, keypointsR, matchesLR) #, matches_color = 'r')

Problem 1

Rederive your formula in Problem 4a from Part I of the assignment for the following modification. Assume there are $N = 53$ matches (p, p') as in figure 1 above. $N_i = 21$ of these matches are inliers for a homography, while the rest of the matches are $N_o = 32$ outliers. To estimate a homography you need a sample with $K = 4$ matches. What is the least number of times one should randomly sample a subset of K matches to get probability $p \geq 0.95$ that at least one of these samples has all of its K matches from

inliers? Derive a general formula and compute a numerical answer for the specified numbers.

Solution:

P(a sample of 4 matches are all inliers) = $\frac{21}{53} \cdot \frac{20}{52} \cdot \frac{19}{51} \cdot \frac{18}{50} \approx 0.02$

P(a sample of 4 matches are not all inliers) = 1 - P(a sample of 4 matches are all inliers) = $1 - 0.02 = 0.98$

P (at least one of the subset of K matches are all inliers) = 1 - P (none of the subset of K matches are all inliers) = $1 - (0.98)^n \geq 0.95$

$$\Rightarrow n \geq 149$$

Problem 2: (RANSAC for Homographies)

Write code below using RANSAC to estimate Homography from matched pairs of points above. This cell should display new figure 2 similar to figure 1 above, but it should show only inlier pairs for the detected homography. HINT: you can use $ProjectiveTransform$ from library $skimage$ declared at the top of the notebok.

In [289...
```python
import random

matchLen = matchesLR.shape[0]
max = 0
maxSet = []
X, Y = [], []
for i in range(len(matchesLR)):
    l = matchesLR[i][0]
    X.append(keypointsL[l])
    r = matchesLR[i][1]
    Y.append(keypointsR[r])
X = np.array(X)
Y = np.array(Y)

for j in range(149):
    model_robust, inliers = ransac((X, Y), ProjectiveTransform, min_samples=3, residual_threshold=2, max_trials=10)

    count = 0
    for i in inliers:
        if i == True:
            count = count+1
    if count > max:
        maxSet = inliers
        max = count
inlierA = []
inlierB = []
for i in range(len(maxSet)):
    if maxSet[i] == True:
        inlierA.append(X[i])
        inlierB.append(Y[i])
inlierA, inlierB = np.array(inlierA), np.array(inlierB)
matches = []
for i in range(len(inlierA)):
    matches.append([i, i])
matches = np.array(matches)

fig = plt.figure(2,figsize = (12, 4))
axA = plt.subplot(111)
plt.gray()
plot_matches(axA, imL, imR, inlierA, inlierB, matches) #, matches_color = 'r')
axA.axis('off')

plt.show()
```
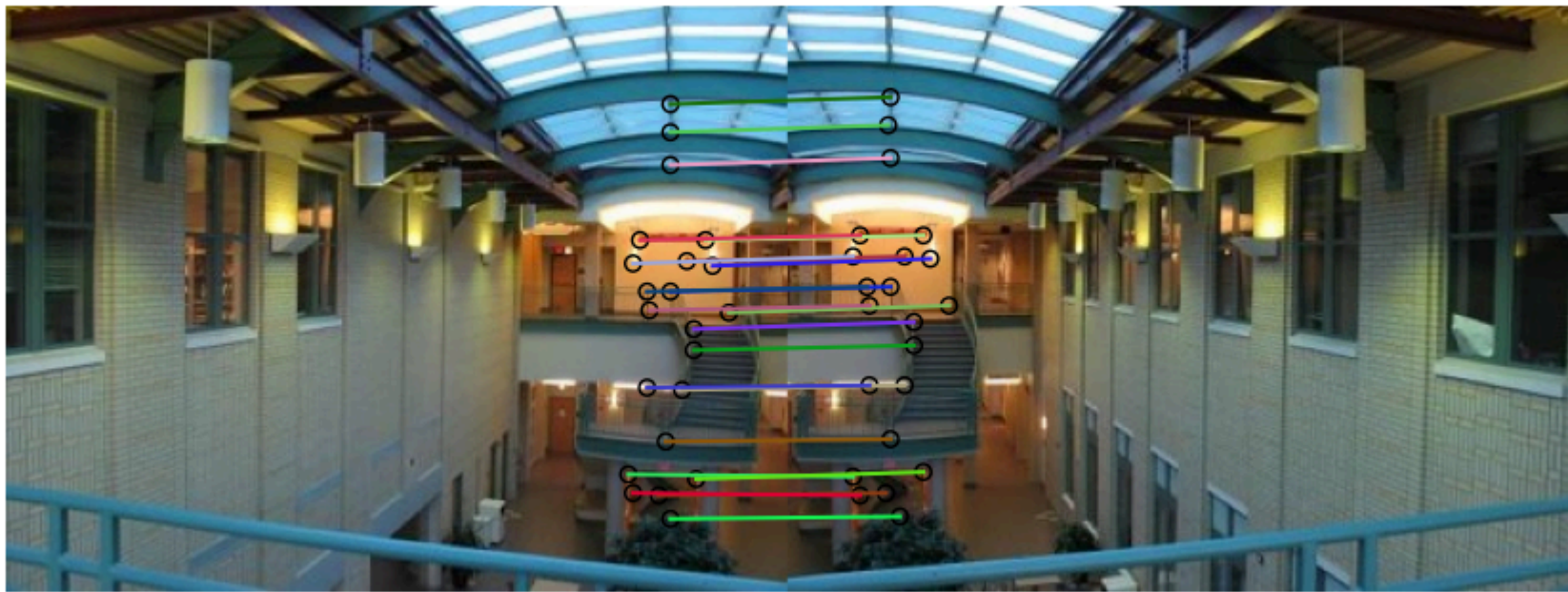
```
C:\Users\韩佳芮\AppData\Local\Temp\ipykernel_42356\3781514337.py:41: FutureWarning: `plot_matches` is deprecated since version
0.23 and will be removed in version 0.25. Use `skimage.feature.plot_matched_features` instead.
  plot_matches(axA, imL, imR, inlierA, inlierB, matches) #, matches_color = 'r')
```

Problem 3 (reprojecting onto common PP)

Use common PP corresponding to the plane of the left image. Your pamorama mosaic should be build inside a "reference frame" (think about it as an empty canvas of certain size) inside this common PP. The reference frame should be big enough to contain the left image and the part of the view covered by the right image after reprojecting onto common PP. Create a new figure 3 including the following three images (spread them vertically). First, show your reference frame only with the left image inside. Second, show the reference frame containing only a reprojected right image (warp it using a homography computed in Problem 1). Third, for comparison, show the reference frame containing only the right image reprojected using a (bad) homography estimated from all matches (including outliers, as in figure 1).

HINT1: use function $warp$ from library $skimage$ declared at the top of the notebook.

HINT2: function $warp$ needs "inverse map" as a (second) argument as it uses "inverse warping" to compute output intensities

```
In [451... np.dot(2, [[1], [2], [3]])

Out[451... array([[2],
               [4],
               [6]])
```

```
In [443... X[:, 0], X[:, 1] = X[:, 1], X[:, 0].copy()
         Y[:, 0], Y[:, 1] = Y[:, 1], Y[:, 0].copy()

         inlierA[:, 0], inlierA[:, 1] = inlierA[:, 1], inlierA[:, 0].copy()
         inlierB[:, 0], inlierB[:, 1] = inlierB[:, 1], inlierB[:, 0].copy()
```

```
In [468... Idv = np.identity(3)
         fig = plt.figure(3,figsize = (12, 14))

         plt.subplot(311)
         outL = warp(imL, Idv, output_shape=(300, 650))
         plt.imshow(outL)
         plt.title("reference frame with the left image")

         ##########################
         plt.subplot(312)

         plt.title("reference frame with the right image (reprojected)")
         tform = ProjectiveTransform()

         tform.estimate(inlierA, inlierB)

         outR = warp(imR, tform, output_shape=(300, 650))
         plt.imshow(outR)
         ##########################
         plt.subplot(313)
         tform1 = ProjectiveTransform()

         tform1.estimate(X, Y)
         outRBad = warp(imR, tform1, output_shape=(300, 650))
         plt.imshow(outRBad)
         plt.title("reference frame with the right image (reprojected badly)")

         plt.show()
```
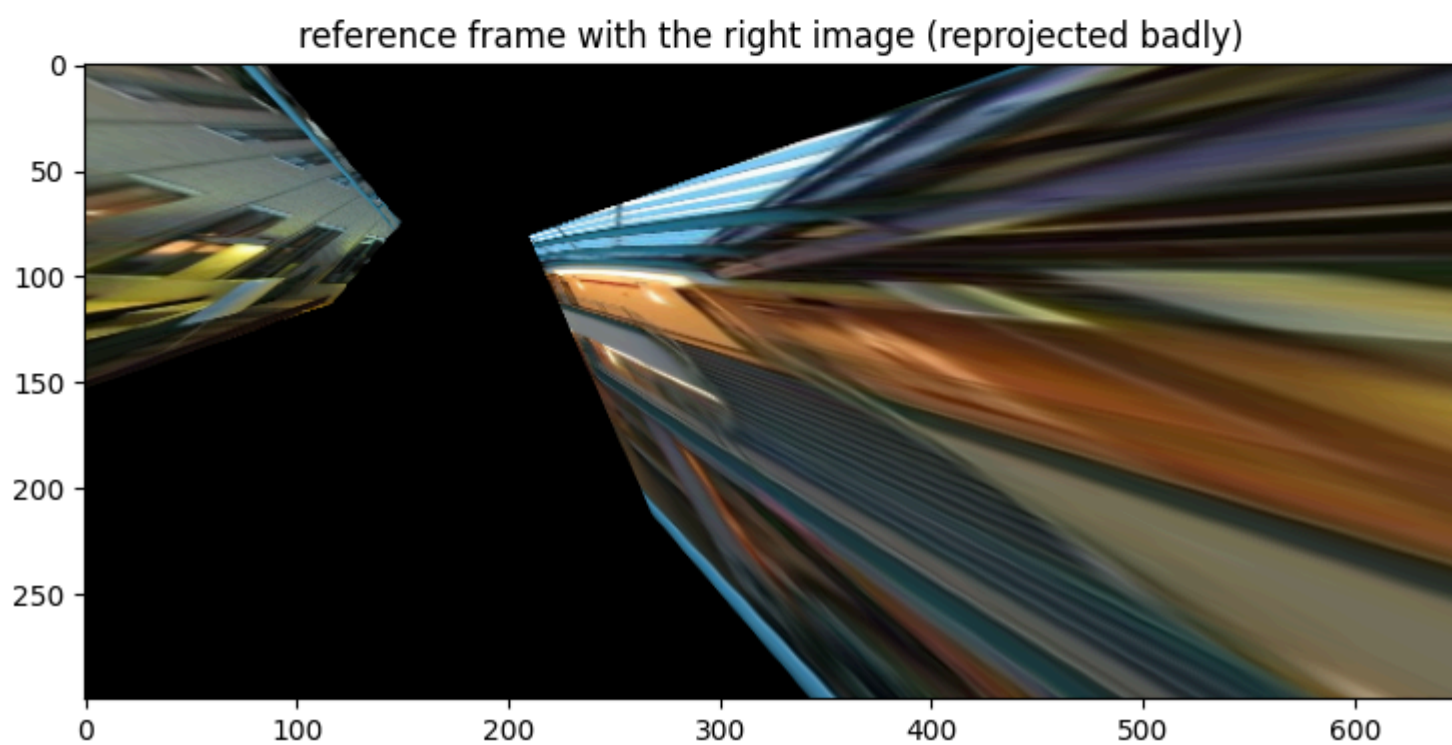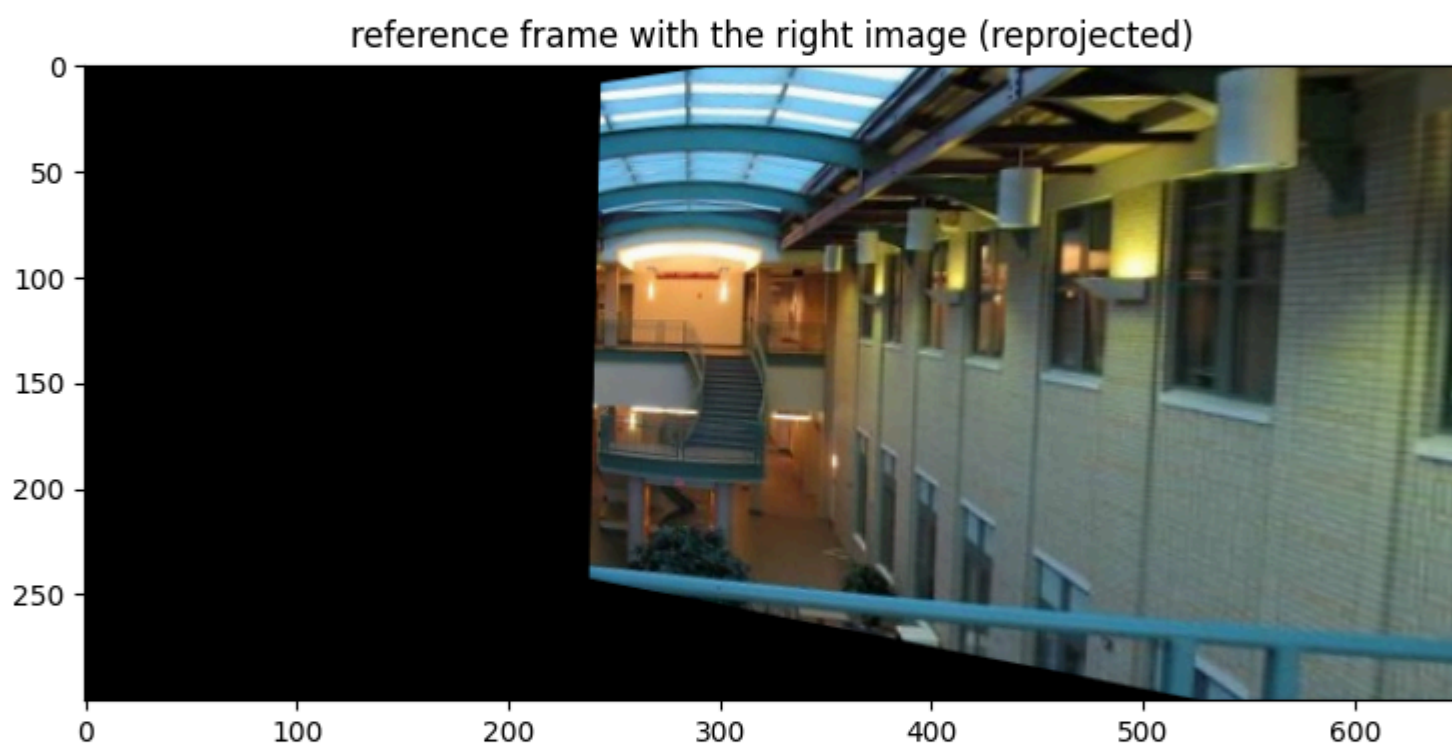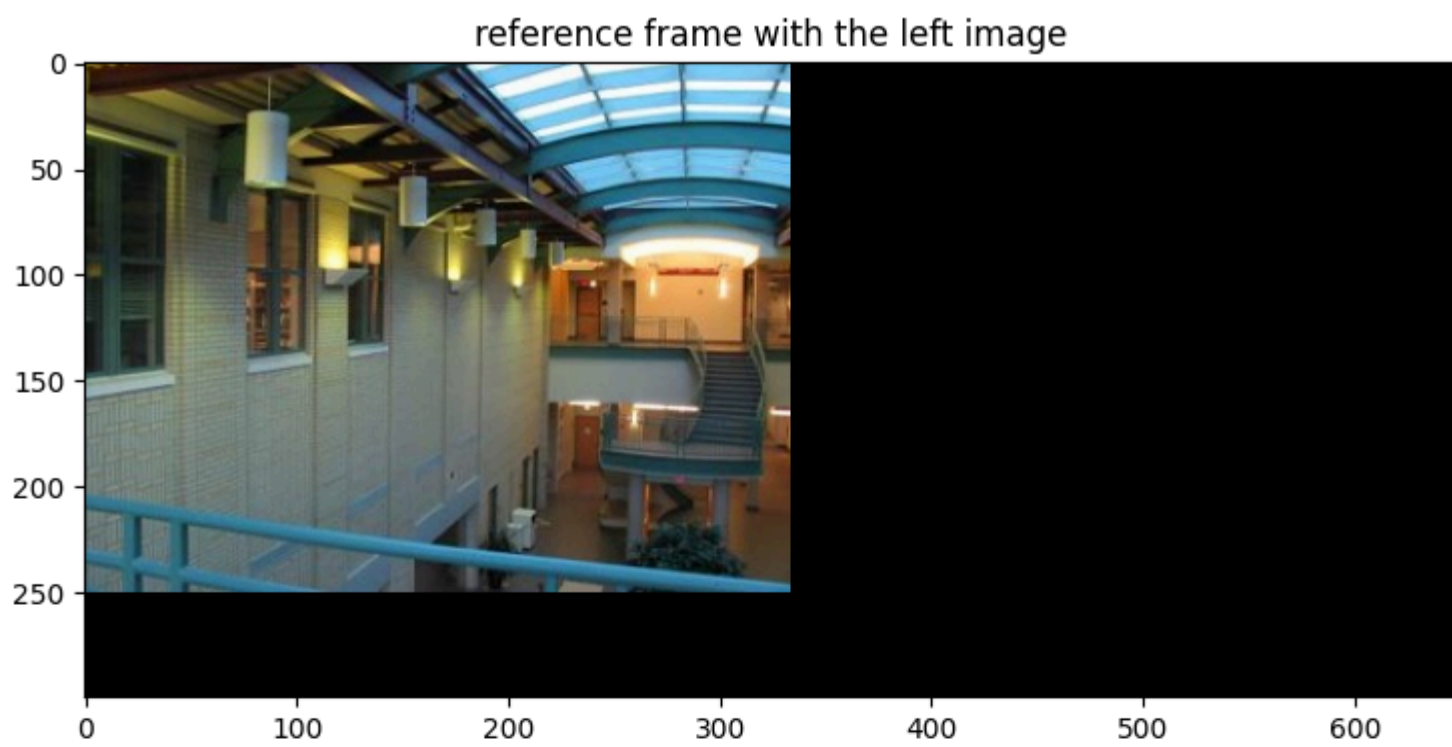
reference frame with the left image

reference frame with the right image (reprojected)

reference frame with the right image (reprojected badly)

Problem 4 (blending)

(part a) Write code for a function below computing distance transform for the boundary of a given image. It returns a numpy array of the same size as the image with distances from each pixel to the closest point on the boundary of the image (float values).

```python
def boundaryDT(image):
    imageDist = np.array([[0 for i in range(image.shape[1])] for j in range(image.shape[0])])
    height, width = image.shape[0], image.shape[1]
    maxIter = int(min(height, width) /2)
    for i in range(1, maxIter+1):
        imageDist[i, i:-i].fill(i)
        imageDist[-i-1, i:-i].fill(i)
        imageDist[i:-i, i].fill(i)
        imageDist[i:-i, -i-1].fill(i)
    return imageDist
```

(part b) Use function from part (a) to compute a distance transform for both images. Create a new figure 4 showing the following two images. First, show reference frame containing only the left image's boundaryDT instead of the left image. Second, show reference frame containing only the right image's boundaryDT reprojected instead of the right image.
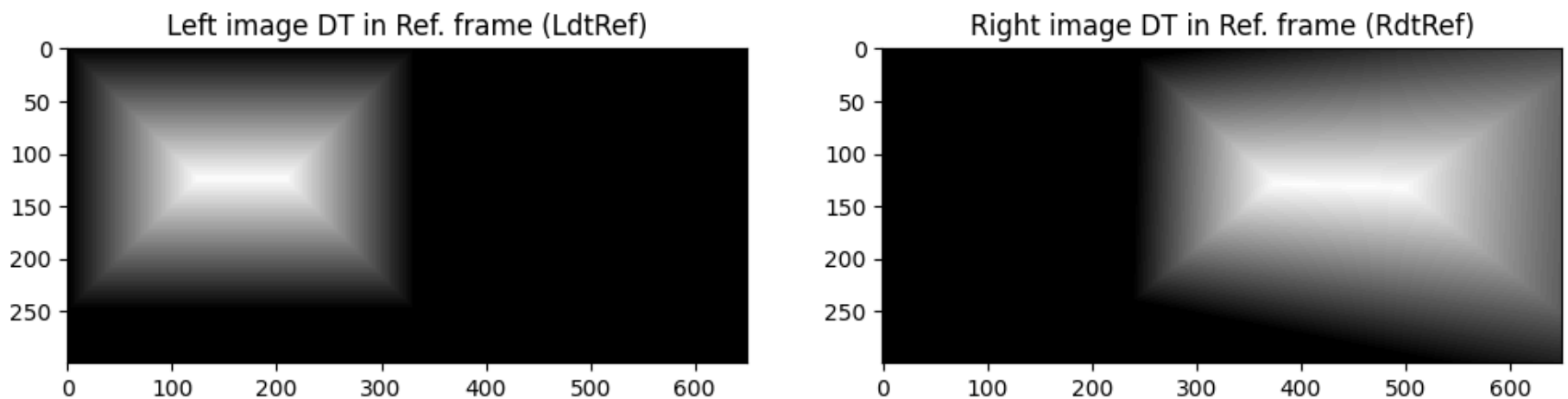
In [505...
```python
fig = plt.figure(4,figsize = (12, 3))
plt.subplot(121)
outLDT = warp(boundaryDT(imL), Idv, output_shape=(300, 650))
plt.imshow(outLDT)
plt.title("Left image DT in Ref. frame (LdtRef)")

plt.subplot(122)
outRDT = warp(boundaryDT(imR), tform, output_shape=(300, 650))

plt.imshow(outRDT)
plt.title("Right image DT in Ref. frame (RdtRef)")
```

Out[505... Text(0.5, 1.0, 'Right image DT in Ref. frame (RdtRef)')

