Assignment 1 (part II): Automatic Panorama Mosaicing

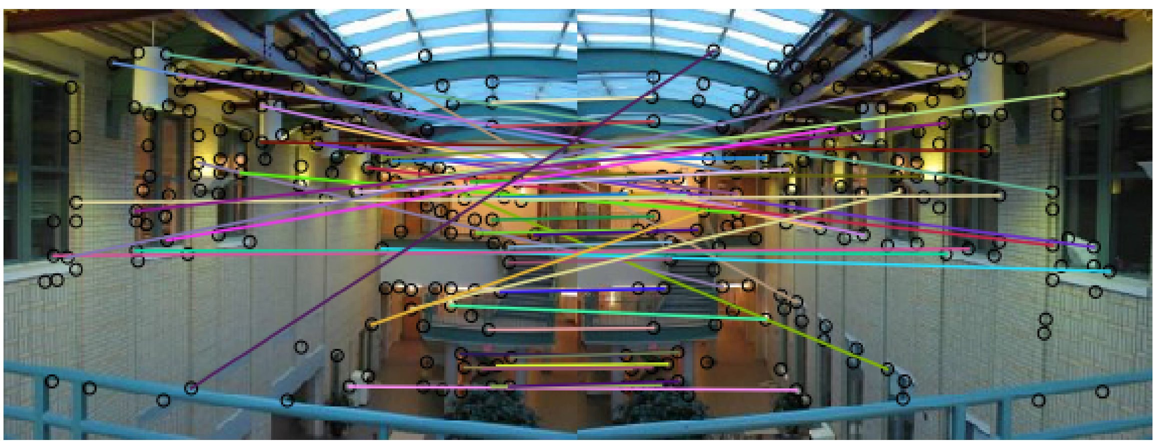
In [1]:
```
%matplotlib inline

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_matche
s, BRIEF, match_descriptors)
from skimage.transform import warp, ProjectiveTransform
from skimage.color import rgb2gray
from skimage.measure import ransac
import math
```

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

In [2]:

```
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 retu
rn point coordinates as (y,x), that is (rows,cols)
keypointsL = corner_peaks(corner_harris(imLgray), threshold_rel=0.000
5, min_distance=5)
keypointsR = corner_peaks(corner_harris(imRgray), threshold_rel=0.000
5, 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) #, mat
ches_color = 'r')
axA.axis('off')
plt.show()
```

```
the number of matches is 53
```



Problem 1

Rederive your formula in Problem 3a 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.

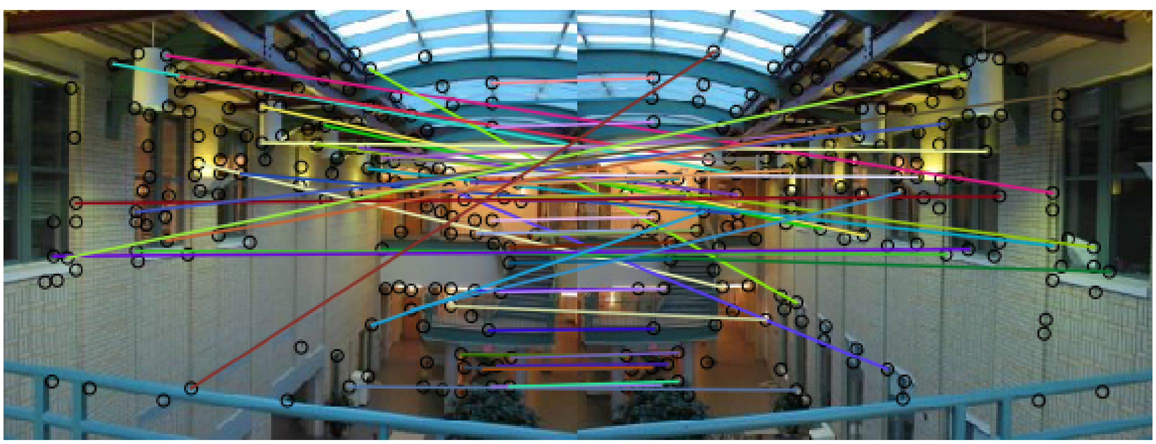
In [3]:
```
def iterations(p, s, N_i, N_o):
    p_i = N_i / (N_i + N_o)
    return (math.log(1 - p)) / (math.log(1 - p_i ** s))

N = iterations(0.95, 4, 21, 32)
print('N =', N)
print('Number of trials:', math.ceil(N))
```

