Camera Calibration

In [2]:

```
import numpy as np
```

In [3]:

```
xw=[0, 21.68, 43.36, 65.04, 86.72, 108.4, 130.08]
yw=[21.68, 43.36, 65.04, 86.72, 108.40, 130.08, 0]
zw=[713.74]*7
xc = [366.9331, 416.9176, 466.7643, 516.8607, 566.9595, 617.385, 678.7745]
yc = [240.6366, 293.9554, 346.9755, 400.1464, 453.5715, 507.2507, 195.8701]
```

- Computing xwxc, ywxc, zwxc, xwyc, ywyc, and zwyc

In [4]:

```
xwxc=-np.multiply(np.array(xw),np.array(xc))
ywxc =-np.multiply(np.array(yw),np.array(xc))
zwxc=-np.multiply(np.array(zw),np.array(xc))
xwyc=-np.multiply(np.array(xw),np.array(yc))
ywyc=-np.multiply(np.array(yw),np.array(yc))
zwyc=-np.multiply(np.array(zw),np.array(yc))
```

In [5]:

```
xwxc
```

Out[5]:

```
array([    -0.      ,  -9038.773568, -20238.900048, -33616.619928,
       -49166.72784 , -66924.534   , -88294.98696 ])
```

Solve this -

$\begin{bmatrix} u^{i} \\ v^{i} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_{w}^{i} \\ y_{w}^{i} \\ z_{w}^{i} \\ 1 \end{bmatrix}$

Considering A.p=0

$\begin{bmatrix}xw^{(i)}&yw^{(i)}&zw^{(i)}&1&0& 0 & 0 & 0 & -xc^{(i)}xw^{(i)}&-xc^{(i)}yw^{(i)}&-xc^{(i)}zw^{(i)} &-xc^{(i)}\\ 0&0&0&0&xw^{(i)}&yw^{(i)}&zw^{(i)}&1&-yc^{(i)}xw^{(i)}&-yc^{(i)}yw^{(i)}& -yc^{(i)}zw^{(i)} &-yc^{(i)} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} $

Now we solve it for almost 7 points which gives us 14 equations for 12 unknowns:

$\begin{bmatrix} xw^{(1)} & yw^{(1)} & zw^{(1)} &1 & 0 & 0 & 0 & 0 & -xc^{(1)}xw^{(1)} & -xc^{(1)}yw^{(1)} & -xc^{(1)}zw^{(1)} & -xc^{(1)} \\ 0 & 0 & 0 & 0 & xw^{(1)} & yw^{(1)} & zw^{(1)}& 1 & -yc^{(1)}xw^{(1)} & -yc^{(1)}yw^{(1)} & -yc^{(1)}zw^{(1)} &-yc^{(1)} \\ ... & ... & ... & ... & ... & ...& ... & ... & ... & ... & ... & ...\\ xw^{(7)} & yw^{(7)} & zw^{(7)} &1 & 0 & 0 & 0 & 0 & -xc^{(7)}xw^{(7)} & -xc^{(7)}yw^{(7)} & -xc^{(7)}zw^{(7)} & -xc^{(7)} \\ 0 & 0 & 0 & 0 & xw^{(7)} & yw^{(7)} & zw^{(7)}& 1 & -yc^{(7)}xw^{(7)} & -yc^{(7)}yw^{(7)} & -yc^{(7)}zw^{(7)} &-yc^{(7)}\end{bmatrix}\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} =\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} $

we then find the rank of the matrix and take the least eigen values and then the linear combination of their eigenvectors are your projection matrix on reshaping

In [6]:

```
A=[]
for i in range(0,7):
    mat1 = [xw[i],yw[i],zw[i],1,0,0,0,0,xwxc[i],ywxc[i],zwxc[i],-xc[i]]
    mat2 = [0,0,0,0,xw[i],yw[i],zw[i],1,xwyc[i],ywyc[i],zwyc[i],-yc[i]]
    A.append(mat1)
    A.append(mat2)
```

