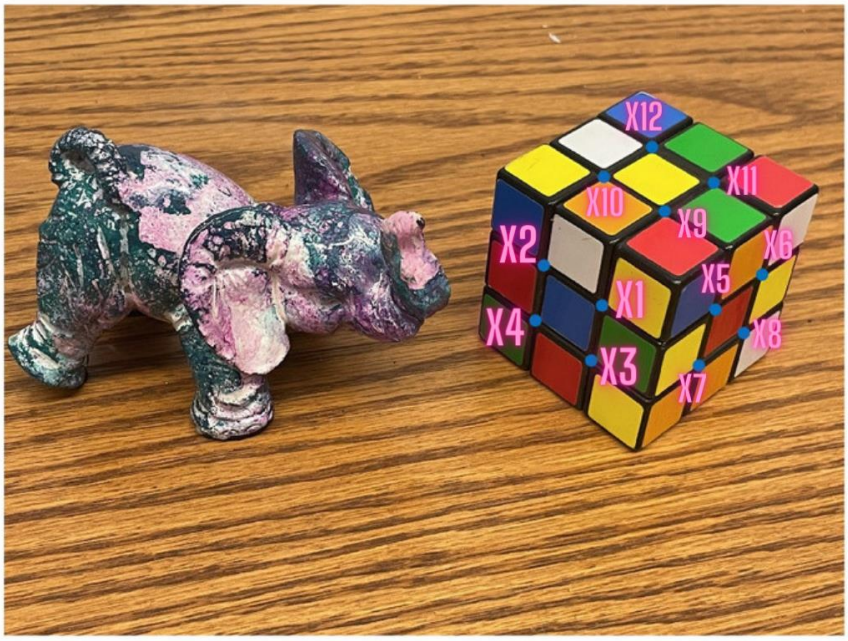
Q2) For this part, I selected some points to work with and find matrix P. I selected 12 points to make sure that the problem is solvable. In the following image, you can see the selected points in blue.

I considered that the rubik's cube dimension is 18x18x18 mm.



These are the world coordinates for selected points in the above image.

	X	Y	Z
X1	0	6	6
X2	0	6	12
X3	0	12	6
X4	0	12	12
X5	6	6	0
X6	12	6	0
X7	6	12	0
X8	12	12	0
X9	6	0	6
X10	6	0	12
X11	12	0	6
X12	12	0	12

Using the getCursorInfo and impixel functions in matlab, I found the corrseponding points in the pixel coordinates. I commented that part of the code that gets pixel values.

The corresponding pixel coordinates is as below:

	x	y
X1	2881	1429
X2	2602	1248
X3	2823	1701
X4	2560	1510
X5	3392	1493
X6	3612	1336
X7	3314	1763
X8	3543	1586
X9	3167	1010
X10	2869	826
X11	3402	857
X12	3105	698

In order to normalize world coordinate points, I first computed the mean of each component. i.e. X, Y, Z.

Then using the variance formula, I normalized the 12 points, by subtracting from mean and dividing by the variance.

$$(\bar{X}, \bar{Y}, \bar{Z}) = \frac{1}{N}\sum_{i=1}^{N}(X_i, Y_i, Z_i)$$

$$\sigma_2^2 = \frac{1}{3N}\sum_{i=1}^{N}(X_i - \bar{X})^2 + (Y_i - \bar{Y})^2 + (Z_i - \bar{Z})^2$$

$$(X_i, Y_i, Z_i) \rightarrow (\frac{X_i - \bar{X}}{\sigma_2}, \frac{Y_i - \bar{Y}}{\sigma_2}, \frac{Z_i - \bar{Z}}{\sigma_2}).$$

I used the same strategy of normalizing for the pixel coordinates. Now that we have normalized points in world coordinates and in image coordinates, we can compute P_{norm}.

I used the formula below to construct the matrix A.

Matrix A

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N & -x_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N & -y_N \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \\ \vdots \\ P_{31} \\ P_{32} \\ P_{33} \\ P_{34} \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In a for loop over each sample point, I select the corresponding points in the world and pixel coordinates and compute the corresponding two rows in the matrix A to that sample point.

This is the total least square problem. In order to find the solution of this problem (find vector P), we can use the SVD function to decomposite matrix A. By finding the min singular value in matrix Σ and then select the corresponding eigenvector to that sigular value. This vector would be the solution.

The result is a vector that should be reshaped to 3x4 matrix, and this is the normalized P.

In order to complete the computation of matrix P, we need to have this matrix in terms of the original data. Using the variance and average that we calculated in the normalization part, we can now compute matrices M1 and M2.

For pixel coordinates:

$$\underbrace{\begin{bmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\bar{x} \\ 0 & 1 & -\bar{y} \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{M_1}}$$

For world coordinates:

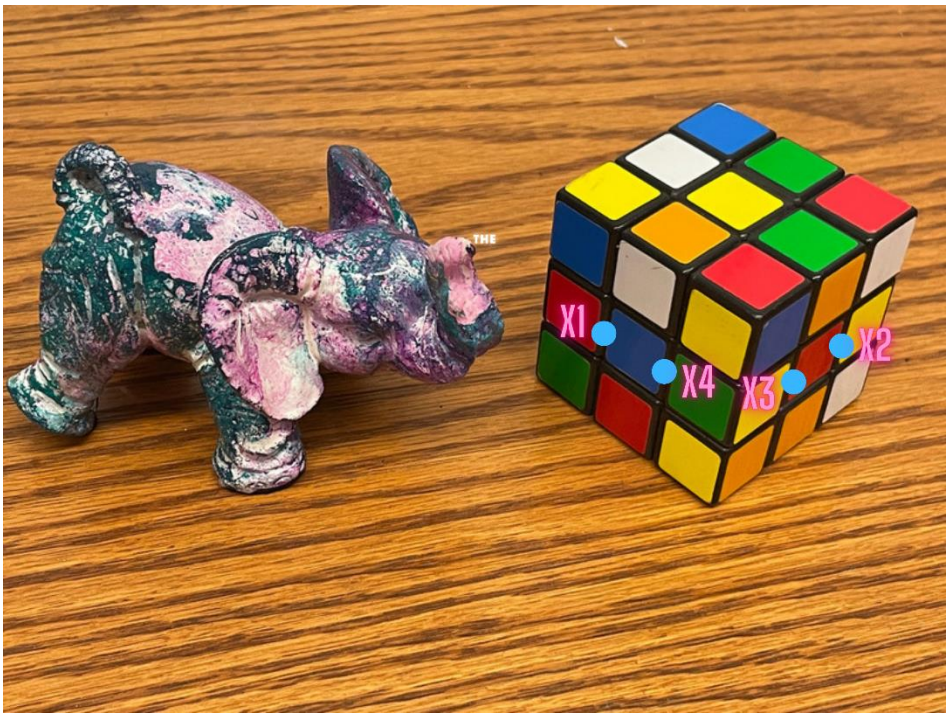
$$\underbrace{\begin{bmatrix} 1/\sigma_2 & 0 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 & 0 \\ 0 & 0 & 1/\sigma_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\bar{X} \\ 0 & 0 & 0 & -\bar{Y} \\ 0 & 0 & 1 & -\bar{Z} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{M_2}}$$

Matrix P would be:

$$P = M_1^{-1} * P_{norm} * M_2$$

We estimated matrix P. To normalize this matrix, I also divided all the elements by the norm of the third row in this matrix. In this case, the third row would have be a normal vector. Now, we need to pick some other points in the pixel and world coordinates to test our estimation. I picked 4 points.

Test points are blue points in this image:



	X	Y	Z
X1	0	6	8
X2	8	6	0
X3	3	6	0
X4	0	6	3

And the true values in the pixel coordinates are:

	x	y
X1	2720	1332
X2	3524	1378
X3	3271	1579
X4	3014	1551

By adding 1 as the last element to the end of the datapoints, we'll have homogenous 3d coordinates, we multiply matrix P to this world coordinates (4x3), with 3 data samples. The result is a 3x3 matrix corresponding to image coordinates for these 3 points. We divide the first and the second rows by the third row, as the result is in this form:

$$\begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i \end{bmatrix}$$

The result is as following:

	x	y
X1	2941	1348
X2	3524	1365
X3	3170	1369
X4	3035	1361

As requested in the question, the mean and std of the error between estimation and true image coordinate is as follows (the error is element-wise subtraction):

Error (rounded):

	x	y
X1	221	18
X2	0	13
X3	101	210
X4	21	190

- Mean of errors: 96.3365
- Std of errors: 96.6358

Q3) In this question, I did the same job as question 2 to obtain the matrix P.