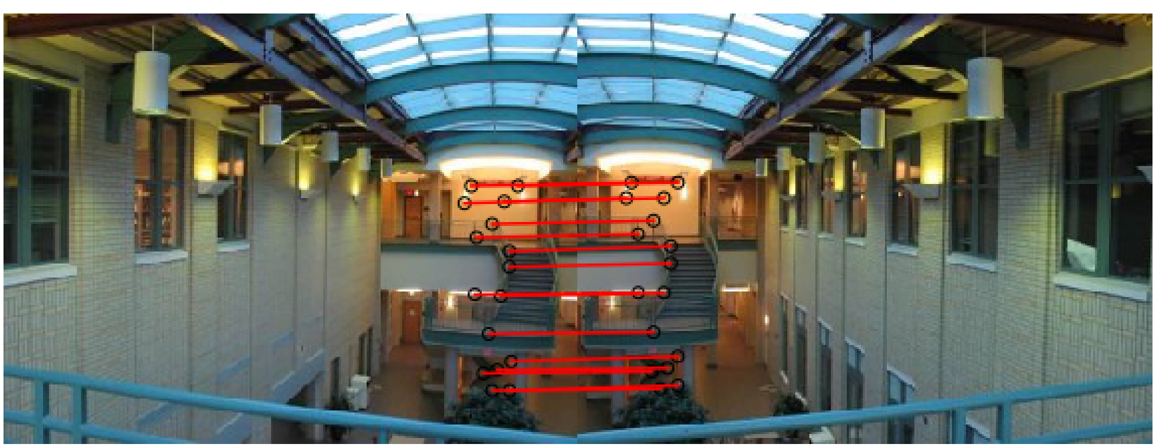
```
N = 120.03872066964435
Number of trials: 121
```

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 [4]:
```
keyMatchesL = keypointsL[matchesLR[:,0]]
keyMatchesR = keypointsR[matchesLR[:,1]]

model_robust, inliers = ransac((keyMatchesL, keyMatchesR),
                               ProjectiveTransform, min_samples = 4, residua
l_threshold = 2.5, max_trials = math.ceil(N))

matches = []
for i in range (len(keyMatchesL[inliers])):
    matches.append([i, i])
matches = np.reshape(matches, (len(matches), 2))

fig = plt.figure(2,figsize = (12, 4))
axA = plt.subplot(111)
plt.gray()
plot_matches(axA, imL, imR, keypointsL, keypointsR, matchesLR) #, mat
ches_color = 'r')
axA.axis('off')
plt.show()

fig = plt.figure(2,figsize = (12, 4))
axA = plt.subplot(111)
plt.gray()
plot_matches(axA, imL, imR, keyMatchesL[inliers], keyMatchesR[inliers
],
             matches, matches_color = 'r')
axA.axis('off')
plt.show()
```





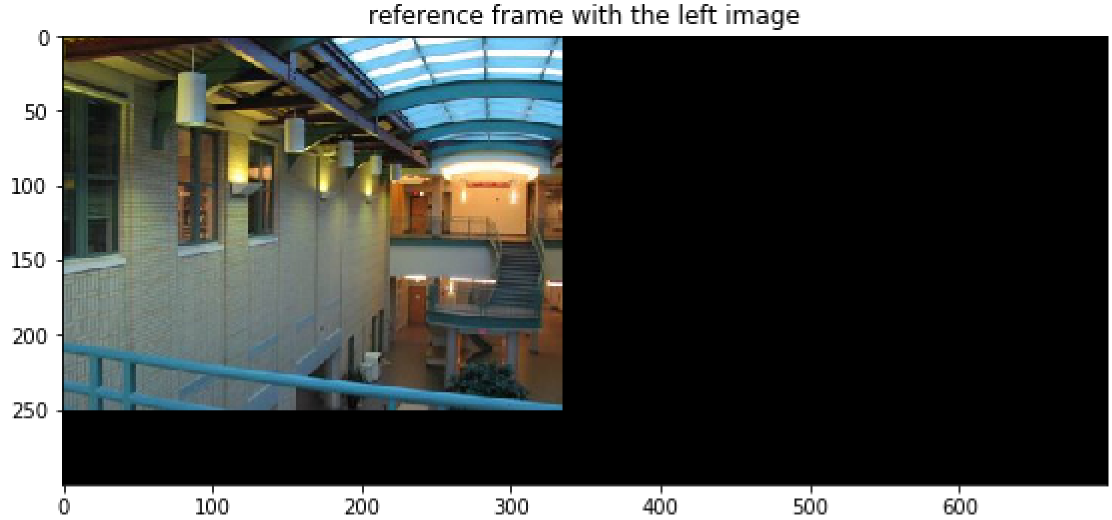
Problem 3 (reprojecting onto common PP)

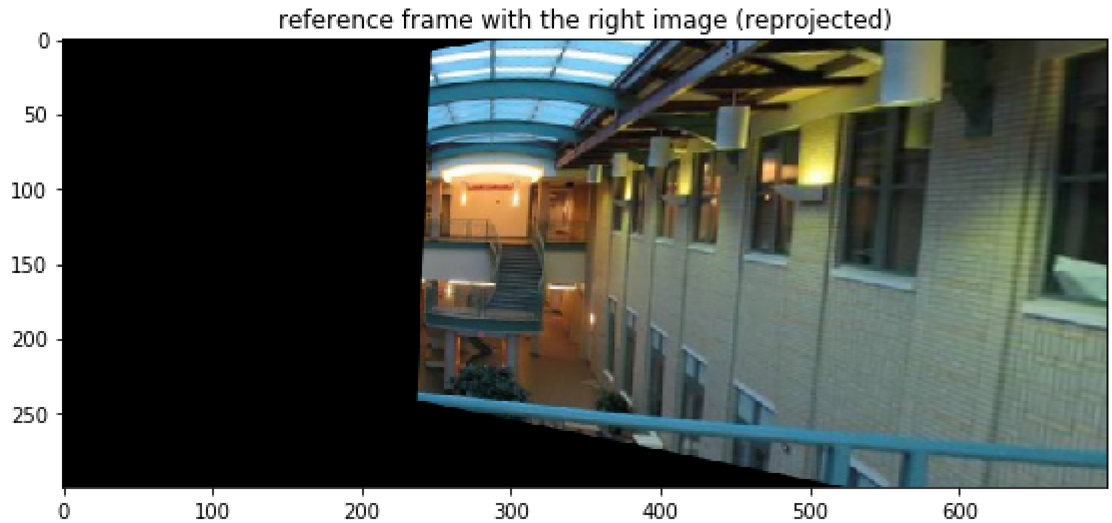
Use common PP corresponding to the plane of the left image. Your pamorama mosaic should be build inside a "reference frame" (think about it as an empty canvas of certain size) inside this common PP. The reference frame should be big enough to contain the left image and the part of the view covered by 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).

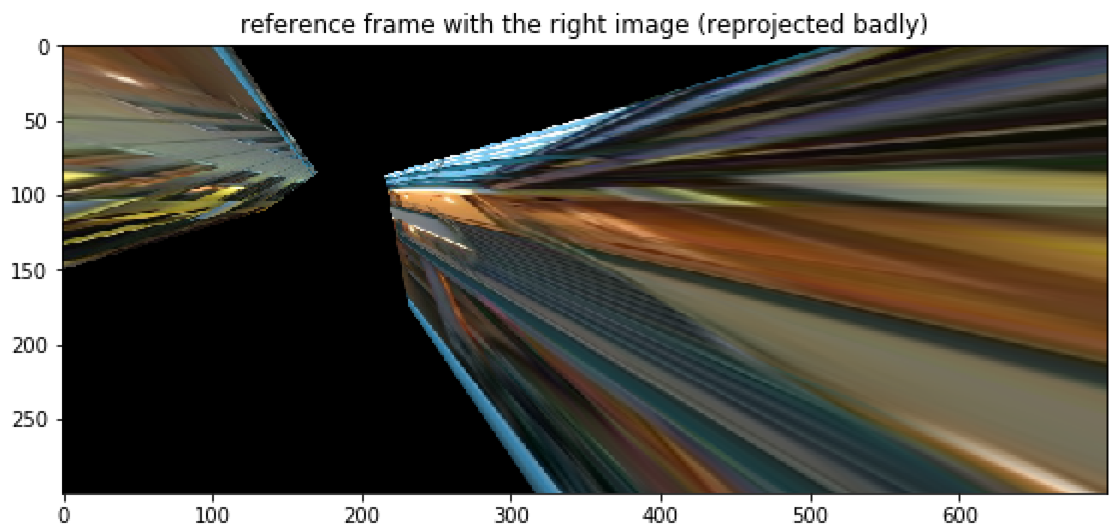
HINT1: use function $warp$ from library $skimage$ declared at the top of the notebook.

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

In [5]:
```
imageL = np.zeros([300, 700, 3])
imageL[: len(imL), : len(imL[0])] = imL/255

imageT = warp(imR.transpose((1, 0, 2)), model_robust, output_shape =
(700, 300))
imageR = imageT.transpose((1, 0, 2))

model_robustBad, inliersBad = ransac((keyMatchesL, keyMatchesR),
                                    ProjectiveTransform, min_samples = 4, res
idual_threshold = 5, max_trials = 10)

modelBadPTrans = ProjectiveTransform()
modelBadPTrans.estimate(keyMatchesL, keyMatchesR)
imageBadT = warp(imR.transpose((1, 0, 2)), modelBadPTrans, output_sha
pe = (700, 300))
imageBad = imageBadT.transpose((1, 0, 2))

fig = plt.figure(3,figsize = (12, 14))
plt.subplot(311)
plt.imshow(imageL)
plt.title("reference frame with the left image")

plt.subplot(312)
plt.imshow(imageR)
plt.title("reference frame with the right image (reprojected)")

plt.subplot(313)
plt.imshow(imageBad)
plt.title("reference frame with the right image (reprojected badly)")

plt.show()
```







Problem 4 (blending)

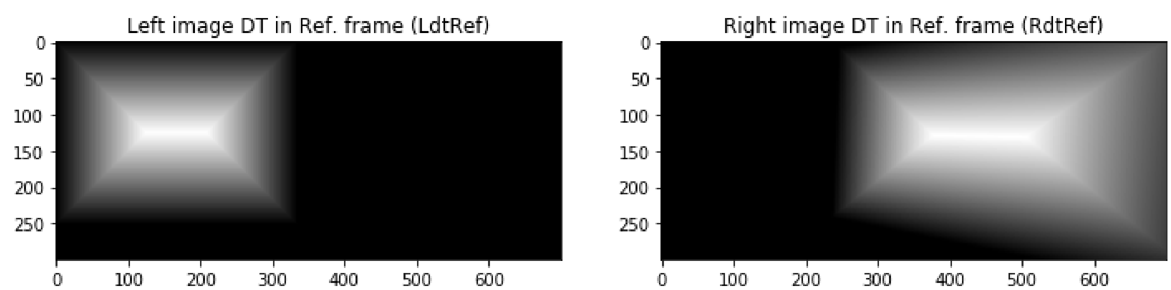
(part a) Write code for a function below computing distance transform for the boundary of a given image. It returns a numpy array of the same size as the image with distances from each pixel to the closest point on the boundary of the image (float values).

In [6]:
```
def boundaryDT(image):
    height = len(image) - 1
    width = len(image[0]) - 1
    distances = np.zeros((len(image), len(image[0])))
    for i in range(len(image)):
        for j in range(len(image[0])):
            distances[i][j] = float(min(i, height - i, j, width - j)
+ 0.5)
    return distances
```

(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.

