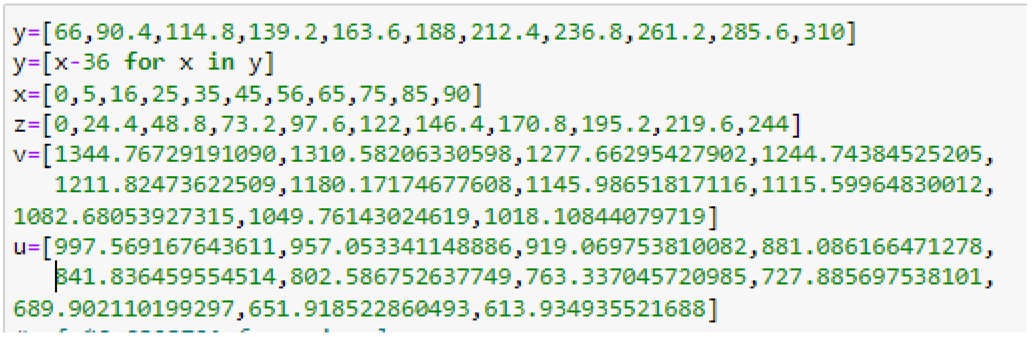
CV Assignment-1

Sai Kowshik Ananthula

Part-A

Getting the values from real world and image using manual measurement and MATLAB values.



```
y=[66,90.4,114.8,139.2,163.6,188,212.4,236.8,261.2,285.6,310]
y=[x-36 for x in y]
x=[0,5,16,25,35,45,56,65,75,85,90]
z=[0,24.4,48.8,73.2,97.6,122,146.4,170.8,195.2,219.6,244]
v=[1344.76729191090,1310.58206330598,1277.66295427902,1244.74384525205,
   1211.82473622509,1180.17174677608,1145.98651817116,1115.59964830012,
1082.68053927315,1049.76143024619,1018.10844079719]
u=[997.569167643611,957.053341148886,919.069753810082,881.086166471278,
   841.836459554514,802.586752637749,763.337045720985,727.885697538101,
689.902110199297,651.918522860493,613.934935521688]
```

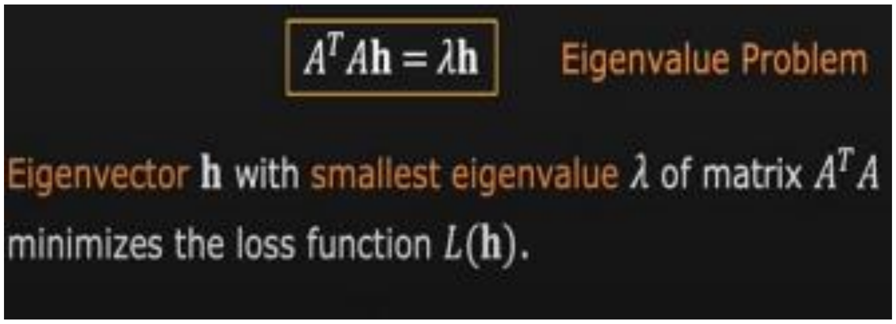
$$\begin{bmatrix} x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ & & & & & \vdots & & & \\ x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\ & & & & & \vdots & & & \\ x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

By using the above system of equations re-arranging, the collected values.

```
In [7]: a=[]
        for i in range(1,11):
            a.append([x[i],y[i],z[i],1,0,0,0,0,-1*u[i]*x[i],-1*u[i]*y[i],-1*u[i]*z[i],-1*u[i]])
            a.append([0,0,0,0,x[i],y[i],z[i],1,-1*v[i]*x[i],-1*v[i]*y[i],-1*v[i]*z[i],-1*v[i]])
```

```
In [8]: a
```

```
Out[8]: [[10, 24, 24, 1, 0, 0, 0, 0, -15222.5, -36534.0, -36534.0, -1522.25],
         [0, 0, 0, 0, 10, 24, 24, 1, -7602.5, -18246.0, -18246.0, -760.25],
         [16, 48, 48, 1, 0, 0, 0, 0, -23492.0, -70476.0, -70476.0, -1468.25],
         [0, 0, 0, 0, 16, 48, 48, 1, -11204.0, -33612.0, -33612.0, -700.25],
         [24, 72, 72, 1, 0, 0, 0, 0, -33978.0, -101934.0, -101934.0, -1415.75],
         [0, 0, 0, 0, 24, 72, 72, 1, -15438.0, -46314.0, -46314.0, -643.25],
         [30, 96, 96, 1, 0, 0, 0, 0, -40987.5, -131160.0, -131160.0, -1366.25],
         [0, 0, 0, 0, 30, 96, 96, 1, -17452.5, -55848.0, -55848.0, -581.75],
         [36, 120, 120, 1, 0, 0, 0, 0, -47403.0, -158010.0, -158010.0, -1316.75],
         [0, 0, 0, 0, 36, 120, 120, 1, -18945.0, -63150.0, -63150.0, -526.25],
         [44, 144, 144, 1, 0, 0, 0, 0, -55561.0, -181836.0, -181836.0, -1262.75],
         [0, 0, 0, 0, 44, 144, 144, 1, -20581.0, -67356.0, -67356.0, -467.75],
         [50, 168, 168, 1, 0, 0, 0, 0, -60662.5, -203826.0, -203826.0, -1213.25],
         [0, 0, 0, 0, 50, 168, 168, 1, -20462.5, -68754.0, -68754.0, -409.25],
         [55, 192, 192, 1, 0, 0, 0, 0, -63841.25, -222864.0, -222864.0, -1160.75],
         [0, 0, 0, 0, 55, 192, 192, 1, -19208.75, -67056.0, -67056.0, -349.25],
         [64, 216, 216, 1, 0, 0, 0, 0, -70832.0, -239058.0, -239058.0, -1106.75],
         [0, 0, 0, 0, 64, 216, 216, 1, -18416.0, -62154.0, -62154.0, -287.75],
         [70, 240, 240, 1, 0, 0, 0, 0, -73797.5, -253020.0, -253020.0, -1054.25],
         [0, 0, 0, 0, 70, 240, 240, 1, -15522.5, -53220.0, -53220.0, -221.75]]
```