In order to compute the rotation matrix, I computed three rotational submatrices. In the following, P mean just the first three columns of the estimated projection matrix.

R_{z_θ} :

```
R_z_theta = [
    cos(theta), sin(theta), 0;
    -sin(theta), cos(theta), 0;
    0, 0, 1;
];
```

Where: $\theta = \arctan(\frac{P_{31}}{P_{32}})$

R_{x_β} :

```
R_x_beta = [
    1, 0, 0;
    0, cos(Beta), sin(Beta);
    0, -sin(Beta), cos(Beta);
];
```

Where: $\beta = \arctan(\frac{PR_{z_\theta 32}}{PR_{z_\theta 33}})$

R_{z_γ} :

```
R_z_gamma = [
    cos(gamma), sin(gamma), 0;
    -sin(gamma), cos(gamma), 0;
    0, 0, 1;
];
```

Where: $\gamma = \arctan(\frac{PR_{z_\theta} R_{x_\beta\ 21}}{PR_{z_\theta} R_{x_\beta\ 22}})$

The only thing that should be considered is if $PR_{z_\theta}R_{x_\beta}R_{z_\gamma}$ has negative diagonals. Here is the result, which does not have negative diagonals.

```
>> Q3
   1.0e+03 *

    0.1987   -0.0853    2.9310
    0.0000    0.1955    1.2160
    0.0000    0.0000    0.0010
```

Matrix R is the transpose of $R_{z_\theta} R_{x_\beta} R_{z_\gamma}$. Here is the rotation matrix:

```
 ~
  0.0751    0.1093   -0.9912
 -0.6234   -0.7706   -0.1322
 -0.7783    0.6278    0.0103
```

And its inverse is this:

```
  0.0751   -0.6234   -0.7783
  0.1093   -0.7706    0.6278
 -0.9912   -0.1322    0.0103
```

So, $R^{-1} = R^T$.

The determinent of R is 1, in my experiments.

In order to find the translation vector, I got an idea to find that independantly of matrix R, so it is only dependant on estimation of P.

If we call the first three columns of P as P', Then, we will have P' = KR, according to lecture notes. We can use this in the finite projection camera model, which is:

$$P = KR\,[I\,|-c] \qquad \rightarrow \qquad P = P'\,[I|-c]$$

So, we can conclude that:

$$Inv(P') * P = [I|-c]$$

By doing that, I reached a matrix s, the first three columns are expected to be identity, which is the same in my expremints:

```
  1.0000   -0.0000    0.0000  -11.6139
  0.0000    1.0000    0.0000   -4.5666
  0.0000         0    1.0000  -12.1243
```

To get the translation vector, the negative of forth column was picked.

```
11.6139
 4.5666
12.1243
```

The norm of this vector is: **17.3993.**

There's another way to calculate matrix R to be dependant on K.

$$P' = KR \rightarrow R = INV(K) * P'$$

The result is:

```
-0.2082    0.2145   -0.0575
 0.0323   -0.1077   -0.0094
-0.7783    0.6278    0.0103
```

The determinent is not 1, and the plot of that was not like a valid rotation matrix. So, I use the other version of R in my code, and I commented this version.

I picked the same points for image 2 and image 3, the ones that I described in Q2, and I got the corresponding pixel coordinates in each image individually.

Pixel coordinates in image 2:

	x	y
X1	2684	1426
X2	2480	1214
X3	2652	1706
X4	2468	1486
X5	3240	1542
X6	3560	1402
X7	3184	1806
X8	3492	1698
X9	3048	1002
X10	2832	794
X11	3380	894
X12	3132	682

Pixel coordinates in image 3:

	x	y
X1	2380	1476
X2	2284	1239
X3	2372	1770
X4	2272	1530
X5	2880	1649
X6	3268	1587
X7	2848	1934
X8	3212	1868
X9	2772	1102
X10	2644	890
X11	3148	1038
X12	3016	830

I did the same procedure for these 2 images as described for the first image. Here are some information about them.

Image 2, rotation matrix:

```
 0.1054    0.1233   -0.9868
-0.7353   -0.6584   -0.1608
-0.6695    0.7425    0.0212
```

Image 2, translation vector:

```
13.3934
 4.0012
16.1126
```

Image 3, rotation matrix:

```
 0.2920    0.1778   -0.9397
-0.8681   -0.3631   -0.3384
-0.4014    0.9146    0.0483
```

Image 3, translation vector:

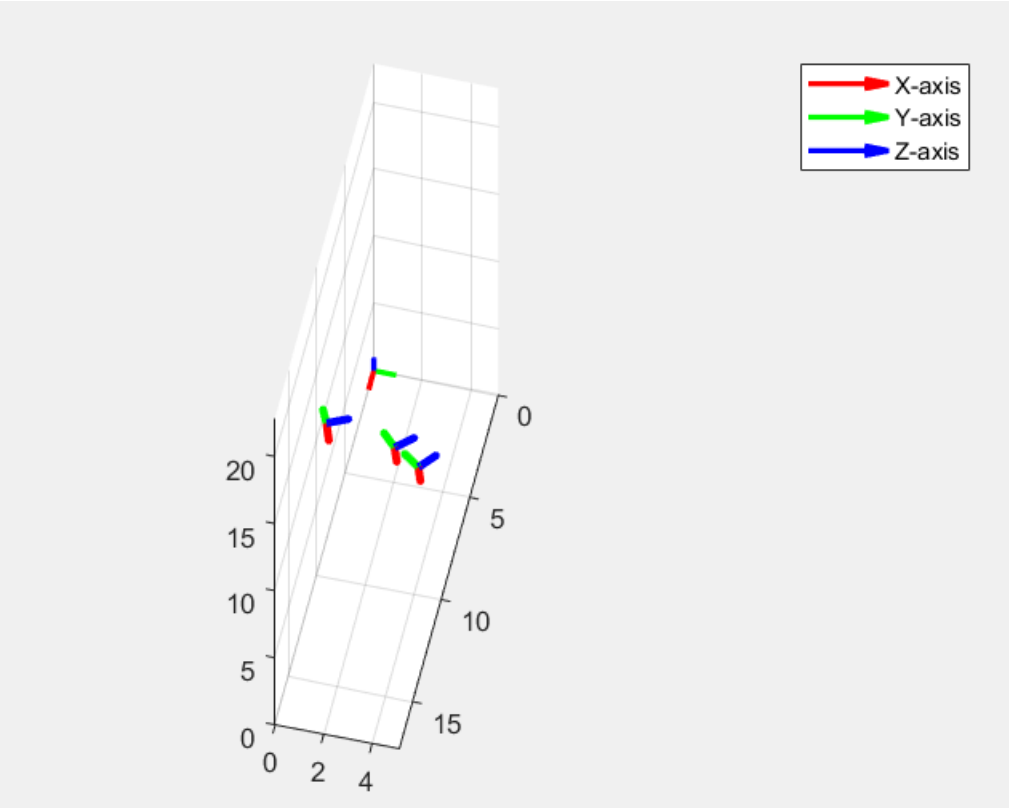
```
17.0740
 2.0519
22.8316
```

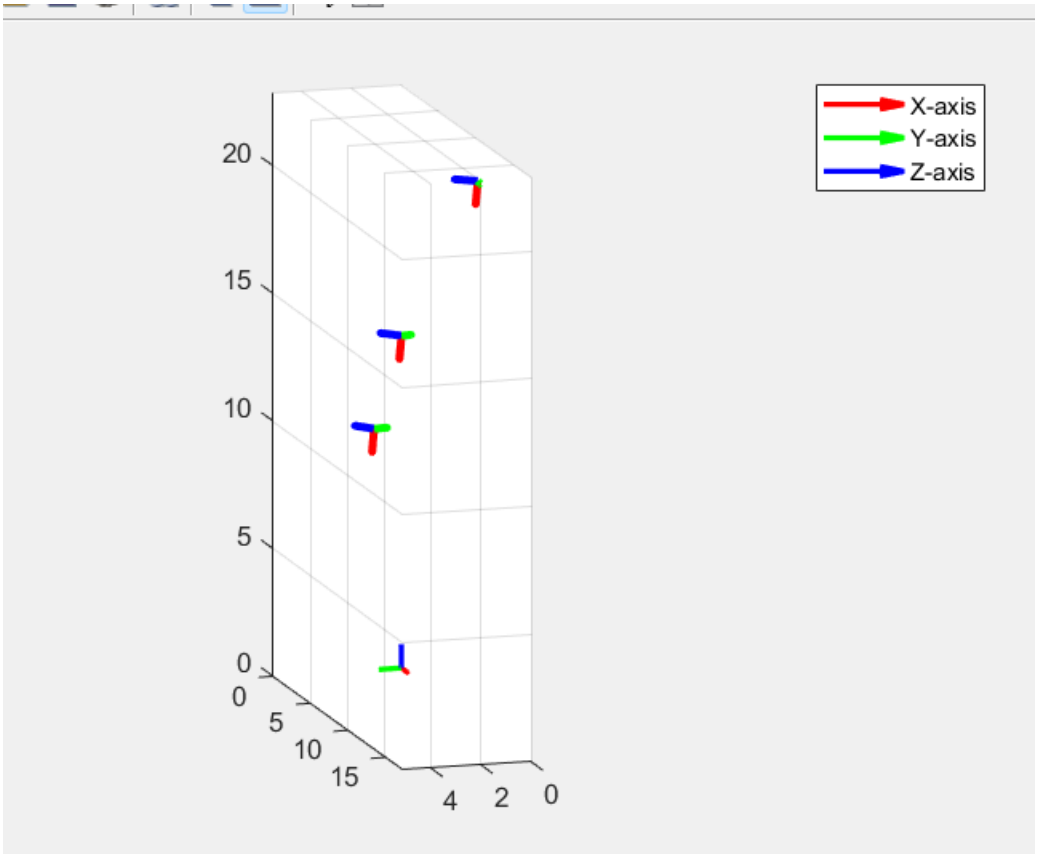
In order to plot the results, I selected the origin as [0, 0, 0] and the following axes:

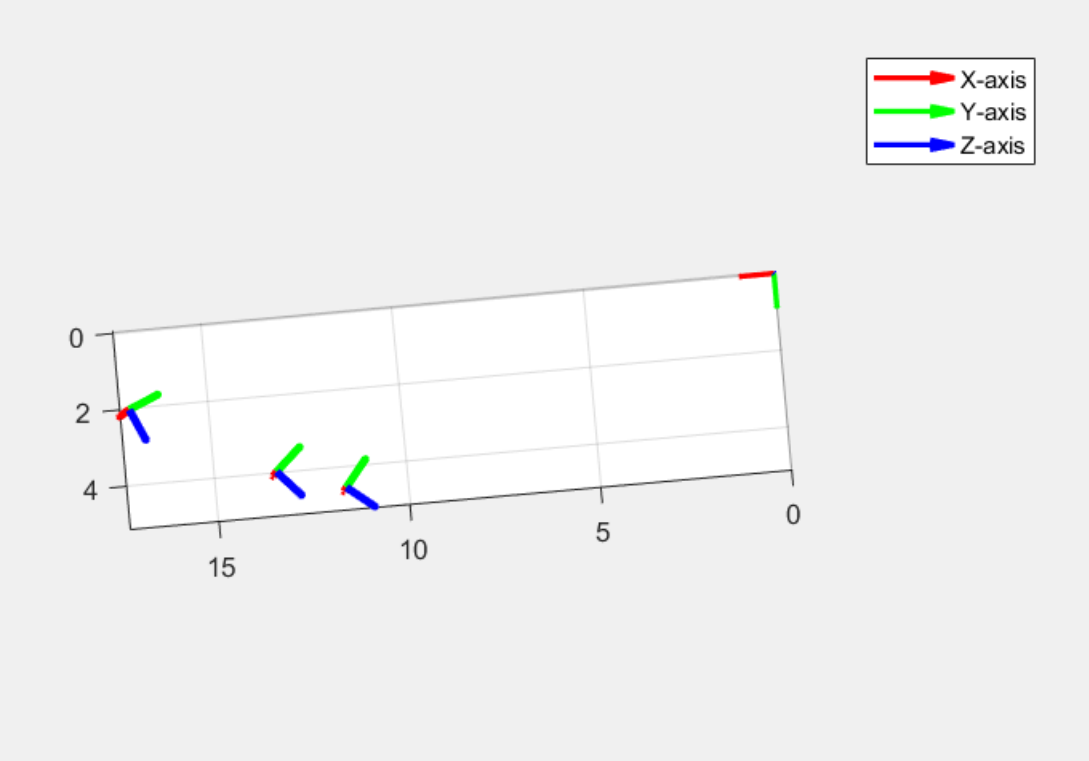
```
x_axis = [1, 0, 0];
y_axis = [0, 1, 0];
z_axis = [0, 0, 1];
```

The translation vector gives us the origin of each camera coordinate, and each row of the rotation matrix give us the axes in the camera coordinate.

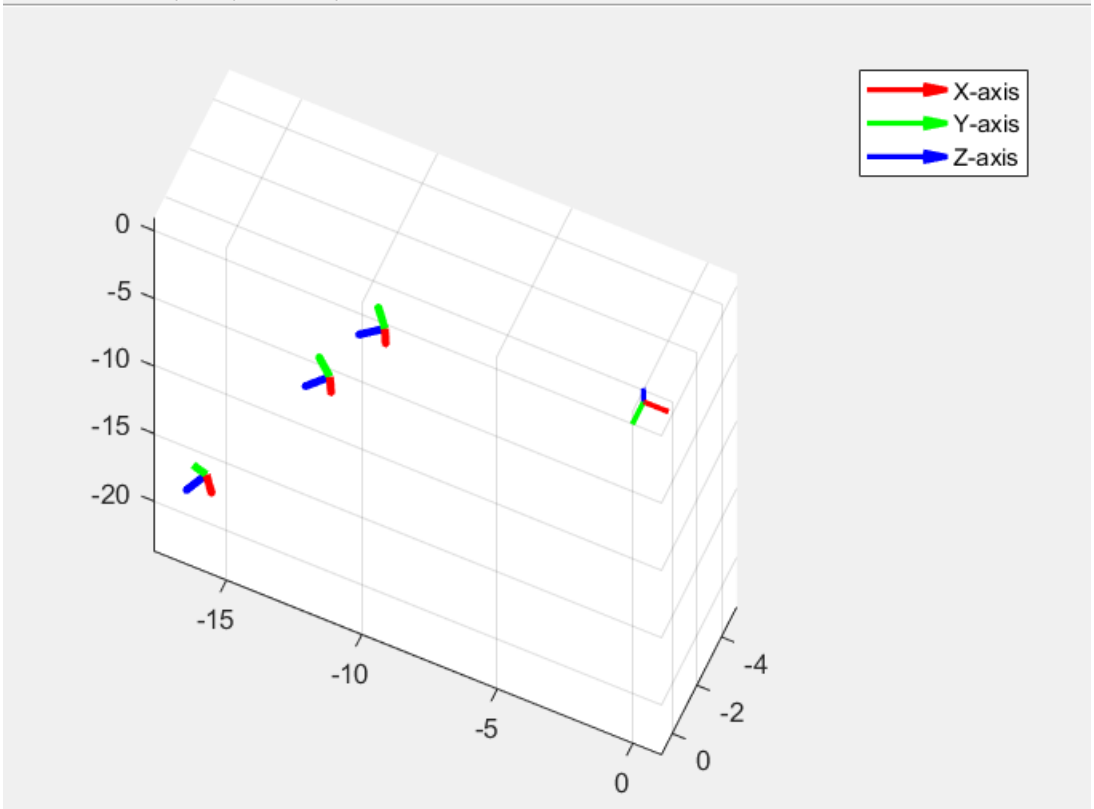
The result is as follows:




X-axis
Y-axis
Z-axis
20
15
10
5
0
0
5
10
15
4
2
0


X-axis
Y-axis
Z-axis
0
2
4
15
10
5
0

The plot shows that the position of three cameras with respect to the world origin point and axes. The images were taken in a sequence, meaning that the second camera should be in the right side of the first one and so on, which is the clearly shown in this plot (I used -c here as the translation vector):



End!