In [7]:

```
A = np.array(A)
```

In [8]:

```
A
```

Out[8]:

```
array([[ 0.00000000e+00,  2.16800000e+01,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -0.00000000e+00,
        -7.95510961e+03, -2.61894831e+05, -3.66933100e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  2.16800000e+01,
         7.13740000e+02,  1.00000000e+00, -0.00000000e+00,
        -5.21700149e+03, -1.71751967e+05, -2.40636600e+02],
       [ 2.16800000e+01,  4.33600000e+01,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -9.03877357e+03,
        -1.80775471e+04, -2.97570768e+05, -4.16917600e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  2.16800000e+01,  4.33600000e+01,
         7.13740000e+02,  1.00000000e+00, -6.37295307e+03,
        -1.27459061e+04, -2.09807727e+05, -2.93955400e+02],
       [ 4.33600000e+01,  6.50400000e+01,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -2.02389000e+04,
        -3.03583501e+04, -3.33148351e+05, -4.66764300e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  4.33600000e+01,  6.50400000e+01,
         7.13740000e+02,  1.00000000e+00, -1.50448577e+04,
        -2.25672865e+04, -2.47650293e+05, -3.46975500e+02],
       [ 6.50400000e+01,  8.67200000e+01,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -3.36166199e+04,
        -4.48221599e+04, -3.68904156e+05, -5.16860700e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  6.50400000e+01,  8.67200000e+01,
         7.13740000e+02,  1.00000000e+00, -2.60255219e+04,
        -3.47006958e+04, -2.85600492e+05, -4.00146400e+02],
       [ 8.67200000e+01,  1.08400000e+02,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -4.91667278e+04,
        -6.14584098e+04, -4.04661674e+05, -5.66959500e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  8.67200000e+01,  1.08400000e+02,
         7.13740000e+02,  1.00000000e+00, -3.93337205e+04,
        -4.91671506e+04, -3.23732122e+05, -4.53571500e+02],
       [ 1.08400000e+02,  1.30080000e+02,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -6.69245340e+04,
        -8.03094408e+04, -4.40652370e+05, -6.17385000e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  1.08400000e+02,  1.30080000e+02,
         7.13740000e+02,  1.00000000e+00, -5.49859759e+04,
        -6.59831711e+04, -3.62045115e+05, -5.07250700e+02],
       [ 1.30080000e+02,  0.00000000e+00,  7.13740000e+02,
         1.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -8.82949870e+04,
        -0.00000000e+00, -4.84468512e+05, -6.78774500e+02],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  1.30080000e+02,  0.00000000e+00,
         7.13740000e+02,  1.00000000e+00, -2.54787826e+04,
        -0.00000000e+00, -1.39800325e+05, -1.95870100e+02]])
```

In [10]:

```
inner_product = np.dot(A.T,A)
```

In [11]:

```
w, v = np.linalg.eig(inner_product)
```

In [12]:

```
w
```

Out[12]:

```
array([ 1.50284271e+12,  4.25117817e+09,  6.30241419e+09,  3.56520226e+06,
        7.36166840e+04,  1.57364399e+04,  9.82206104e+02,  7.79910325e-03,
       -5.23177574e-07,  1.24812733e-08,  1.32458895e-09, -1.11540485e-14])
```

In [18]:

```
w/w[0]
```

Out[18]:

```
array([ 1.00000000e+00,  2.82875789e-03,  4.19366188e-03,  2.37230566e-06,
        4.89849560e-08,  1.04711157e-08,  6.53565471e-10,  5.18956721e-15,
       -3.48125303e-19,  8.30510951e-21,  8.81388945e-22, -7.42196665e-27])
```

Taking last 6 eigen values as the rank of the matrix

In [19]:

```
v
```

Out[19]:

```
array([[ 1.28689505e-04, -1.22744911e-03, -3.58834453e-06,
         4.76238488e-02,  3.76191977e-02, -4.50284414e-01,
        -3.15236715e-01,  4.82545225e-01,  6.71782300e-01,
        -1.09728426e-02, -3.45233066e-04, -5.66171665e-05],
       [ 1.17088595e-04,  5.02741245e-04, -8.48073694e-04,
         5.64151780e-02,  1.96640621e-01, -3.61949215e-01,
        -5.58219096e-01,  2.32750788e-01, -6.71797916e-01,
         1.09723450e-02,  3.45252368e-04,  5.66174995e-05],
       [ 1.24246883e-03,  2.04748045e-03,  3.71147599e-03,
         5.55189348e-01,  7.56493500e-01,  1.19382272e-01,
         2.84481005e-01,  1.54446844e-01,  2.03545604e-02,
        -1.72587631e-03, -2.37069565e-06, -1.51990053e-06],
       [ 1.74078632e-06,  2.86866430e-06,  5.20003921e-06,
         7.77859372e-04,  1.05990066e-03,  1.67262964e-04,
         3.98577993e-04,  2.16170788e-04,  3.41981580e-02,
         9.93866305e-01, -5.78990009e-03, -1.42599906e-04],
       [ 8.06549781e-05, -2.76456390e-04, -4.44938183e-04,
        -7.57600734e-02,  1.98270572e-01,  6.97095049e-01,
        -6.53422371e-01, -6.02454692e-02,  1.93856756e-01,
        -3.16623203e-03, -9.96273273e-05, -1.63377869e-05],
       [ 9.17854146e-05,  3.43974789e-04, -7.24148533e-04,
        -7.27211121e-02, -2.57054215e-01,  4.01412419e-01,
         2.23903971e-01,  8.23980833e-01, -1.93873429e-01,
         3.16570065e-03,  9.96479363e-05,  1.63381424e-05],
       [ 8.35173533e-04,  4.58296916e-03, -9.91493210e-05,
        -8.21749570e-01,  5.31244518e-01, -7.00786742e-02,
         1.76033832e-01,  8.09251539e-02,  5.88687326e-03,
        -9.53503766e-05, -4.52262302e-06, -1.40154639e-03],
       [ 1.17013693e-06,  6.42106252e-06, -1.38915181e-07,
        -1.15132901e-03,  7.44310979e-04, -9.81851573e-05,
         2.46635795e-04,  1.13378116e-04, -1.71913226e-04,
        -6.29630742e-04,  1.06965044e-03,  9.99985545e-01],
       [-1.09871067e-01,  9.59954470e-01,  2.57658712e-01,
         1.99545093e-03, -4.27512772e-03, -2.61361370e-04,
        -2.06618248e-03, -3.92806209e-04,  9.89722883e-04,
        -1.61648992e-05, -5.08642093e-07, -8.34113764e-08],
       [-1.03617465e-01, -2.68881545e-01,  9.57577199e-01,
        -2.80374236e-03, -1.37991432e-03, -7.81597759e-05,
        -1.06295726e-03,  3.76273582e-04, -9.89722524e-04,
         1.61649107e-05,  5.08641646e-07,  8.34113686e-08],
       [-9.88527979e-01, -7.85045700e-02, -1.29006347e-01,
         7.55924779e-05,  2.03996151e-03,  1.20741750e-04,
         7.07670819e-04,  4.28685722e-04, -1.71078316e-04,
        -1.53750220e-04, -1.40106001e-03,  7.26666464e-06],
       [-1.38499731e-03, -1.09990431e-04, -1.80746976e-04,
         1.05910386e-07,  2.85813001e-06,  1.69165107e-07,
         9.91432442e-07,  3.92137972e-06,  1.43555764e-01,
         1.09387008e-01,  9.99981556e-01, -5.18831743e-03]])
```

In [20]:

```
v_prime = v[:, 6]+v[:,7]+v[:,8]+v[:,9]+v[:,10]+v[:,11]
```

In [21]:

```
v_prime
```

Out[21]:

```
array([ 8.27716117e-01, -9.85892009e-01,  4.57552642e-01,  1.02274671e+00,
       -5.23093282e-01,  8.57293061e-01,  2.61344440e-01,  1.00061367e+00,
       -1.48602276e-03, -1.65964924e-03, -5.82265345e-04,  1.24774092e+00])
```

In [22]:

```
P = v_prime.reshape((3,4))
```

In [23]:

```
P
```

Out[23]:

```
array([[ 8.27716117e-01, -9.85892009e-01,  4.57552642e-01,
         1.02274671e+00],
       [-5.23093282e-01,  8.57293061e-01,  2.61344440e-01,
         1.00061367e+00],
       [-1.48602276e-03, -1.65964924e-03, -5.82265345e-04,
         1.24774092e+00]])
```

In [24]:

```
P[:,:-1]
```

Out[24]:

```
array([[ 8.27716117e-01, -9.85892009e-01,  4.57552642e-01],
       [-5.23093282e-01,  8.57293061e-01,  2.61344440e-01],
       [-1.48602276e-03, -1.65964924e-03, -5.82265345e-04]])
```

In [25]:

```
P[:,:-1].shape
```

Out[25]:

```
(3, 3)
```

Projection matrix QR factorization

In [26]:

```
Mext, Mint = np.linalg.qr(P[:,:-1])
```

In [27]:

```
Mext.shape
```

Out[27]:

```
(3, 3)
```

In [28]:

```
Mint.shape
```

Out[28]:

```
(3, 3)
```

In [29]:

```
Mint
```

Out[29]:

```
array([[-0.9791541 ,  1.29140089, -0.24716951],
       [ 0.        , -0.19804279, -0.46528921],
       [ 0.        ,  0.        ,  0.00829817]])
```

