InferringWorldPoints

October 26, 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

[2]:
```
'''Function to plot a box based on the matrix given, as long as the first 8␣
↪values are the 8 corners'''
def plot(box, pointcolor = 'k', linecolor = 'k', degree1 = 30, degree2 = 45):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')


    ax.scatter3D(box[:, 0], box[: ,1], box[:, 2], c = pointcolor)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')


    verts = [[box[0],box[1],box[2],box[3]], [box[4],box[5],box[6],box[7]],␣
↪[box[0],box[1],box[5],box[4]], [box[3],box[2],box[6],box[7]]]
             #[box[1],box[2],box[6],box[5]], [box[1],box[2],box[5],box[7]]]

    # plot sides
    ax.add_collection3d(Poly3DCollection(verts, facecolors='w', linewidths=1,␣
↪edgecolors=linecolor, alpha=.25))
    ax.view_init(degree1, degree2)


#Rotation
'''Rotation function to return Omega based on what degree was given'''
def rotation(x):
    return np.array([[np.cos(x * np.pi/180), 0, -np.sin(x * np.pi/180)],
                    [0, 1, 0],
                    [np.sin(x * np.pi/180), 0 , np.cos(x * np.pi/180)]])


def homogenous(X):
    # print(X.shape[1])
```

```
    return np.vstack((X, np.ones([1,X.shape[1]])))

'''Pinhole camera funciton to find the projected values'''
def pinhole(W, Lambda, Omega, Tau):

    OmegaTau = np.hstack([Omega, Tau.T])

    X = Lambda@ OmegaTau @ W

    #divide by to the z values in order to make it 1
    X = X/X[2]

    X = X.T

    return X

'''Calculate the normalized coordinates'''
def findX_(Lambda, Omega, Tau, W):
    # print(Lambda.shape)
    # print(Omega.shape)
    # print(Tau.shape)
    # print(W.shape)
    OmegaTau = np.hstack([Omega,Tau.T])

    X =  Lambda @ OmegaTau @ W

    X = X/X[2]

    #equation 14.27
    X_ = np.linalg.inv(Lambda) @ X

    X_ = X_/X_[2]

    return X_

'''Consturct the systems of equations from equation 14.30'''
def constructA(W, X_):
    #make empty array to be able to populate
    #print(W.shape)
    A = np.zeros((W.shape[0]*2, 12))
    # print(A.shape)
    i = 0
    for w, x_ in zip(W, X_):
        # print("\nw: ",w)
        # print("x: ",x_)
        A[i] = np.array([w[0], w[1], w[2], 1, 0 ,0 ,0 ,0 , -w[0]*x_[0],
 ↪-w[1]*x_[0], -w[2]*x_[0], -x_[0]])
```

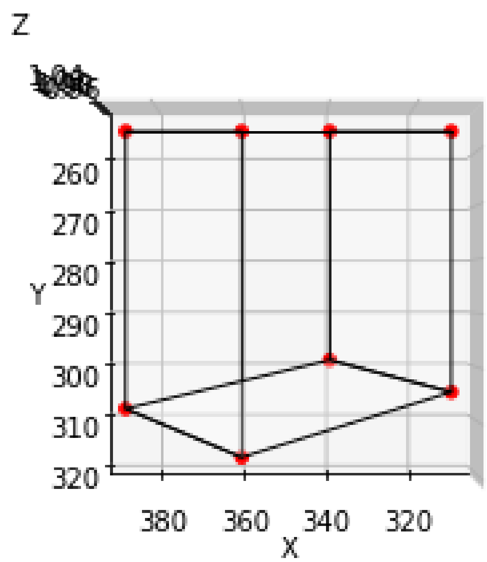
```
        A[i+1] =  np.array([0, 0, 0 ,0 ,w[0], w[1], w[2], 1 , -w[0]*x_[1],␣
↪-w[1]*x_[1], -w[2]*x_[1], -x_[1]])
        # print(A[i])
        # print(A[i+1])
        i += 2

    return A

''' Find the estimates of Omega and Tau'''
def findEstimate(A):

    U,L,V = np.linalg.svd(A)

    #set b hat equal to last column
    b_ = V.T[:,-1]

    Omega = np.array([[b_[0],b_[1],b_[2]],
                      [b_[4],b_[5],b_[6]],
                      [b_[8],b_[9],b_[10]]])

    Tau = np.array([b_[3], b_[7], b_[11]])

    #every fourth point is tau
    U_, L_, V_ = np.linalg.svd(Omega)

    Omega_hat = -(U_@V_)


    Tau_hat = np.array([np.sum(Omega_hat)/np.sum(Omega)*Tau])

    # Tau_hat = Tau_hat *-1

    # Tau_hat = Tau_hat*10

    # return np.hstack([Omega_hat, Tau_hat.T])
    return Omega_hat, Tau_hat
```

[3]:
```
camera1deg = 35

camera2deg = 0

camera3deg = -35

#skew factor
gamma = 0

#focal distances
```

```
phi = np.array([200,200])

#Principal point
delta = np.array([256,256])

#cube coordinates
C = np.array([[0, 50, 50, 0, 0, 50, 50, 0],
              [0, 0, 50, 50, 0, 0, 50, 50],
              [200, 200, 200, 200, 250, 250, 250, 250]])

#make the world coordinates homogenous
homogenousW = homogenous(C)



tau1 = np.array([[200,0,0]])

tau2 = np.array([[0, 0, 0]])

tau3 = np.array([[-200,0,0]])


#intrinsic matrix
Lambda = np.array([[phi[0], gamma, delta[0]],
                    [0, phi[1], delta[1]],
                    [0, 0, 1]])


#rotation
omega1 = rotation(camera1deg)

#rotation and translation
omegatau1 = np.hstack([omega1,tau1.T])



# print(Lambda.shape)
# print(omegatau.shape)
# print(C)


box = pinhole(homogenousW, Lambda, omega1, tau1)

print(box)

plot(box, 'r',degree1 = 90, degree2 = 90)
```

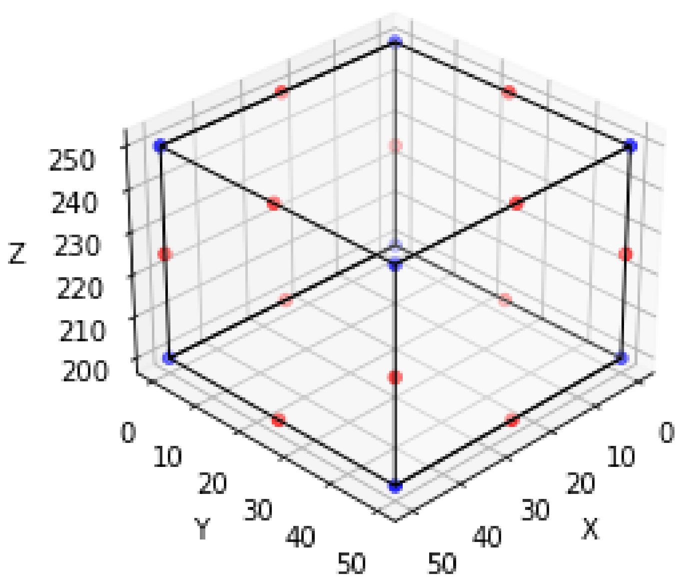
```
[[360.11341011 256.          1.        ]
```

```
[387.15455763 256.          1.        ]
[387.15455763 307.94556108  1.        ]
[360.11341011 317.03872944  1.        ]
[311.28242656 256.          1.        ]
[339.57803284 256.          1.        ]
[339.57803284 298.83263655  1.        ]
[311.28242656 304.83098355  1.        ]]
```



[4]:
```python
threedbox = C.T

additionalpoints = np.array([[25, 50, 25, 0, 0, 50, 50, 0, 25, 50, 25, 0],
                             [0, 0, 0, 0, 25, 25, 25, 25, 50, 50, 50, 50],
                             [200, 225, 250, 225, 200, 200, 250, 250, 200, 225, 250, 225]])

newpoints = additionalpoints.T

fignew = plt.figure()
ax = fignew.add_subplot(111, projection='3d')

threedverts = [[threedbox[0],threedbox[1],threedbox[2],threedbox[3]],
               [threedbox[4],threedbox[5],threedbox[6],threedbox[7]],
               [threedbox[0],threedbox[1],threedbox[5],threedbox[4]],
               [threedbox[3],threedbox[2],threedbox[6],threedbox[7]]]

ax.scatter3D(threedbox[:, 0], threedbox[: ,1], threedbox[:, 2], c='blue')
ax.scatter3D(newpoints[:, 0], newpoints[:, 1], newpoints[:, 2], c='red')
```

```
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')


ax.add_collection3d(Poly3DCollection(threedverts, facecolors='w', linewidths=1,␣
↪edgecolors='k', alpha=.25))
ax.view_init(30, 45)

#add the synthethic points
synthethicW = np.concatenate((C, additionalpoints), axis = 1)
HGsynthethicW = np.vstack((synthethicW, np.ones([1,20])))

# print("SynthethicW:\n", synthethicW.shape, "\n X_:\n", X_.shape)

# print("SynthethicW:\n", synthethicW.T, "\n\n X_:\n", X_.T)

# print(threebox)
```



[5]:
```
#set w equal to the box coordinates
omega2 = rotation(camera2deg)
omega3 = rotation(camera3deg)

#set w to homogenous box coordinates
w = HGsynthethicW
# w = homogenousW
```

```python
#get image coordinates
X1 = pinhole(w, Lambda, omega1, tau1).T
X2 = pinhole(w, Lambda, omega2, tau2).T
X3 = pinhole(w, Lambda, omega3, tau3).T

# Xcoord = np.concatenate([X1[0, np.newaxis],X2[0,np.newaxis],X3[0, np.
 ↪newaxis]])
Xcoord = np.concatenate([X1[0], X2[0], X3[0]])
Ycoord = np.array([X1[1],X2[1],X3[1]])

print("X :\n", Xcoord)
print("Y :\n", Ycoord)

#image coordinates make homogenous for matrix operation
# X1 = homogenous(X1.T)
# X2 = homogenous(X2.T)
# X3 = homogenous(X3.T)

#find normalized image coordinates
X1_ = findX_(Lambda, omega1, tau1, w)
X2_ = findX_(Lambda, omega2, tau2, w)
X3_ = findX_(Lambda, omega3, tau3, w)

print("\nNormalized Camera 1 image coordinates:\n", X1_)
print("\nNormalized Camera 2 image coordinates:\n",X2_)
print("\nNormalized Camera 3 image coordinates:\n",X3_)


Omega = np.array([omega1, omega2, omega3])

Tau = np.concatenate([tau1, tau2, tau3])

X_ = np.array([X1_[0], X2_[0], X3_[0]])

Y_ = np.array([X1_[1], X2_[1], X3_[1]])

oneX = np.array([X1, X2, X3])

print(oneX)
# print(Y_)

# print("\n\n")
# for x_, y_, omega in zip(X_, Y_, Omega):
#     print(x_)
#     print(omega)
```

```
X :
```

```
 [360.11341011 387.15455763 387.15455763 360.11341011 311.28242656
 339.57803284 339.57803284 311.28242656 374.7221427  361.0790536
 326.35604337 332.98508592 360.11341011 387.15455763 339.57803284
 311.28242656 374.7221427  361.0790536  326.35604337 332.98508592
 256.         306.         306.         256.         256.
 296.         296.         256.         281.         300.44444444
 276.         256.         256.         306.         296.
 256.         281.         300.44444444 276.         256.
 151.88658989 190.40386238 190.40386238 151.88658989 200.71757344
 238.22887884 238.22887884 200.71757344 169.29790755 217.46292512
 218.06106028 179.01491408 151.88658989 190.40386238 238.22887884
 200.71757344 169.29790755 217.46292512 218.06106028 179.01491408]
Y :
 [[256.         256.         307.94556108 317.03872944 256.
  256.         298.83263655 304.83098355 256.         256.
  256.         256.         286.51936472 281.97278054 277.41631828
  280.41549178 312.12622841 302.95099493 301.63554807 310.25664839]
 [256.         256.         306.         306.         256.
  256.         296.         296.         256.         256.
  256.         256.         281.         281.         276.
  276.         306.         300.44444444 296.         300.44444444]
 [256.         256.         329.99099203 317.03872944 256.
  256.         312.78295404 304.83098355 256.         256.
  256.         256.         286.51936472 292.99549602 284.39147702
  280.41549178 322.89366007 320.25480344 308.50760568 310.25664839]]

Normalized Camera 1 image coordinates:
 [[0.52056705 0.65577279 0.65577279 0.52056705 0.27641213 0.41789016
  0.41789016 0.27641213 0.59361071 0.52539527 0.35178022 0.38492543
  0.52056705 0.65577279 0.41789016 0.27641213 0.59361071 0.52539527
  0.35178022 0.38492543]
 [0.         0.         0.25972781 0.30519365 0.         0.
  0.21416318 0.24415492 0.         0.         0.         0.
  0.15259682 0.1298639  0.10708159 0.12207746 0.28063114 0.23475497
  0.22817774 0.27128324]
 [1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.        ]]

Normalized Camera 2 image coordinates:
 [[0.         0.25       0.25       0.         0.         0.2
  0.2        0.         0.125      0.22222222 0.1        0.
  0.         0.25       0.2        0.         0.125      0.22222222
  0.1        0.        ]
 [0.         0.         0.25       0.25       0.         0.
  0.2        0.2        0.         0.         0.         0.
  0.125      0.125      0.1        0.1        0.25       0.22222222
```

```
 0.2        0.22222222]
 [1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.        ]]

Normalized Camera 3 image coordinates:
 [[-0.52056705 -0.32798069 -0.32798069 -0.52056705 -0.27641213 -0.08885561
  -0.08885561 -0.27641213 -0.43351046 -0.19268537 -0.1896947  -0.38492543
  -0.52056705 -0.32798069 -0.08885561 -0.27641213 -0.43351046 -0.19268537
  -0.1896947  -0.38492543]
 [ 0.          0.          0.36995496  0.30519365  0.          0.
   0.28391477  0.24415492  0.          0.          0.          0.
   0.15259682  0.18497748  0.14195739  0.12207746  0.3344683   0.32127402
   0.26253803  0.27128324]
 [ 1.          1.          1.          1.          1.          1.
   1.          1.          1.          1.          1.          1.
   1.          1.          1.          1.          1.          1.
   1.          1.        ]]
[[[360.11341011 387.15455763 387.15455763 360.11341011 311.28242656
   339.57803284 339.57803284 311.28242656 374.7221427  361.0790536
   326.35604337 332.98508592 360.11341011 387.15455763 339.57803284
   311.28242656 374.7221427  361.0790536  326.35604337 332.98508592]
  [256.         256.         307.94556108 317.03872944 256.
   256.         298.83263655 304.83098355 256.         256.
   256.         256.         286.51936472 281.97278054 277.41631828
   280.41549178 312.12622841 302.95099493 301.63554807 310.25664839]
  [  1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.        ]]

 [[256.         306.         306.         256.         256.
   296.         296.         256.         281.         300.44444444
   276.         256.         256.         306.         296.
   256.         281.         300.44444444 276.         256.        ]
  [256.         256.         306.         306.         256.
   256.         296.         296.         256.         256.
   256.         256.         281.         281.         276.
   276.         306.         300.44444444 296.         300.44444444]
  [  1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.
     1.           1.           1.           1.           1.        ]]

 [[151.88658989 190.40386238 190.40386238 151.88658989 200.71757344
   238.22887884 238.22887884 200.71757344 169.29790755 217.46292512
   218.06106028 179.01491408 151.88658989 190.40386238 238.22887884
```

```
 200.71757344 169.29790755 217.46292512 218.06106028 179.01491408]
[256.         256.         329.99099203 317.03872944 256.
 256.         312.78295404 304.83098355 256.         256.
 256.         256.         286.51936472 292.99549602 284.39147702
 280.41549178 322.89366007 320.25480344 308.50760568 310.25664839]
[  1.           1.           1.           1.           1.
   1.           1.           1.           1.           1.
   1.           1.           1.           1.           1.
   1.           1.           1.           1.           1.        ]]]
```

[6]:
```
def constructLeastSquare(Omega, Tau, X_, Y_):

#     print(Omega.shape)
#     print(Tau.shape)
#     print(X_.shape)
#     print(Y_.shape)

    x = np.empty([X_.shape[1], X_.shape[0]])

# print(omega for x_, y_, omega in zip(X_, Y_, Omega))
    # print(x_)
    # print(y_)
    # print(omega)

#had to search up how to do compound matricies
#build A for multiple cameras
    # print(X_.T[0][0])
    # print(Omega[0][2])
    # print((Omega[0][2]*X_.T[0][0]))
    # print((Omega[0][2]*X_.T[0][0] - Omega[0][0]))
#     for x_, y_ in zip(X_.T,Y_.T):
#             print(x_)

    #get the jth [u,v,w] normalized image coordinate from Jth Camera and make A
    A = np.array([
                [[Omega[0][2]*x_[0]-Omega[0][0]],
                [Omega[0][2]*y_[0]-Omega[0][1]],
                [Omega[1][2]*x_[1]-Omega[1][0]],
                [Omega[1][2]*y_[1]-Omega[1][1]],
                [Omega[2][2]*x_[2]-Omega[2][0]],
                [Omega[2][2]*y_[2]-Omega[2][1]]
                ]
                 for x_, y_ in zip(X_.T,Y_.T)])

    #reshape A to fit the dimensions properly
    A = A.reshape(20,6,3)
```

```python
    B = np.array([
                [[Tau[0][0]-Tau[0][2]*x_[0]],
                [Tau[0][1]-Tau[0][2]*y_[0]],
                [Tau[1][0]-Tau[1][2]*x_[1]],
                [Tau[1][1]-Tau[1][2]*y_[1]],
                [Tau[2][0]-Tau[2][2]*x_[2]],
                [Tau[2][1]-Tau[2][2]*y_[2]]
                ]
                for x_, y_ in zip(X_.T,Y_.T)])

    # for a_, b_ in zip(A, b):
    #     print(a_.shape)
    #     print(b_.shape)

    for enum, (a_, b_) in enumerate(zip(A,B)):

        x[enum] = (np.linalg.inv(a_.T@a_) @ a_.T @ b_).T


    # print(x)
    return x
```

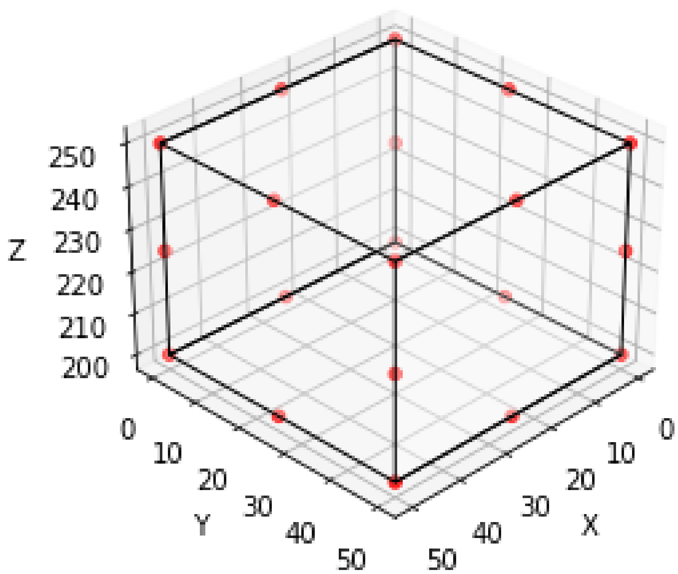
0.0.1 Box estimated coordinate projection

[7]:

```python
W = constructLeastSquare(Omega, Tau, X_, Y_)

print(W)

plot(W, 'red')
```

```
[[1.55431223e-14 0.00000000e+00 2.00000000e+02]
 [5.00000000e+01 0.00000000e+00 2.00000000e+02]
 [5.00000000e+01 5.00000000e+01 2.00000000e+02]
 [1.19904087e-14 5.00000000e+01 2.00000000e+02]
 [4.88498131e-15 0.00000000e+00 2.50000000e+02]
 [5.00000000e+01 0.00000000e+00 2.50000000e+02]
 [5.00000000e+01 5.00000000e+01 2.50000000e+02]
 [1.02140518e-14 5.00000000e+01 2.50000000e+02]
 [2.50000000e+01 0.00000000e+00 2.00000000e+02]
 [5.00000000e+01 0.00000000e+00 2.25000000e+02]
 [2.50000000e+01 0.00000000e+00 2.50000000e+02]
 [8.88178420e-15 0.00000000e+00 2.25000000e+02]
 [1.42108547e-14 2.50000000e+01 2.00000000e+02]
 [5.00000000e+01 2.50000000e+01 2.00000000e+02]
 [5.00000000e+01 2.50000000e+01 2.50000000e+02]
 [8.43769499e-15 2.50000000e+01 2.50000000e+02]
 [2.50000000e+01 5.00000000e+01 2.00000000e+02]
```

```
 [5.00000000e+01 5.00000000e+01 2.25000000e+02]
 [2.50000000e+01 5.00000000e+01 2.50000000e+02]
 [9.76996262e-15 5.00000000e+01 2.25000000e+02]]
```

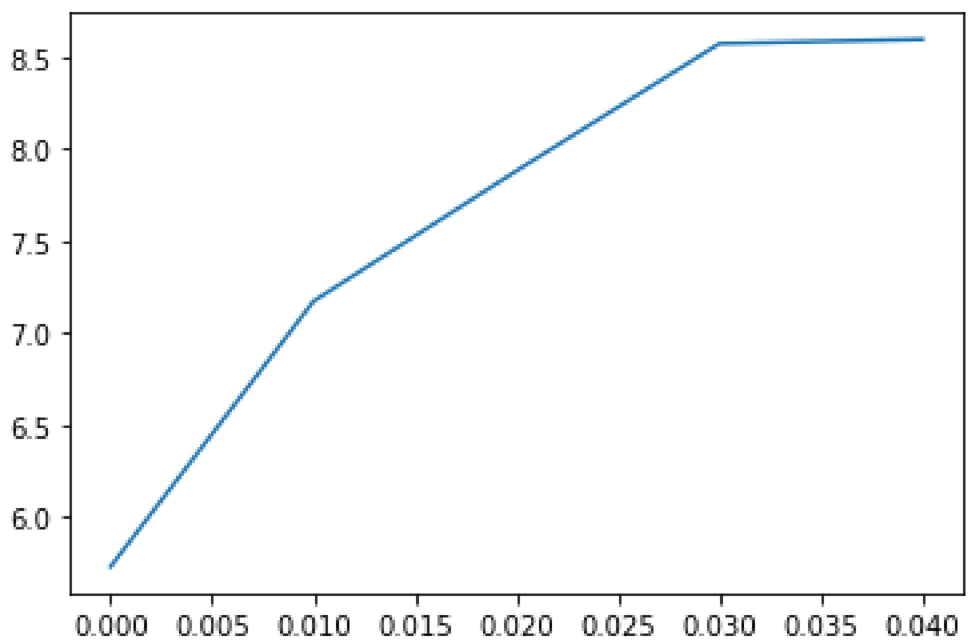


0.0.2 Sensitivity analysis

[8]:
```
def Error(target, estimate):
    return (np.linalg.norm(((target-estimate)**2).sum())).sum()
```

[41]:
```
#increase levels of noise to coordinates
error = np.empty(5)
for i in range(5):
    #code given from the slide
    mu, sigma = 0, (i+1)/100
    xnoise = np.random.normal(mu, sigma, X_.shape)
    ynoise = np.random.normal(mu, sigma, Y_.shape)

    # print(xnoise)
    # print(ynoise)


    x_noise = X_ + xnoise
    y_noise = Y_ + ynoise

    noiseW = constructLeastSquare(Omega, Tau, x_noise, y_noise).T
    # plot(noiseW.T)
```

```
    #make homogenous for camera function
    noiseW = homogenous(noiseW)

    #run points Wj through cameras to generate Xij
    # print(Lambda.shape, omega1.shape, tau1.shape, noiseW.shape)
    cam1 = pinhole(noiseW, Lambda, omega1, tau1).T
    cam2 = pinhole(noiseW, Lambda, omega2, tau2).T
    cam3 = pinhole(noiseW, Lambda, omega3, tau3).T


    error1 = Error(cam1, X1)
    error2 = Error(cam2, X2)
    error3 = Error(cam3, X3)

    error[i] = error1 + error2 + error3


xaxis = np.arange(0,5)/100
plt.plot(xaxis, np.log(error))
plt.show()
# print(error)

# plot(noisebox)
```



```
[ ]:
```