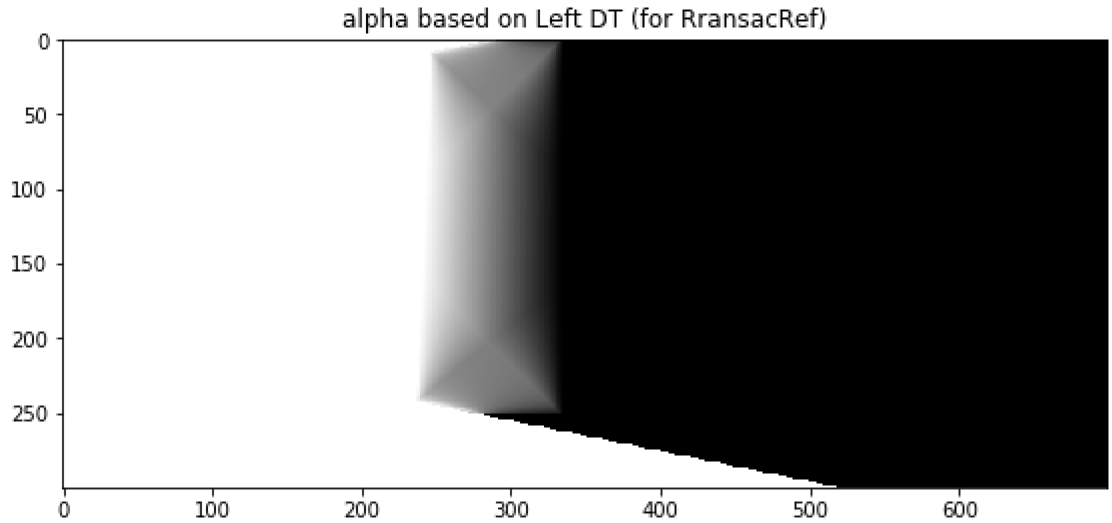
In [7]:
```
l = np.zeros([300, 700])
leftDT = boundaryDT(imL)
l[: len(leftDT), : len(leftDT[0])] = leftDT

rightDT = boundaryDT(imR)
r = warp(rightDT.transpose(), model_robust, output_shape = (700, 300
))
r = r.transpose()

fig = plt.figure(4,figsize = (12, 3))
plt.subplot(121)
plt.imshow(l)
plt.title("Left image DT in Ref. frame (LdtRef)")

fig = plt.figure(4,figsize = (12, 3))
plt.subplot(122)
plt.imshow(r)
plt.title("Right image DT in Ref. frame (RdtRef)")
```

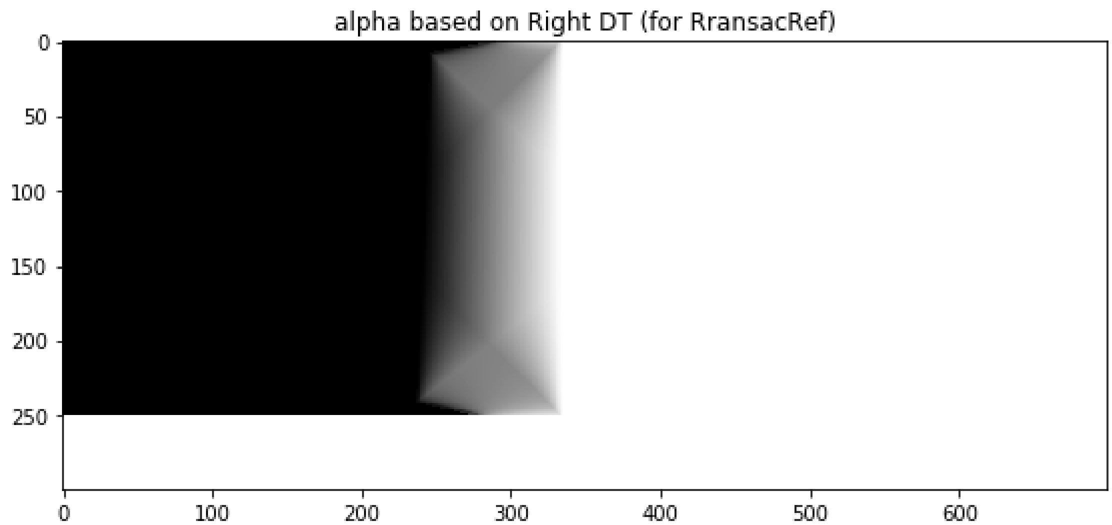
Out[7]:
```
Text(0.5, 1.0, 'Right image DT in Ref. frame (RdtRef)')
```