In [30]:

```
f_x_pixels = Mint[0][0]/P[2][2]
```

In [31]:

```
f_x_pixels
```

Out[31]:

```
1681.6286825782588
```

In [32]:

```
p_x_pixels = Mint[0][2]/P[2][2]
```

In [33]:

```
p_x_pixels
```

Out[33]:

```
424.49634514003293
```

In [34]:

```
p_y_pixels = Mint[1][2]/P[2][2]
```

In [35]:

```
p_y_pixels
```

Out[35]:

```
799.1016719823848
```

In [36]:

```
Mext
```

Out[36]:

```
array([[-0.84533794, -0.53411771,  0.01104667],
       [ 0.53422978, -0.84521254,  0.01463933],
       [ 0.00151766,  0.01827664,  0.99983182]])
```

Finding Rotations along X-axis, Y-axis and Z-axis

We will consider for our experiment that checkered board was first rotated along X-axis, Y-axis and Z-axis

Rotation Matrix along X:

$R_{\theta_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x\end{bmatrix}$

Rotation Matrix along Y:

$R_{\theta_y} = \begin{bmatrix} cos\theta_y & 0 & sin\theta_y \\ 0 & 1 & 0 \\ -sin\theta_y & 0 & cos\theta_y\end{bmatrix}$

Rotation Matrix along Z:

$R_{\theta_z} = \begin{bmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Multiplying R_{θ_x} and R_{θ_y} and R_{θ_z} :

$R_{\theta_{xyz}} = \begin{bmatrix} cos\theta_y cos\theta_z & -cos\theta_y sin\theta_z & sin\theta_y \\ sin\theta_x sin\theta_y cos\theta_z + cos\theta_x sin\theta_z & -sin\theta_x sin\theta_y sin\theta_z + cos\theta_x cos\theta_z & -sin\theta_x cos\theta_y \\ -cos\theta_x sin\theta_y cos\theta_z + sin\theta_x sin\theta_z & cos\theta_x sin\theta_y sin\theta_z + sin\theta_x cos\theta_z & cos\theta_x cos\theta_y \end{bmatrix}$

So our final rotation matrix is in this form from this $R_{\theta_{xyz}}$ matrix we need to compare with the values in M_{ext} and we will get our angles of rotation at each axis respectively

To get θ_y

We need take $cos^{-1}(R_{\theta_{xyz}}(1)(3))$ and this will give us θ_y.

where (1)(3) correspond to the indices of the rows and colums of the matrix

In [37]:

```
theta_y = np.arcsin(Mext[0][2])
```

Therefore, $\theta_y = 0.011047008729553763^{\circ} \approx 0^{\circ}$

In [38]:

```
theta_y
```

Out[38]:

```
0.01104689398824776
```

To get θ_z

1. We know θ_y. We can use it to get θ_z from $R_{\theta_{xyz}}(1)(1)$
2. Divide $cos\theta_y$ from $R_{\theta_{xyz}}(1)(1)$ to get $cos\theta_z$
3. Once we get $cos\theta_z$ we can take $cos^{-1}(\theta_z)$ to get θ_z

In [39]:

```
cos_theta_z = Mext[0][0]/np.cos(theta_y)
```

In [40]:

```
cos_theta_z
```

Out[40]:

```
-0.845389526216112
```

In [41]:

```
theta_z = np.arccos(cos_theta_z)
```

Therefore, $\theta_z = 2.5780871591054306^{\circ}$

In [42]:

```
theta_z
```

Out[42]:

```
2.5780903181602675
```

To get θ_x

1. We know θ_y. We can use it to get θ_x from $R_{\theta_{xyz}}(3)(3)$
2. Divide $cos\theta_y$ from $R_{\theta_{xyz}}(3)(3)$ to get $cos\theta_x$
3. Once we get $cos\theta_x$ we can take $cos^{-1}(\theta_x)$ to get θ_x

In [43]:

```
cos_theta_x = Mext[2][2]/np.cos(theta_y)
```

In [44]:

```
cos_theta_x
```

Out[44]:

```
0.9998928262377489
```

In [45]:

```
theta_x = np.arccos(cos_theta_x)
```

Therefore, $\theta_x = 0.014640797214501924^{\circ} \approx 0^{\circ}$

In [46]:

```
theta_x
```

Out[46]:

```
0.014640742923434221
```

To find translations:

$ t = M^{-1}*\begin{bmatrix} P_{14} \\ P_{24} \\ P_{34} \\ P_{44}\end{bmatrix}$

In [47]:

```
t = np.matmul(np.linalg.inv(Mint),P[:,-1])
```

In [48]:

```
t
```

Out[48]:

```
array([-511.58967457, -358.32184377,  150.36334918])
```

In [ ]:

In [ ]: