Fast Linear Approximate Nearest Neighbour feature detection

Two main types of star tracking:

I. Relative rotation and rate measurement
II. The lost-in-space problem

We've chosen the lost-in-space problem as it's more interesting and more difficult.

Okay, so to find stars in an image is an easy process, we can take a picture and produce some threshold values, such that anything darker is space and anything lighter is probably a star.

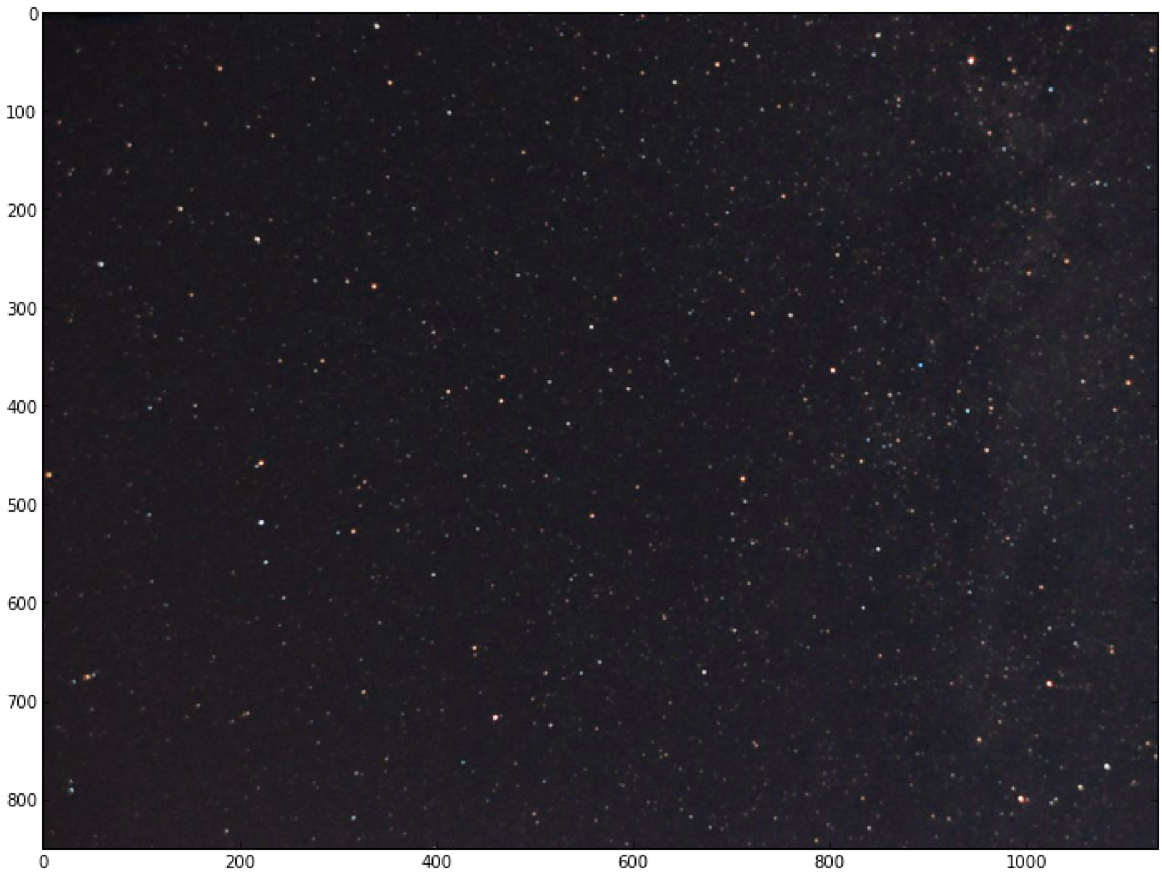
Using OpenCV and Fast Library Approximate Nearest Neighbour (FLANN) to search for matches.

In [32]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

We'll Import our image to use as an example.