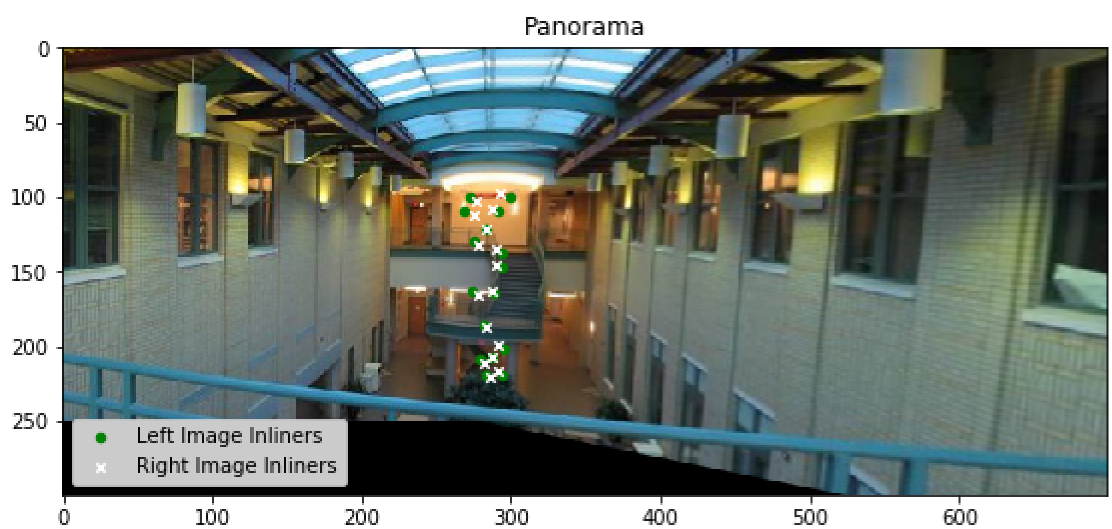


(part c) Use boundary distance transforms to blend left and right images (reprojected) into the reference frame. Create a new figure 5 showing the following three images. First and second should be $alpha$'s for blending the left and right images. These alphas should be based on distance transforms as discussed in class. Third, 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 [10]:
```
np.seterr(divide='ignore', invalid='ignore')

alphaL = l / (l + r)
alphaR = r / (l + r)

alphaLShape = alphaL.reshape((len(alphaL), len(alphaL[0]), 1))
alphaRShape = alphaR.reshape((len(alphaR), len(alphaR[0]), 1))
image = alphaLShape * imageL + alphaRShape * imageR

#r' = Hr -> r' are the points in the right image. H is the homograph
y. r is the points on the actual image
#we know r' and H. need to find r, which is r = H^-1r'
#since H is a 3x3 matrix, adding a colum 1 to the inliers to make mul
tiplication possible
rightInliers = np.ones((keyMatchesR[inliers].shape[0], 3))
rightInliers[:,:2] = keyMatchesR[inliers]
rightPts = np.dot(rightInliers, np.linalg.inv(model_robust.params).T)

fig = plt.figure(5,figsize = (12, 14))
plt.subplot(311)
plt.imshow(alphaL)
plt.title("alpha based on Left DT (for RransacRef)")

fig = plt.figure(5,figsize = (12, 14))
plt.subplot(312)
plt.imshow(alphaR)
plt.title("alpha based on Right DT (for RransacRef)")

fig = plt.figure(5,figsize = (12, 14))
plt.subplot(313)
plt.imshow(image)
plt.title("Panorama")
plt.scatter(keyMatchesL[inliers][:,1], keyMatchesL[inliers][:,0], s =
20, c = 'green', marker = 'o', label = 'Left Image Inliners')
plt.scatter(rightPts[:,1], rightPts[:,0], s = 20, c = 'white', marker
= 'x', label = 'Right Image Inliners')
plt.legend(loc = 'lower left')
plt.show()
```


alpha based on Left DT (for RransacRef)
0
50
100
150
200
250
0
100
200
300
400
500
600


alpha based on Right DT (for RransacRef)
0
50
100
150
200
250
0
100
200
300
400
500
600


Panorama
Left Image Inliners
Right Image Inliners
0
50
100
150
200
250
0
100
200
300
400
500
600

In [ ]: