assignment_6_1

September 6, 2022

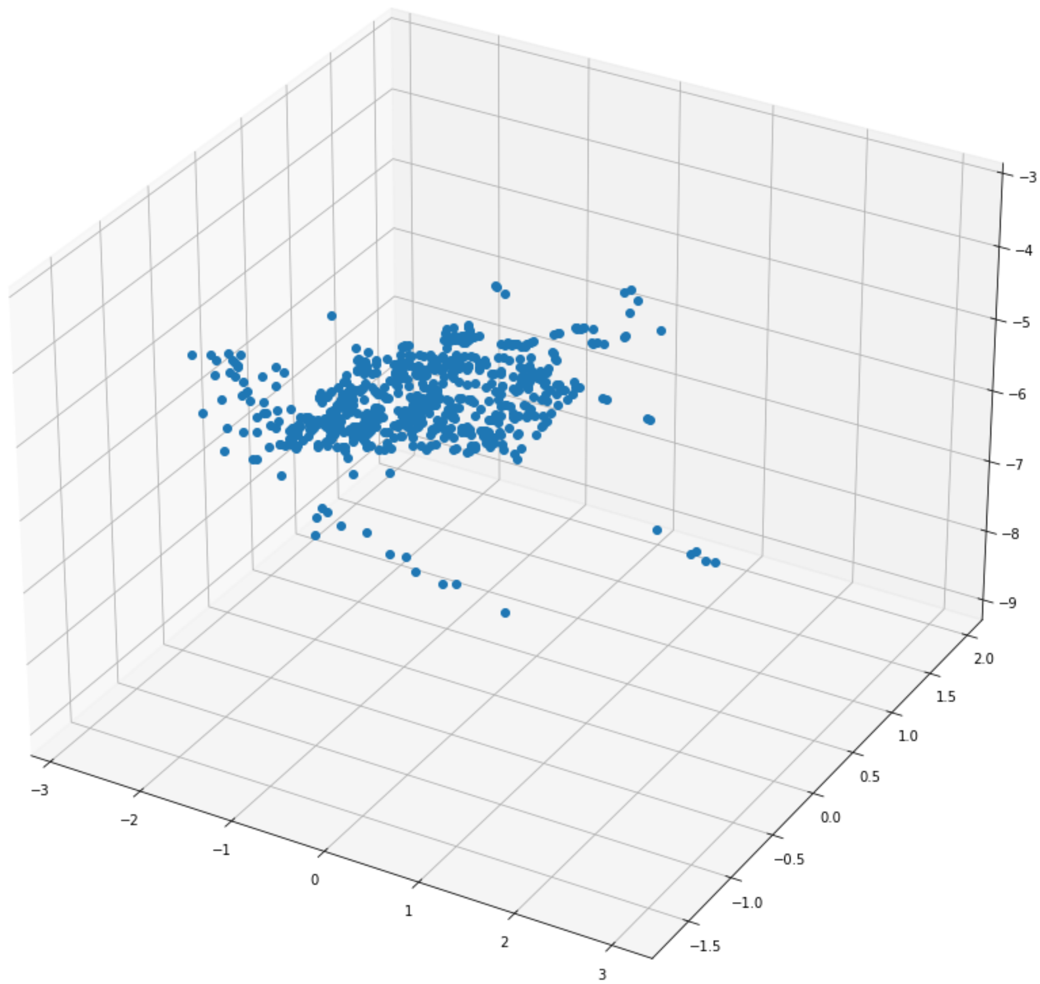
[1]:
```
import numpy as np
#setup camera with a simple camera matrix P
f = 100
cx = 200
cy = 200
K = np.array([[f, 0, cx], [0, f, cy], [0, 0, 1]])
I = np.eye(3) #i.e., R
t = np.array([[0], [0], [0]])
P = np.dot(K, np.hstack((I, t)))
```

[2]:
```
def project(P, X): #X is an array of 3D points
    x = np.dot(P, X)
    for i in range(3): #convert to inhomogeneous coordinates
        x[i] /= x[2]
    return x
```

[3]:
```
#load data
points_3D = np.loadtxt('house.p3d').T #T means tranpose
points_3D = np.vstack((points_3D, np.ones(points_3D.shape[1])))
print (points_3D.shape)
print(P)
```

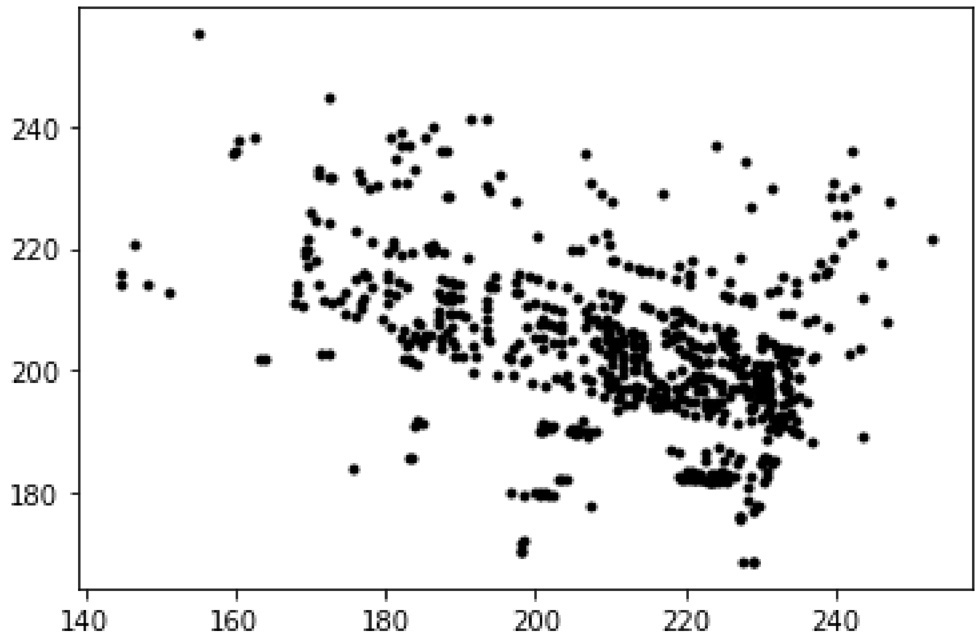
```
(4, 672)
[[100.   0. 200.   0.]
 [  0. 100. 200.   0.]
 [  0.   0.   1.   0.]]
```

[4]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
fig = plt.figure(figsize = [15,15])
ax = fig.gca(projection = "3d")
ax.view_init(elev = None, azim = None) #you can set elevation and azimuth with␣
 ↪different values
ax.plot(points_3D[0], points_3D[1], points_3D[2], 'o')
plt.draw()
plt.show()
```



[5]:

```
#projection
points_2D = project(P, points_3D)
#plot projection
from matplotlib import pyplot as plt
plt.plot(points_2D[0], points_2D[1], 'k.')
plt.show()
```



[6]:
```
print(points_2D.shape)
print(points_3D.shape)
```

```
(3, 672)
(4, 672)
```

[7]:
```
n_points = 6
points_3D_sampled = points_3D[:,:n_points]
points_2D_sampled = points_2D[:,:n_points]
```

[8]:
```
A = np.zeros((2*n_points, 12), np.float32)
for i in range(n_points):
    A[2*i,:4] = points_3D_sampled[:,i].T
    A[2*i,8:12] = -points_2D_sampled[0,i] * points_3D_sampled[:,i].T
    A[2*i+1,4:8] = points_3D_sampled[:,i].T
    A[2*i+1,8:12] = -points_2D_sampled[1,i] * points_3D_sampled[:,i].T
```

[9]:
```
from scipy import linalg
p = linalg.solve(A, np.zeros((12, 1), np.float32))
print(p)
```

```
[[-0.]
 [ 0.]
 [-0.]
 [ 0.]
 [-0.]
 [ 0.]
```

```
 [-0.]
 [ 0.]
 [-0.]
 [-0.]
 [ 0.]
 [ 0.]]
```

```
<ipython-input-9-f9ae5b52bf9e>:2: LinAlgWarning: Ill-conditioned matrix
(rcond=3.3444e-12): result may not be accurate.
  p = linalg.solve(A, np.zeros((12, 1), np.float32))
```

[10]:
```
U, S, V = linalg.svd(A)
```

[11]:
```
minS = np.min(S)
conditon = (S == minS)
minID = np.where(conditon)
print('index of the smallest singular value is: ', minID[0])
```

```
index of the smallest singular value is:  [11]
```

[12]:
```
P_hat = V[minID[0],:].reshape(3, 4) / minS
```

[13]:
```
print(P)
print(P_hat)
```

```
[[100.   0. 200.   0.]
 [  0. 100. 200.   0.]
 [  0.   0.   1.   0.]]
[[ 9.3289719e+04 -4.0221130e+02  1.8646670e+05 -2.8556070e+01]
 [ 3.6980091e+01  9.2909969e+04  1.8644198e+05 -1.8464259e+02]
 [ 9.3261771e-02 -1.7817017e+00  9.3240582e+02  6.0331091e-02]]
```

[14]:
```
x_P_hat = project(P_hat, points_3D_sampled[:, 0])
print(x_P_hat)
x_P = points_2D_sampled[:,0]
print(x_P)
```

```
[225.25317749 181.44376362   1.        ]
[225.25322768 181.44331296   1.        ]
```

[15]:
```
x_P = points_2D
x_P_hat = project(P_hat, points_3D)
dist = 0
for i in range(x_P.shape[1]):
    dist += np.linalg.norm(x_P[:,i] - x_P_hat[:,i])
dist /= x_P.shape[1]
print(dist)
```

```
0.01231894491141471
```

[18]:

```
proportions = [i/100 for i in range(10,110,10)]
min_dist = 111111
best_p = 0

for p in proportions:
    n_points = int(p * points_3D.shape[1])
    points_3D_sampled = points_3D[:,:n_points]
    points_2D_sampled = points_2D[:,:n_points]

    A = np.zeros((2*n_points, 12), np.float32)
    for i in range(n_points):
        A[2*i,:4] = points_3D_sampled[:,i].T
        A[2*i,8:12] = -points_2D_sampled[0,i] * points_3D_sampled[:,i].T
        A[2*i+1,4:8] = points_3D_sampled[:,i].T
        A[2*i+1,8:12] = -points_2D_sampled[1,i] * points_3D_sampled[:,i].T

    U, S, V = linalg.svd(A)

    minS = np.min(S)
    condition = (S == minS)
    minID = np.where(condition)
    P_hat = V[minID[0],:].reshape(3, 4) / minS

    x_P = points_2D
    x_P_hat = project(P_hat, points_3D)

    dist = 0
    for i in range(x_P.shape[1]):
        dist += np.linalg.norm(x_P[:,i] - x_P_hat[:,i])
    dist /= x_P.shape[1]

    print (n_points, dist)

    if dist < min_dist:
        min_dist = dist
        best_p = p

print("best number of points ", int(best_p * points_3D.shape[1]))
print("min distance", min_dist)
```

```
67 0.004087307426213173
134 0.0017667722945224182
201 0.0003026686403123419
268 0.0005762488178965705
336 0.004115267511264261
403 0.0032054763529057425
470 0.00023883683435455002
537 0.0032774937700263684
```

```
604 0.0036446909522239207
672 0.0006061415839525945
best number of points  470
min distance 0.00023883683435455002
```

[19]:
```python
import homography
import sfm
import ransac
```

[20]:
```python
import cv2 as cv
sift = cv.xfeatures2d.SIFT_create()
img1 = cv.imread('alcatraz1.jpg')
img1_gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
kp1, des1 = sift.detectAndCompute(img1_gray, None)
img2 = cv.imread('alcatraz2.jpg')
img2_gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
kp2, des2 = sift.detectAndCompute(img2_gray, None)
img1_kp = img1.copy()
img1_kp = cv.drawKeypoints(img1, kp1, img1_kp)
print("Number of detected keypoints in img1: %d" % (len(kp1)))
img2_kp = img2.copy()
img2_kp = cv.drawKeypoints(img2, kp2, img2_kp)
print("Number of detected keypoints in img2: %d" % (len(kp2)))
img1_2_kp = np.hstack((img1_kp, img2_kp))
plt.figure(figsize = (20, 10))
plt.imshow(img1_2_kp[:,:,::-1])
plt.axis('off')
```

```
Number of detected keypoints in img1: 25477
Number of detected keypoints in img2: 24552
```

[20]:
```
(-0.5, 3871.5, 1295.5, -0.5)
```

[21]:
```
bf = cv.BFMatcher(crossCheck = True) #crossCheck = True means we want to find␣
↪consistent matches
matches = bf.match(des1, des2)
matches = sorted(matches, key = lambda x:x.distance)
print("Number of consistent matches: %d" % len(matches))
```

```
Number of consistent matches: 9690
```

[22]:
```
img1_2_matches = cv.drawMatches(img1, kp1, img2, kp2,
matches[:20],
None,
flags = cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.figure(figsize = (20, 10))
plt.imshow(img1_2_matches[:,:,::-1])
plt.axis('off')
```

[22]: (-0.5, 3871.5, 1295.5, -0.5)

[24]:
```
n_matches = 1000
kp1_array = np.zeros((2, n_matches), np.float32)
for i in range(n_matches):
    kp1_array[0][i] = kp1[matches[i].queryIdx].pt[0]
    kp1_array[1][i] = kp1[matches[i].queryIdx].pt[1]
kp2_array = np.zeros((2, n_matches), np.float32)
for i in range(n_matches):
    kp2_array[0][i] = kp2[matches[i].trainIdx].pt[0]
    kp2_array[1][i] = kp2[matches[i].trainIdx].pt[1]
```

[25]:
```
x1 = homography.make_homog(kp1_array)
x2 = homography.make_homog(kp2_array)
```

[26]:
```
K = np.array([[2394,0,932], [0,2398,628], [0,0,1]])
P1 = np.array([[1,0,0,0], [0,1,0,0], [0,0,1,0]])
```

[27]:
```
x1n = np.dot(linalg.inv(K), x1)
x2n = np.dot(linalg.inv(K), x2)
```

[28]:
```
#estimate E with RANSAC
model = sfm.RansacModel()
E, inliers = sfm.F_from_ransac(x1n, x2n, model)
```

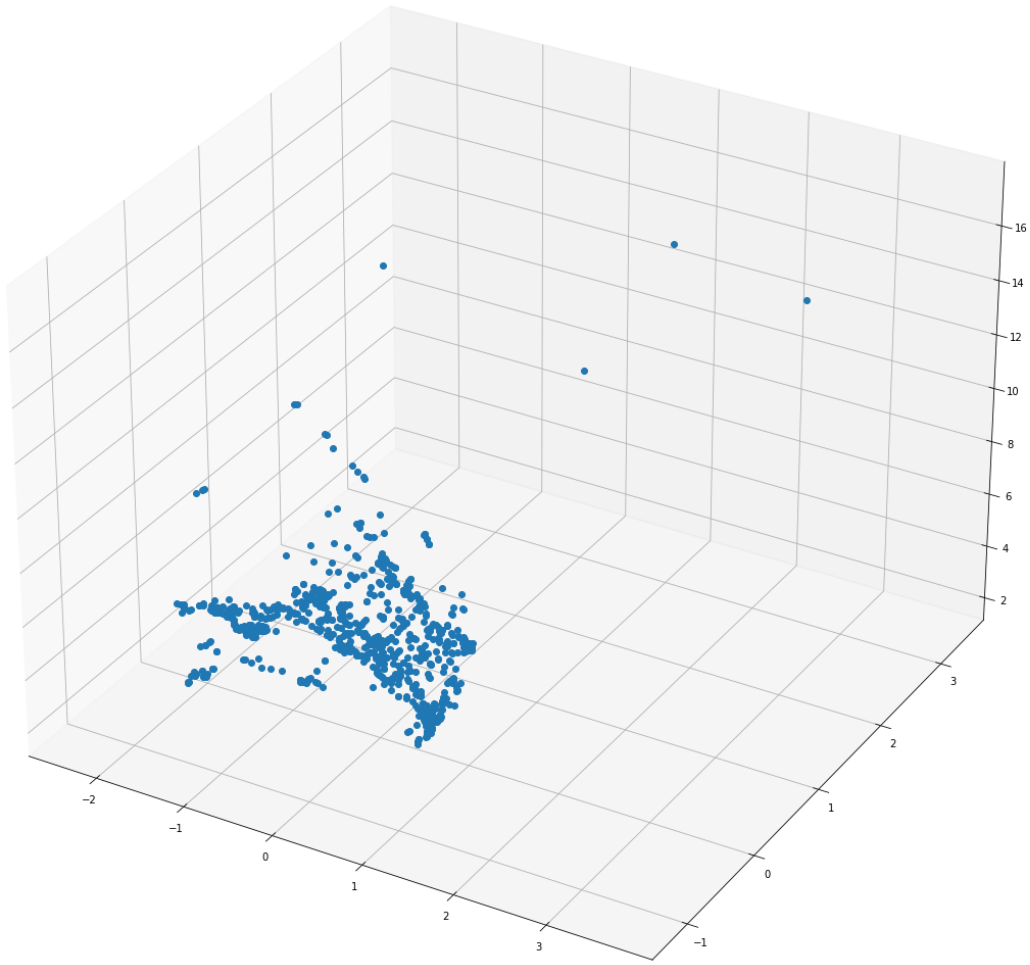
[29]:
```
#compute camera matrices (P2 will be list of four solutions)
P2_all = sfm.compute_P_from_essential(E)
#pick the solution with points in front of cameras
ind = 0
maxres = 0
for i in range(4):
#triangulate inliers and compute depth for each camera
    X = sfm.triangulate(x1n[:, inliers], x2n[:, inliers], P1, P2_all[i])
    d1 = np.dot(P1, X)[2]
    d2 = np.dot(P2_all[i], X)[2]
    s = sum(d1 > 0) + sum(d2 > 0)
    if s > maxres:
        maxres = s
        ind = i
        infront = (d1 > 0) & (d2 > 0)
P2 = P2_all[ind]
```

[30]:
```
#triangulate inliers and remove points not in front of both cameras
X = sfm.triangulate(x1n[:, inliers], x2n[:, inliers], P1, P2)
X = X[:, infront]
```

[31]:
```
print(len(X[0]))
```

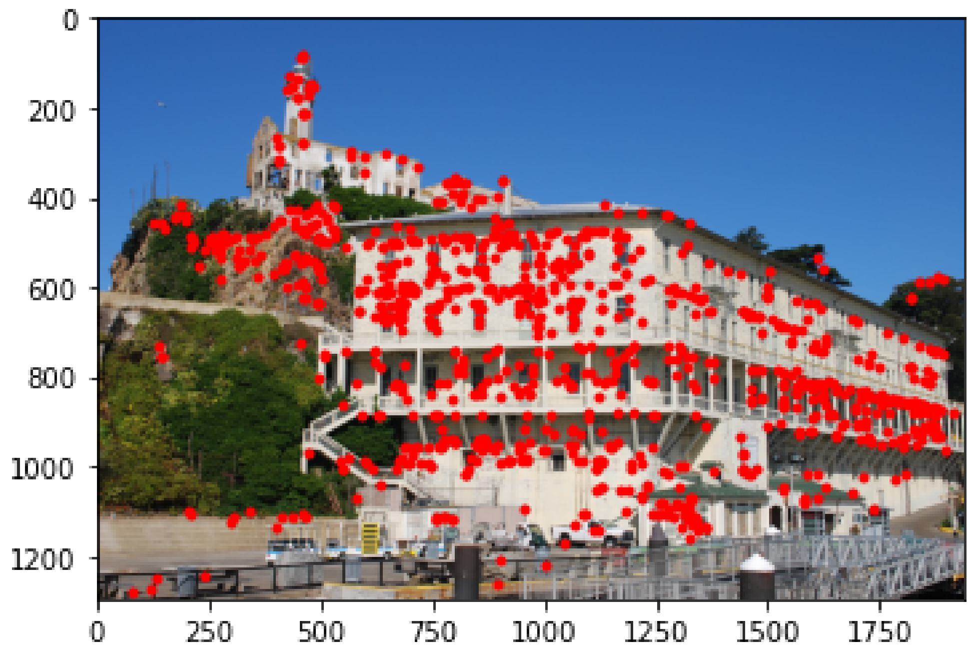
837

[32]:
```
#3D plot
fig = plt.figure(figsize = [20,20])
ax = fig.gca(projection = "3d")
ax.view_init(elev = None, azim = None) #you can set elevation and azimuth with␣
 ↪different values
ax.plot(X[0], X[1], X[2], 'o')
plt.draw()
plt.show()
```



[33]:
```
plt.figure()
plt.imshow(img2[:,:,::-1])
plt.plot(x2[0],x2[1],'r.')
```

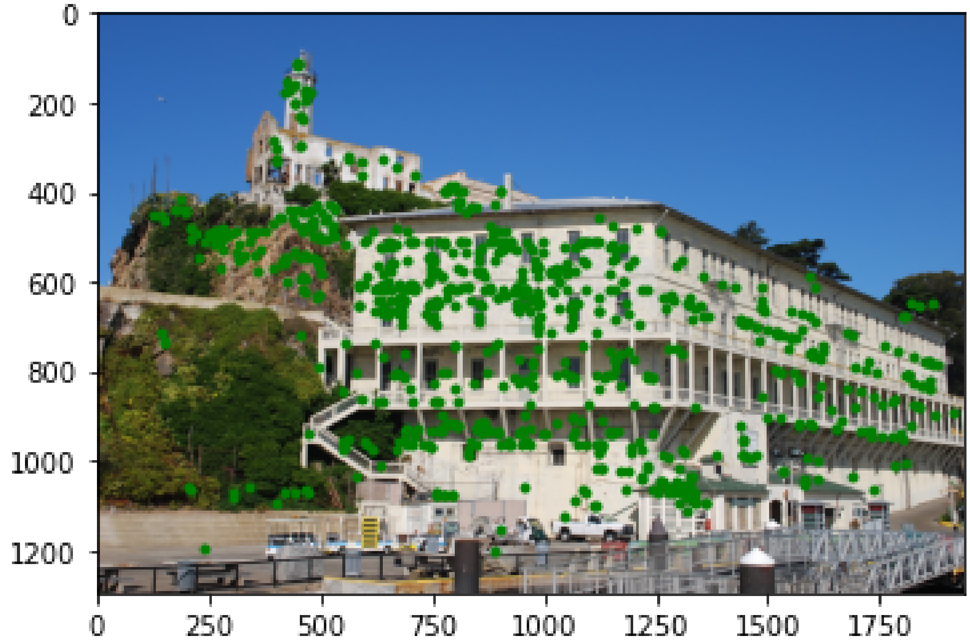
[33]: [<matplotlib.lines.Line2D at 0x1e23a3bf610>]



```
[35]: x2p = project(P2, X)
      x2p = np.dot(K, x2p)
```

```
[36]: plt.figure()
      plt.imshow(img2[:,:,::-1])
      plt.plot(x2p[0], x2p[1], 'g.')
```

```
[36]: [<matplotlib.lines.Line2D at 0x1e23a8a95b0>]
```


0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750