In [33]:
```
sample_image = cv2.imread('star.jpg')
plt.figure(figsize(12,12))
plt.imshow(sample_image)
```

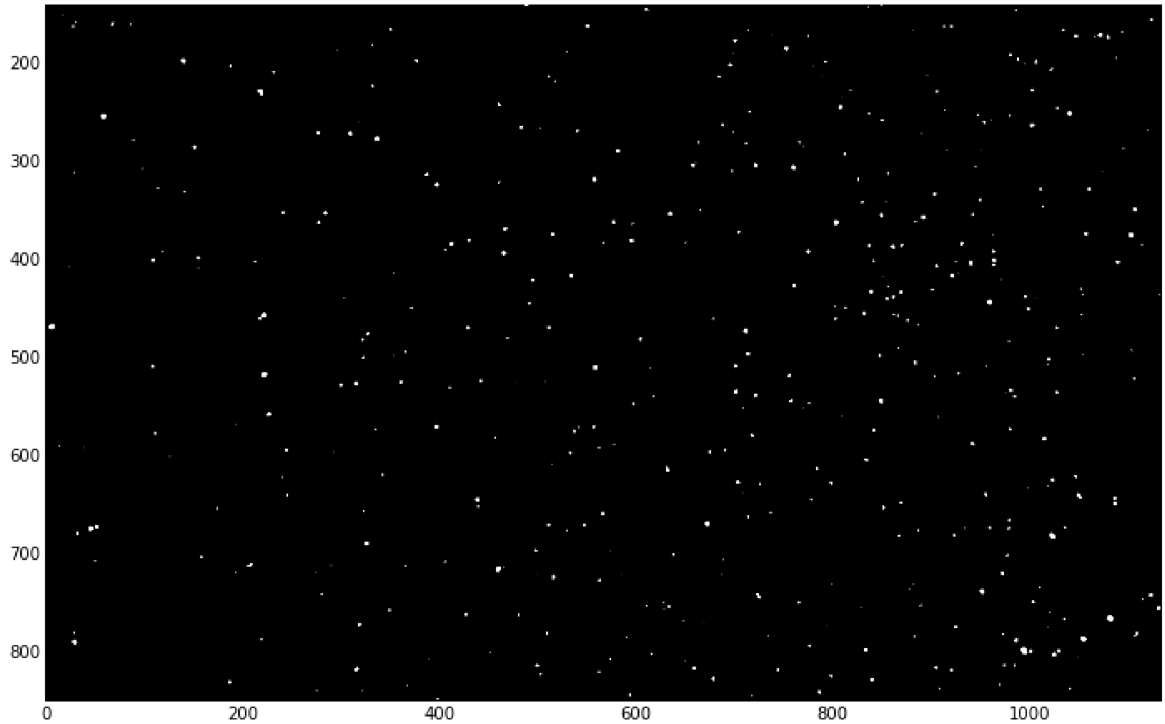
Out[33]: <matplotlib.image.AxesImage at 0x7b8ca50>



We can now take a threshold value for this image, in this case, the average luminosity of the image.

In [34]:
```
gray = cv2.cvtColor(sample_image,cv2.COLOR_BGR2GRAY)
ret,th1 = cv2.threshold(gray,80,255,cv2.THRESH_BINARY)
plt.imshow(th1,'gray')
```

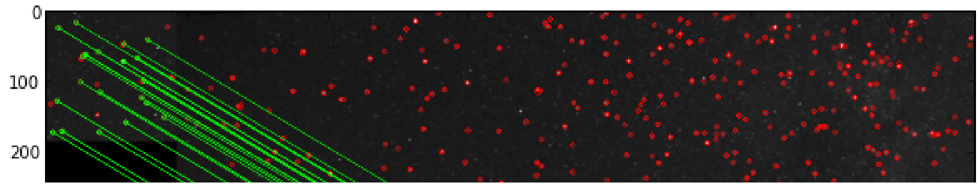
Out[34]: <matplotlib.image.AxesImage at 0x6756a50>

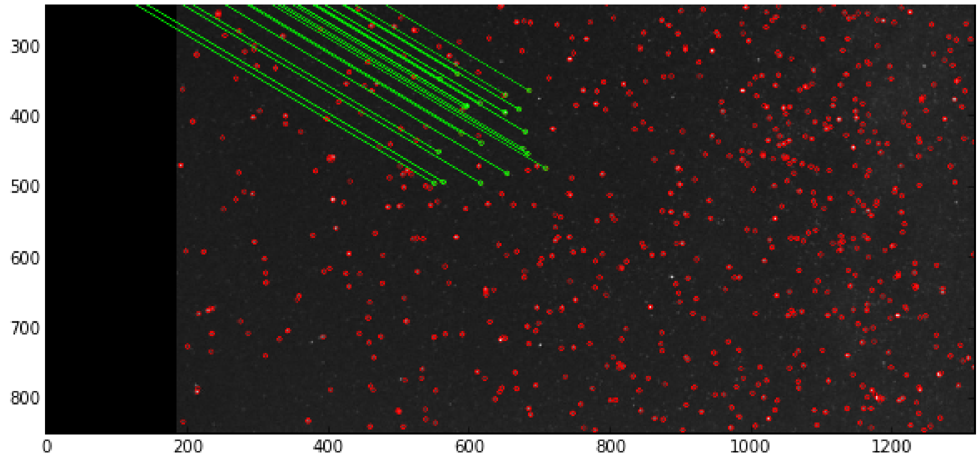


So without much effort, we've made it extremely easy to detect stars in the night sky from the position of white cells in the image. This only solves the easy part of the problem, to find the position of a star field is without the celestial mackdro f.

In [35]:

```
img1 = cv2.imread('star_crop.jpg',0)          # queryImage
#img1 = cv2.imread('star_crop_distort.jpg',0) #The Distorted image
img2 = cv2.imread('star.jpg',0) # trainImage
# Initiate SIFT detector
sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in xrange(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.figure(figsize(10,10))
plt.imshow(img3,),plt.show()
```





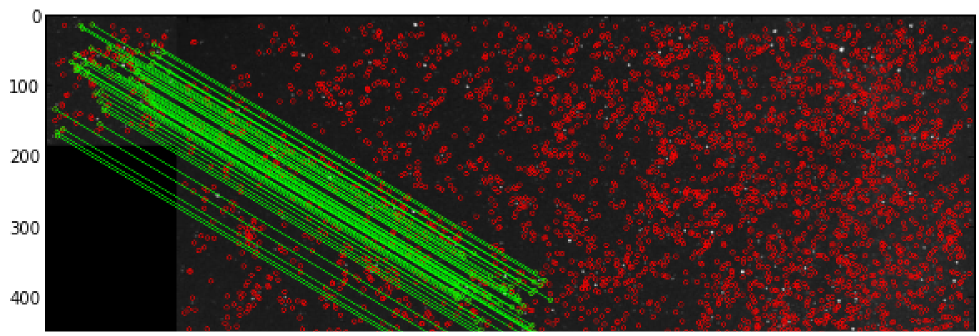

```
Out[35]: (<matplotlib.image.AxesImage at 0xc717890>, None)
```

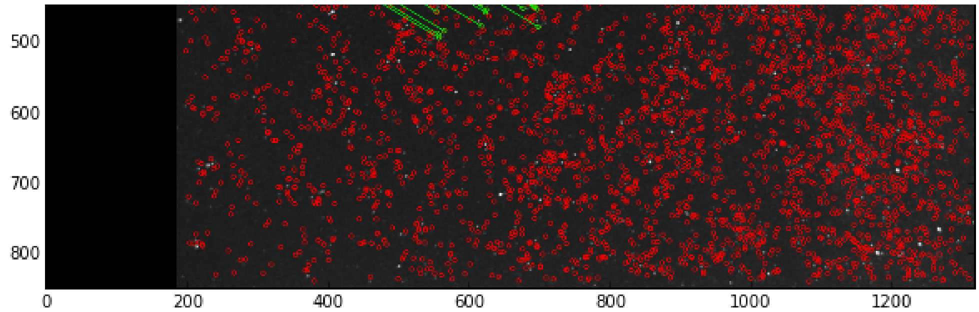
However, sift contains position information and is comparatively slow, we'll use SURF instead. With invariant direction, only size as stars should look the same from any angle (obviously).

