```
from Calibration import *
import pylab
```

```
uv = np.loadtxt("pts2d-norm-pic_a.txt")
uv1 = np.loadtxt("pts2d-pic_a.txt")
uv2 = np.loadtxt("pts2d-pic_b.txt")
xyz = np.loadtxt("pts3d-norm.txt")
```

```
ima = cv2.imread('pic_a.jpg')
imb = cv2.imread('pic_b.jpg')
```

1: Calibration

1.1 We start by computing the calibration matrix. The residual is computed as the sum of squared differences.

```
M = compute_projection_matrix(uv, xyz)
print "residual:", compute_residual(uv, xyz, M)
```

```
residual: 0.000168775782172
```

1.2 here are the results for k = 8, 12, and 16 with 100 repetition. The results shows that having more points to compute the calibration matrix improve the average residual. If we have less points, is easyier to fit perfectly those points, but the obtained generalize poorly on new data. This kind of overfitting is due to the presence of noise, and the discretisation of the scene through the image.

```
for k in [8, 12, 16]:
    (mini, avg) = k_fold(k, uv, xyz)
    print "when k=", k, " the min residual is: ", mini, " and the avg residual
is: ", avg
```

```
when k= 8  the min residual is:  1.40734342519e-05  and the avg residual is:
0.000519085480256
when k= 12  the min residual is:  8.54372231494e-06  and the avg residual is:
0.000105485601961
when k= 16  the min residual is:  8.84590434401e-06  and the avg residual is:
0.000104563854896
```

1.3 we can then compute the camera center

```
print find_center(M)
```

```
[[-1.51267725]
 [-2.35168754]
 [ 0.28262819]]
```

2 Fundamental matrix estimation

2.1 we now compute F from the 2 set of points

```
F = compute_fundamental(uv1, uv2)
print F
```

```
[[ -6.60698417e-07   7.91031621e-06  -1.88600198e-03]
 [  8.82396296e-06   1.21382933e-06   1.72332901e-02]
 [ -9.07382302e-04  -2.64234650e-02   9.99500092e-01]]
```

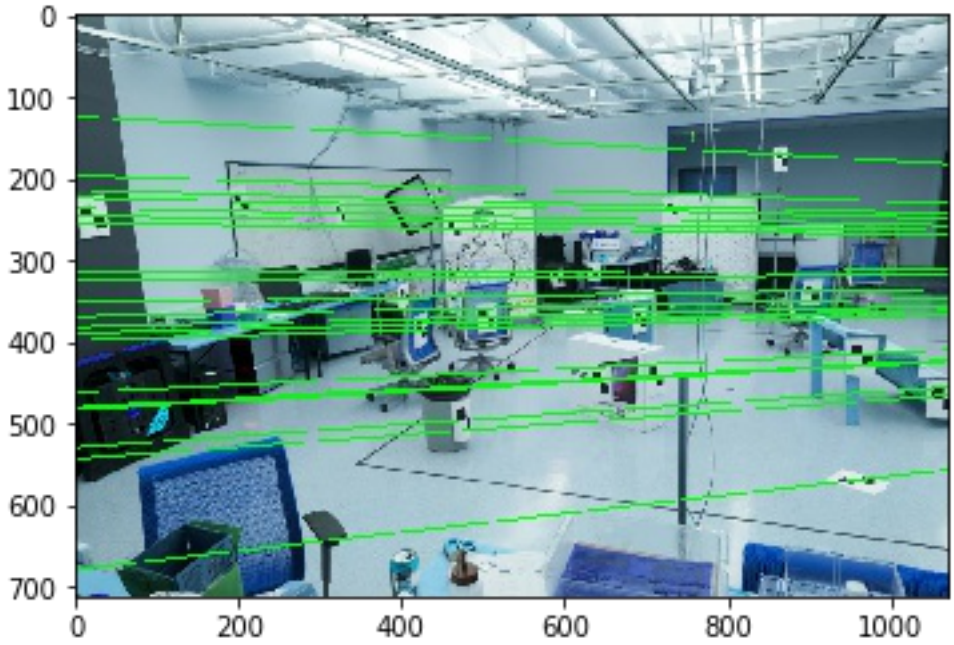
2.2 we can now use the SVD avoid having a full rank solution. By looking at the frobenius distance between the two matrix we can check how close they are.

```
F2 = clean_fundamental(F)
print F2
print "frobenius distance between F and F2:", np.linalg.norm(F-F2, 'fro')
```

```
[[ -5.36264198e-07   7.90364771e-06  -1.88600204e-03]
 [  8.83539184e-06   1.21321685e-06   1.72332901e-02]
 [ -9.07382264e-04  -2.64234650e-02   9.99500092e-01]]
frobenius distance between F and F2: 1.25137298813e-07
```

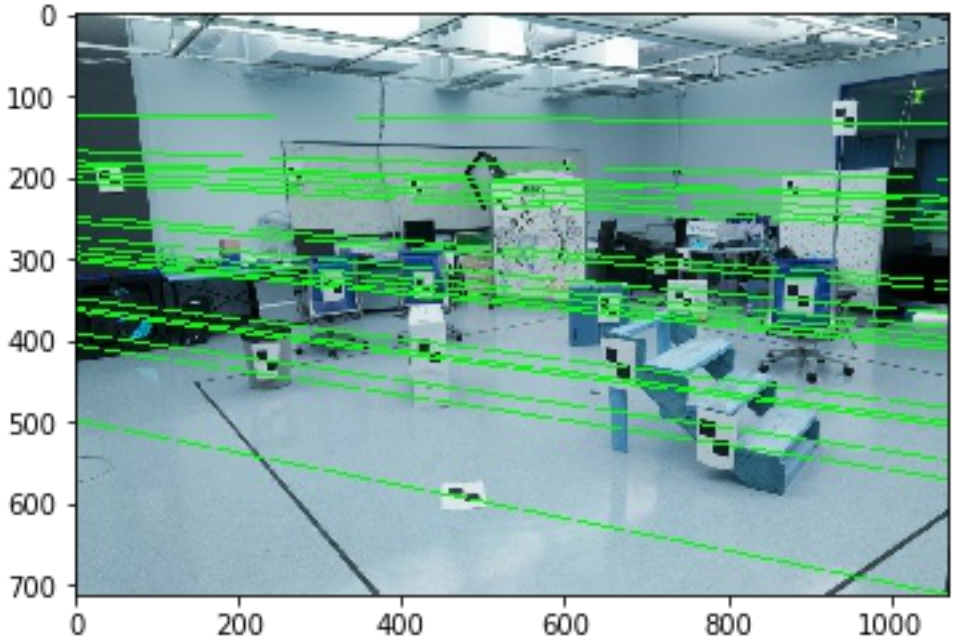
2.3 with the cleaned F we can draw the epipolar lines

```
res = draw_epipolar_line(imb, F2, uv1)
pylab.imshow(res, cmap=pylab.gray())
pylab.show()
cv2.imwrite(filename="img/pic_b-epipolar.jpg", img=res)
```




```
True
```

```
res = draw_epipolar_line_2(ima, F2, uv2)
pylab.imshow(res, cmap=pylab.gray())
pylab.show()
cv2.imwrite(filename="img/pic_a-epipolar.jpg", img=res)
```




```
True
```

extra question: Normalization

2.4.x

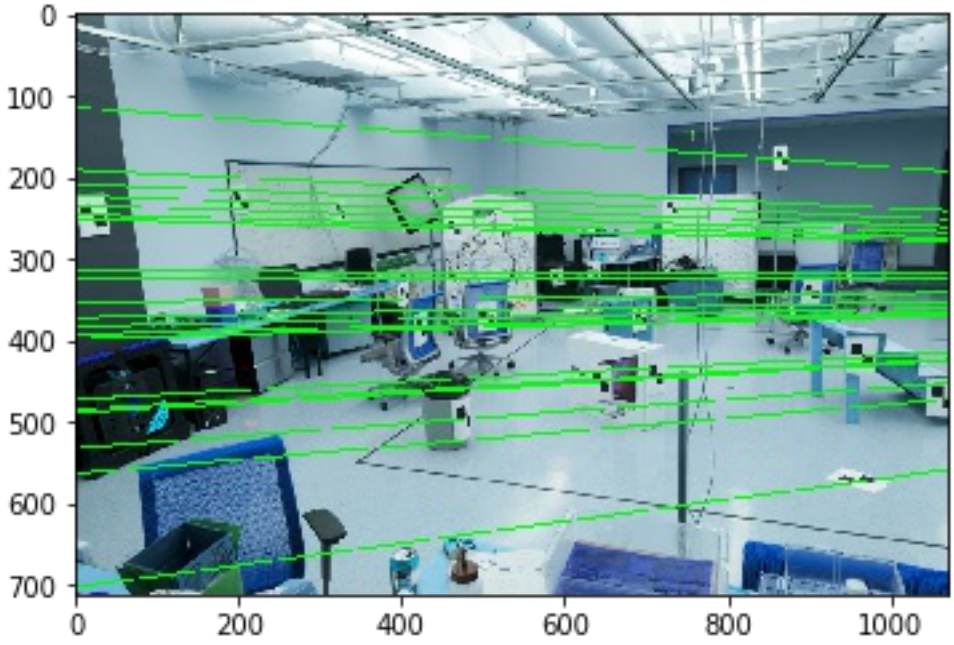
```
(uv1_norm, Ta) = normalize(uv1)
print "Ta: ", Ta
(uv2_norm, Tb) = normalize(uv2)
print "Tb: ", Tb
```

```
Ta:  [[ 0.00422627  0.         -2.36227228]
 [ 0.          0.0085811  -2.79400691]
 [ 0.          0.          1.        ]]
Tb:  [[ 0.00385727  0.         -2.37877704]
 [ 0.          0.00958482 -3.32497426]
 [ 0.          0.          1.        ]]
```

```
F_norm = Tb.T * compute_fundamental(uv1_norm, uv2_norm) * Ta
F2_norm = clean_fundamental(F_norm)
```

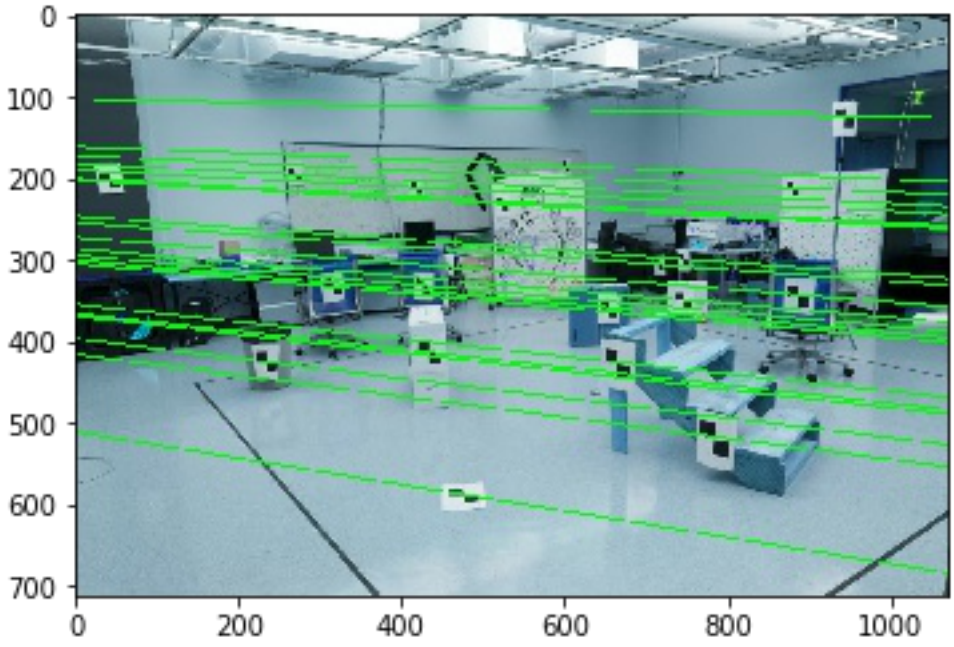
2.5.x

```
res = draw_epipolar_line(imb, F2_norm, uv1)
pylab.imshow(res, cmap=pylab.gray())
pylab.show()
cv2.imwrite(filename="img/pic_b_normed-epipolar.jpg", img=res)
```




```
True
```

```
res = draw_epipolar_line_2(ima, F2_norm, uv2)
pylab.imshow(res, cmap=pylab.gray())
pylab.show()
cv2.imwrite(filename="img/pic_a_normed-epipolar.jpg", img=res)
```




```
True
```