(part c) Use boundary distance transforms to blend left and right images (reprojected) into the reference frame. Create a new figure 5 showing the following three images. The first and second images should be smooth $alpha$'s suitable for blending the left and right images (e.g. based on distance transforms as discussed in class). The third image should be your panorama: left and (reprojected) right images blended inside the reference frame. Your panorama should also show (reprojected) features - homography inliers - from both left and right images. Use different colors/shapes to distinct features from the left and the right images.

In [503...
```python
w1 = warp(boundaryDT(imL), Idv, output_shape=(300, 650))
w2 = warp(boundaryDT(imR), tform, output_shape=(300, 650))
sum = w1+w2 +0.0000000000000000001   # avoid divide by zero error
fig = plt.figure(5,figsize = (12, 14))
plt.subplot(311)
imgL = warp(np.divide(w1, sum), Idv, output_shape=(300, 650))
plt.imshow(imgL)
plt.title("alpha based on DT (for RransacRef)")

plt.subplot(312)
imgR = warp(np.divide(w2, sum), Idv, output_shape=(300, 650))
plt.imshow(imgR)
plt.title("alpha based on DT (for RransacRef)")


plt.subplot(313)
outL[:,:,0] = np.multiply(outL[:,:,0], imgL) + np.multiply(outR[:,:,0], imgR)
outL[:,:,1] = np.multiply(outL[:,:,1], imgL) + np.multiply(outR[:,:,1], imgR)
outL[:,:,2] = np.multiply(outL[:,:,2], imgL) + np.multiply(outR[:,:,2], imgR)

plt.imshow(outL)
inlierBnew = warp(inlierB, tform)
plt.scatter(inlierA[:, 0], inlierA[:, 1], s=10, c='red', marker='o')

inlierBX = inlierB[:, 0]
inlierBY = inlierB[:, 1]
```

```
tform = ProjectiveTransform()
tform.estimate(inlierA, inlierB)
tmat = np.linalg.inv(tform.params)
col1 = np.divide(np.dot(tmat[0][0], inlierBX) + np.dot(tmat[0][1], inlierBY) + tmat[0][2], np.dot(tmat[2][0], inlierBX) + np.d
col2 = np.divide(np.dot(tmat[1][0], inlierBX) + np.dot(tmat[1][1], inlierBY) + tmat[1][2], np.dot(tmat[2][0], inlierBX) + np.d
plt.scatter(col1, col2, s=10, c='blue', marker='o')

plt.title("Panorama")

plt.show()
```


alpha based on DT (for RransacRef)

alpha based on DT (for RransacRef)

Panorama