In [36]:
```
surf = cv2.SURF(10)
surf.upright = True #Direction invariant
kp1, des1 = surf.detectAndCompute(img1,None)
kp2, des2 = surf.detectAndCompute(img2,None)
#for kps in kp1:
#    print "x: " + str(kps.pt[0]) + " y: " + str(kps.pt[1]) + " Size: " + str(kps.size) + " Octav
#    + str(kps.octave) + " Response: " + str(kps.response)
#for desc in des1:
#    print desc
#    break
print len(kp1)
```

```
154
```

In [37]:
```
# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=100)   # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in xrange(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.figure(figsize(10,10))
plt.imshow(img3,),plt.show()
```





Out[37]: (<matplotlib.image.AxesImage at 0x9f4de50>, None)

Let's use this to give us a relative position, if we assume the middle of the picture is 0az (lat), 0zen(lon). We'll assume this picture gives us a full view of the sky.

In [38]:
```
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        selected_match = m
        print "Destination x:" + str(kp1[m.queryIdx].pt[0] ) + "  y:" + str(kp1[m.queryIdx].pt[1]
        print "Sky x:" + str(kp2[m.trainIdx].pt[0] ) + "  y:" + str(kp2[m.trainIdx].pt[1] )
        break #Just so we pick one point and
```

```
Destination x:53.7616882324  y:68.4222717285
Sky x:409.82510376  y:391.379638672
```

Get the centroid of the image and relate it to a keypoint.

In [39]:
```
print "Destination x:" + str(kp1[m.queryIdx].pt[0] ) + "  y:" + str(kp1[m.queryIdx].pt[1] )
print "Sky x:" + str(kp2[m.trainIdx].pt[0] ) + "  y:" + str(kp2[m.trainIdx].pt[1] )
height, width = img1.shape
height2, width2 = img2.shape
print height2
print width2
cent_x = kp1[selected_match.queryIdx].pt[0] - width/2
cent_y = kp1[selected_match.queryIdx].pt[1] - height/2
print "Offset x: " + str(cent_x ) + " y:" + str(cent_y)
```

```
Destination x:53.7616882324  y:68.4222717285
Sky x:409.82510376  y:391.379638672
850
1134
Offset x: -38.2383117676 y:-23.5777282715
```

To the center of the main image

In [40]:
```
#print kp2[selected_match.trainIdx].pt[0]
#print kp2[selected_match.trainIdx].pt[1]
azimuth = 180*((kp2[selected_match.trainIdx].pt[0] - cent_x)-(width2/2))/(width2/2)
zenith = 180*((height2/2)-(kp2[selected_match.trainIdx].pt[1] - cent_y))/(height2/2)
print "Azimuth angle in the sky: " + str(azimuth)
print "Zenith angle in the sky: " + str(zenith)
```

```
Azimuth angle in the sky: -37.7576458643
Zenith angle in the sky: 4.25335047105
```

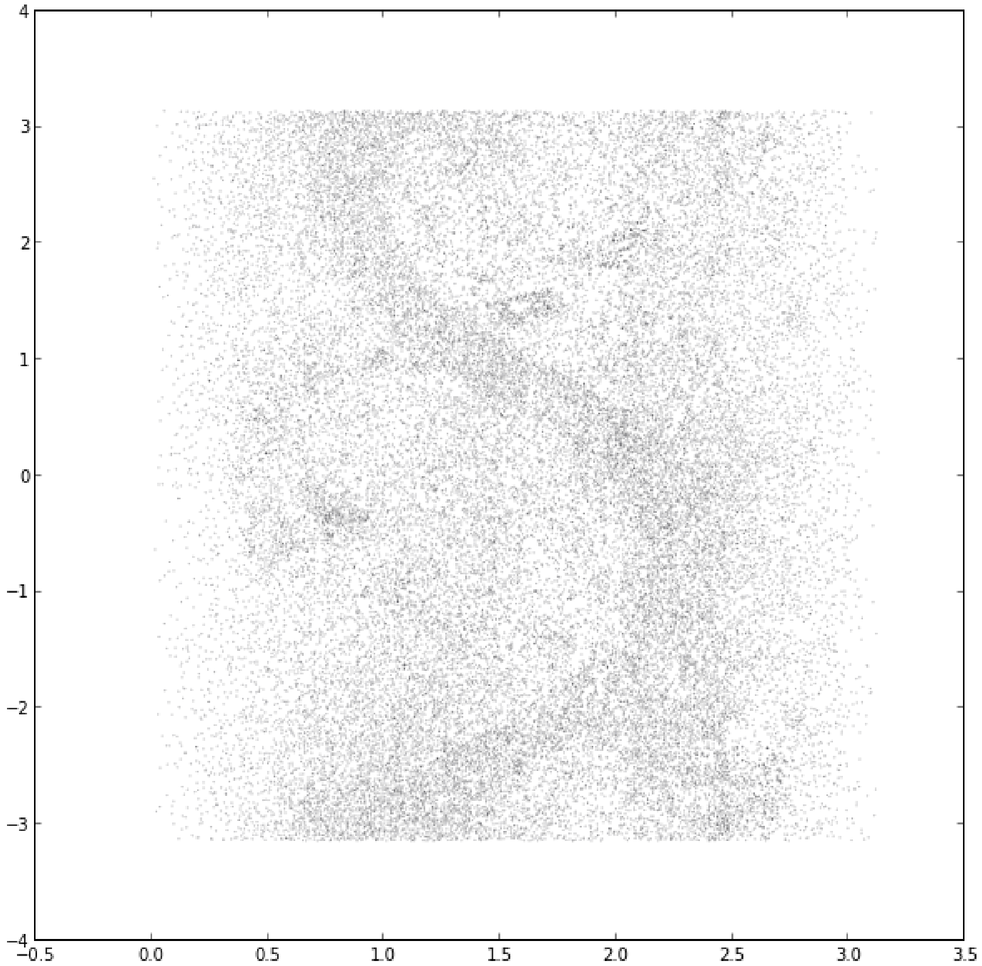
Now we want to do this for the whole starfield

In [41]:
```
import csv
from math import *

class Point:
        def __init__(self, x, y, z, mag=0):
                self.x = x
                self.y = y
                self.z = z
                self.mag = mag

class Polar:
        def __init__(self, r, theta, vphi, mag=0):
                self.r = r
                self.theta = theta  # 0 < theta < pi
                self.vphi = vphi    # -pi < vphi < pi
                self.mag = mag
```

```
def get_polar_from(point):
        r = sqrt(point.x**2 + point.y**2 + point.z**2)
        theta = acos(point.z/r)
        varphi = atan2(point.y,point.x)
        return Polar(r, theta, varphi, point.mag)

def get_data_from_csv(filename):
        points = []
        with open(filename, 'rb') as f:
                f.readline()
                reader = csv.reader(f, delimiter=',', quoting=csv.QUOTE_NONE)
                for row in reader:
                        try:
                                data = row[17:20]
                                data.append(row[13])
                                data = map(float,data)
                        except:
                                pass

                        points.append(Point(*data))

        return points

cartesian_stars = get_data_from_csv('hygxyz_bigger9.csv')
polar_stars = map(get_polar_from, cartesian_stars)

a1 = map(lambda x: x.theta, polar_stars)
a2 = map(lambda x: x.vphi, polar_stars)
mag = map(lambda x: x.mag, polar_stars)
```

In [42]:
```
plt.scatter(a1, a2, s=0.01)
```

Out[42]: <matplotlib.collections.PathCollection at 0xb3a9610>

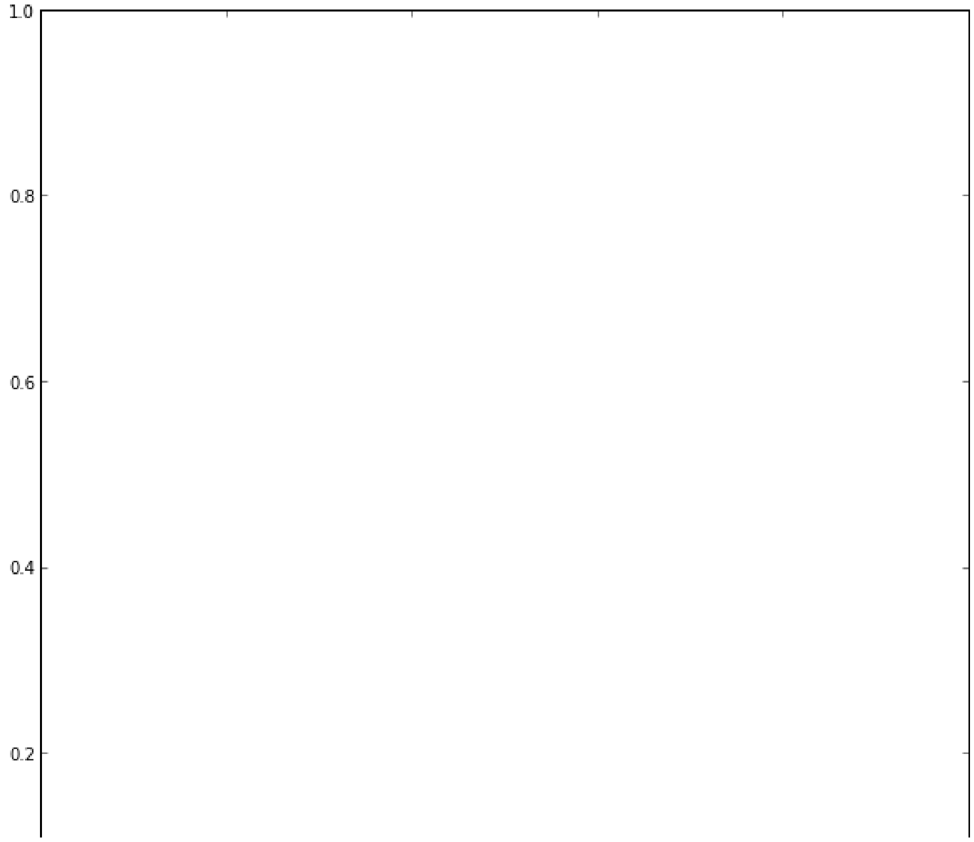


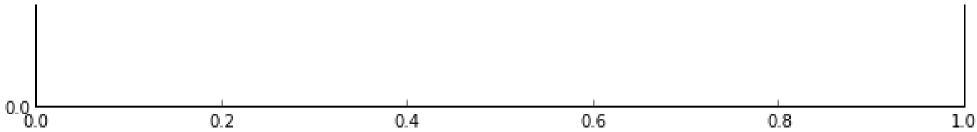
In [43]:
```
import pylab
```

```
fig = pylab.gcf()
starfield = fig.canvas.print_raw
```

In [44]:
```
from cStringIO import StringIO
sio=StringIO()
fig.canvas.print_png(sio)

img2 = cv2.imdecode(np.asarray(bytearray(sio),dtype=np.uint8),-1)
plt.imshow(img2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/ben/spaceTrack/<ipython-input-44-c235c4492b99> in <module>()
      4
      5 img2 = cv2.imdecode(np.asarray(bytearray(sio),dtype=np.uint8),-1)
----> 6 plt.imshow(img2)

/usr/lib/pymodules/python2.7/matplotlib/pyplot.pyc in imshow(X, cmap, norm, aspect, interpolation, alpha, vmin, vmax, origin, extent, shape, filternorm, filterrad, imlim, resample, url, hold, **kwargs)
   2375         ax.hold(hold)
   2376     try:
-> 2377         ret = ax.imshow(X, cmap, norm, aspect, interpolation, alpha, vmin, vmax, origin, extent, shape, filternorm, filterrad, imlim, resample, url, **kwargs)
   2378         draw_if_interactive()
   2379     finally:

/usr/lib/pymodules/python2.7/matplotlib/axes.pyc in imshow(self, X, cmap, norm, aspect, interpolation, alpha, vmin, vmax, origin, extent, shape, filternorm, filterrad, imlim, resample, url, **kwargs)
   6794                        filterrad=filterrad, resample=resample, **kwargs)
   6795
-> 6796         im.set_data(X)
   6797         im.set_alpha(alpha)
   6798         self._set_artist_props(im)

/usr/lib/pymodules/python2.7/matplotlib/image.pyc in set_data(self, A)
    405
    406         if self._A.dtype != np.uint8 and not np.can_cast(self._A.dtype, np.float):
--> 407             raise TypeError("Image data can not convert to float")
    408
    409         if (self._A.ndim not in (2, 3) or

TypeError: Image data can not convert to float
```





In [ ]:

```
# Initiate SIFT detector
sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in xrange(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.figure(figsize(10,10))
plt.imshow(img3,),plt.show()
```