In [201…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
```

The Camera Rig

Instrisic Parameters

- I created a function to compute the image points using the instrinsic parameters of the camera

In [202…

```
def image_points(w):
    skew = 0
    delta = np.array([256, 256])
    phi = np.array([200,200])
    x = (phi[0]*w[0] + skew*w[1])/w[2] + delta[0]
    y = (phi[1]*w[1])/w[2] + delta[1]
    return x, y

def rotation_translation(w, R, T):
    w_prime = R@w + T
    return w_prime
```

Extrinsic Parameters

- I created an Ω matrix and τ vector for the three cameras

In [203…

```
angle_1 = 0*np.pi/180
omega_1 = np.array([[np.cos(angle_1),0,np.sin(angle_1)],[0, 1, 0],[np.sin(angle_1), 0, np.cos(angle_1)]])
T_1 = np.array([[0],[0],[0]])

angle_2 = 35*np.pi/180
omega_2 = np.array([[np.cos(angle_2),0,-np.sin(angle_2)],[0, 1, 0],[np.sin(angle_2), 0, np.cos(angle_2)]])
T_2 = np.array([[200],[0],[0]])

angle_3 = (-35)*np.pi/180
omega_3 = np.array([[np.cos(angle_3),0,-np.sin(angle_3)],[0, 1, 0],[np.sin(angle_3), 0, np.cos(angle_3)]])
T_3 = np.array([[-200],[0],[0]])
```

Measurements

Computing the synthetic image points

1. I transformed all the cube points using the extrinsic parameter for each cameras.
2. I computed the image points for the three cameras using the three new coordinate 3D points.

In [204…

```
CC = [[0,0,200],[25,0,200],[50,0,200],[50,25,200],[50,50,200],[25,50,200],[0,50,200],[0,25,200],[0,0,225],[50,0,2
C = np.zeros((20,3))
for i in range(len(CC)):
    c = np.vstack(CC[i])
    C[i,:] = c.T
```

In [295…

```
N = 3
X = np.zeros((N,len(C)))
Y = np.zeros((N,len(C)))

CP_1 = np.zeros((len(C),3))
CP_2 = np.zeros((len(C),3))
CP_3 = np.zeros((len(C),3))

for i in range(len(C)):
    c = np.vstack(C[i])
    cp_1 = rotation_translation(c, omega_1, T_1)
    cp_2 = rotation_translation(c, omega_2, T_2)
    cp_3 = rotation_translation(c, omega_3, T_3)
    CP_1[i,:] = np.vstack(cp_1).T
    CP_2[i,:] = np.vstack(cp_2).T
    CP_3[i,:] = np.vstack(cp_3).T

    x_1,y_1 = image_points(cp_1)
    x_2,y_2 = image_points(cp_2)
    x_3,y_3 = image_points(cp_3)

    X[0,i] = x_1
    Y[0,i] = y_1
    X[1,i] = x_2
```

```
    Y[1,i] = y_2
    X[2,i] = x_3
    Y[2,i] = y_3
```

Noise Addition

1. Created a vector of a random normal signal with parameter mu and sigma.
2. Added the random noise to the image points

In [296…

```
mu, sigma = 0, 2
noise = np.random.normal(mu, sigma, [2,20])

for i in range(N):
    X[i] = X[i] + noise[0]
    Y[i] = Y[i] + noise[1]
```

Estimation Method

Coordinate Normalization

- Multiplied the augmented x and y vectors by the inverse of the Λ matrix to get the normalized image coordinates

$$\Lambda = \begin{bmatrix} 200 & 0 & 256 \\ 0 & 200 & 256 \\ 0 & 0 & 1 \end{bmatrix}$$

In [297…

```
lambda_ = np.array(([200, 0, 256],[0, 200, 256], [0, 0, 1]))
lambda_inverse = np.linalg.inv(lambda_)

X_norm = np.zeros((N,len(C)))
Y_norm = np.zeros((N,len(C)))
Z_norm = np.zeros((N,len(C)))

for i in range(N):
    for j in range(len(C)):
        x_aug = np.vstack([X[i,j], Y[i,j], 1])
        x_norm = lambda_inverse@x_aug
        X_norm[i,j] = x_norm[0]
        Y_norm[i,j] = x_norm[1]
        Z_norm[i,j] = x_norm[2]
```

Solving Linear system of Equations

1. I solved the equation: $$\begin{bmatrix} \omega_{31j}x'_j - \omega_{11j} & \omega_{32j}x'_j - \omega_{12j} & \omega_{33j}x'_j - \omega_{13j} \\ \omega_{31j}y'_j - \omega_{21j} & \omega_{32j}y'_j - \omega_{22j} & \omega_{33j}y'_j - \omega_{23j} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \tau_{xj} - \tau_{zj}x'_j \\ \tau_{yj} - \tau_{zj}y'_j \end{bmatrix}$$ Using the solution for linear system of equations of the form: Ax = b
2. I created an A matrix for each camera in the form: $$\begin{bmatrix} \omega_{31j}x'_j - \omega_{11j} & \omega_{32j}x'_j - \omega_{12j} & \omega_{33j}x'_j - \omega_{13j} \\ \omega_{31j}y'_j - \omega_{21j} & \omega_{32j}y'_j - \omega_{22j} & \omega_{33j}y'_j - \omega_{23j} \end{bmatrix}$$
3. Found the **w** points using the following equation for each camera **A** matrix: $\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.
4. Using the reconstructed **w**, I computed the image points **x** using the instrinsic and extrinsic parameters for each camera

In [298…

```
A = []
b = []

for i in range(len(C)):
    A_1 = np.array(([(omega_1[2,0]*X_norm[0,i]-omega_1[0,0]), (omega_1[2,1]*X_norm[0,i]-omega_1[0,1]), (omega_1[2
                     [(omega_1[2,0]*Y_norm[0,i]-omega_1[1,0]), (omega_1[2,1]*Y_norm[0,i]-omega_1[1,1]), (omega_1[2,2]

    A_2 = np.array(([(omega_2[2,0]*X_norm[1,i]-omega_2[0,0]), (omega_2[2,1]*X_norm[1,i]-omega_2[0,1]), (omega_2[2
                     [(omega_2[2,0]*Y_norm[1,i]-omega_2[1,0]), (omega_2[2,1]*Y_norm[1,i]-omega_2[1,1]), (omega_2[2,2]

    A_3 = np.array(([(omega_3[2,0]*X_norm[2,i]-omega_3[0,0]), (omega_3[2,1]*X_norm[2,i]-omega_3[0,1]), (omega_3[2
                     [(omega_3[2,0]*Y_norm[2,i]-omega_3[1,0]), (omega_3[2,1]*Y_norm[2,i]-omega_3[1,1]), (omega_3[2,2]

    b_1 = np.array((T_1[0]-T_1[2]*X_norm[0,i],T_1[1]-T_1[2]*Y_norm[0,i]))
    b_2 = np.array((T_2[0]-T_2[2]*X_norm[1,i],T_2[1]-T_2[2]*Y_norm[1,i]))
    b_3 = np.array((T_3[0]-T_3[2]*X_norm[2,i],T_3[1]-T_3[2]*Y_norm[2,i]))
```

```
    A.append(np.vstack((A_1,A_2,A_3)))
    b.append(np.vstack((b_1,b_2,b_3)))
```

In [299…

```
W = np.zeros((len(C),3))
for i in range(len(A)):
    A_ = A[i]
    b_ = b[i]
    w = (np.linalg.inv(A_.T@A_))@A_.T@b_
    W[i,:] = w.T
```

In [300…

```
X_hat = np.zeros((N,len(C)))
Y_hat  = np.zeros((N,len(C)))

for i in range(len(C)):
    w = np.vstack(W[i])
    Wp_1 = w
    Wp_2 = rotation_translation(w, omega_2, T_2)
    Wp_3 = rotation_translation(w, omega_3, T_3)

    x_1,y_1 = image_points(Wp_1)
    x_2,y_2 = image_points(Wp_2)
    x_3,y_3 = image_points(Wp_3)

    X_hat[0,i] = x_1
    Y_hat[0,i] = y_1
    X_hat[1,i] = x_2
    Y_hat[1,i] = y_2
    X_hat[2,i] = x_3
    Y_hat[2,i] = y_3
```

Sensitivity Analysis

Re-projection error

- I found the re-projection error between the orignal **X** and **Y** computed using the intrinsic and extrinsic parameters and the original 3D points, and the re-projected **X** and **Y** using the estimation method.
- I repeated the whole estimation method after adding different levels of noise and computed the re-projection error.

In [210…

```
E_ =  []
```

In [301…

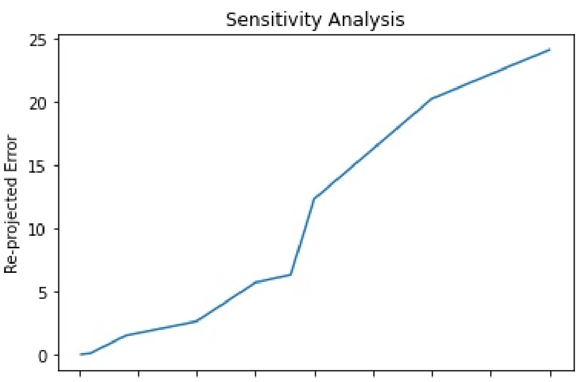
```
E = 0
for j in range(N):
    for i in range(len(C)):
        x1 = np.vstack((X_hat[j,i],Y_hat[j,i]))
        x2 = np.vstack((X[j,i],Y[j,i]))
        e = distance.euclidean(x1, x2)
        E = E + e
E_.append(E)
```

In [312…

```
s = [0.01, 0.05, 0.1, 0.2, 0.5, 0.75, 0.9, 1, 1.5, 2]
plt.plot(s, E_)
plt.title('Sensitivity Analysis')
plt.xlabel('Sigma')
plt.ylabel('Re-projected Error')
```

Out[312…

```
Text(0, 0.5, 'Re-projected Error')
```



Sigma	Error
0	1.02e-12
0.01	0.100321
0.05	0.579971
0.1	1.492708
0.2	2.620154
0.5	5.678475
0.75	6.288986
0.9	6.288986
1	12.28457
1.5	20.20812
2	24.08086