$\boxed{A^T A\mathbf{h} = \lambda\mathbf{h}}$ Eigenvalue Problem

Eigenvector $\mathbf{h}$ with smallest eigenvalue λ of matrix $A^T A$ minimizes the loss function $L(\mathbf{h})$.

Calculating parametric matric by using Eigen value decomposition

```
c=np.linalg.eig(b.T*b)
p=c[1][np.where(c[0]==min(c[0]))[0][0]]
p=np.array(p).reshape((3,4))
```

```
p
```

```
array([[ 1.07752786e-04, -1.73820071e-02, -1.05056488e-01,
         -2.45115951e-01],
       [-2.83318763e-01,  5.81408591e-01,  9.31285259e-02,
         -5.22361073e-01],
       [-3.93185066e-01,  1.02689974e-01,  2.15440041e-01,
          1.31093616e-01]])
```

Using QR Factorization to get both translation and rotation matrix and translation matrix

```
r,K=np.linalg.qr(qr_mat)
K=K.T
K/=K[2][2]
K[0][0]=K[0][0]*4208/5.867
K[1][1]=K[1][1]*3120/5.867
```

Intrincsic Matrix

```
r
```

```
array([[-2.22341358e-04,  4.19574966e-02,  9.99119372e-01],
       [ 5.84611135e-01, -8.10593729e-01,  3.41705584e-02],
       [ 8.11313608e-01,  5.84103908e-01, -2.43485912e-02]])
```

```
K.T
```

```
array([[ 3.24767830e+03, -3.95427391e+00, -2.14203851e+00],
       [-0.00000000e+00,  2.04727653e+03, -4.29255239e-01],
       [-0.00000000e+00, -0.00000000e+00,  1.00000000e+00]])
```

Extrensic Matrix

```
P=[p[0][3],p[1][3],p[2][3]]
np.dot(np.linalg.inv(K),P)
```

```
array([-7.54742091e-05, -2.55295028e-04,  1.30822361e-01])
```

Angles of rotation

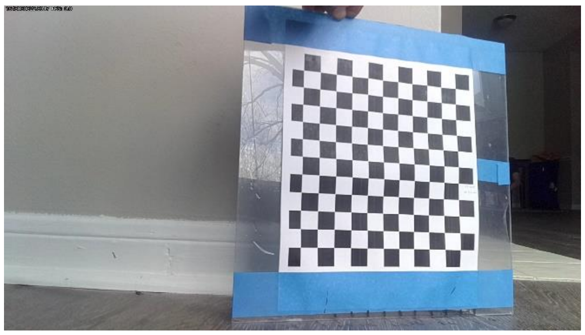
```
theta=acos(r[2][2])
phi=asin(r[2][1]/sin(theta))
gamma=asin(r[1][2]/sin(theta))
[theta,phi,gamma]
```

```
[1.5951473245038992, 0.6239890476480262, 0.034187351150459636]
```

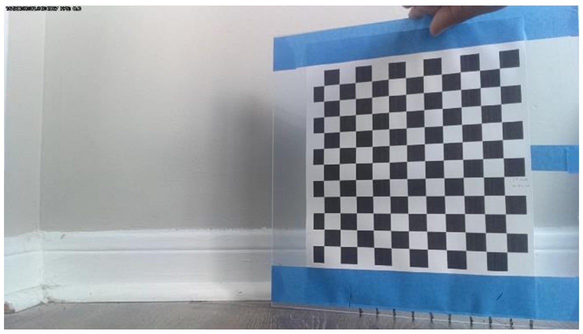
$$\mathbf{t} = K^{-1}\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$

Getting translation from calibration matrix

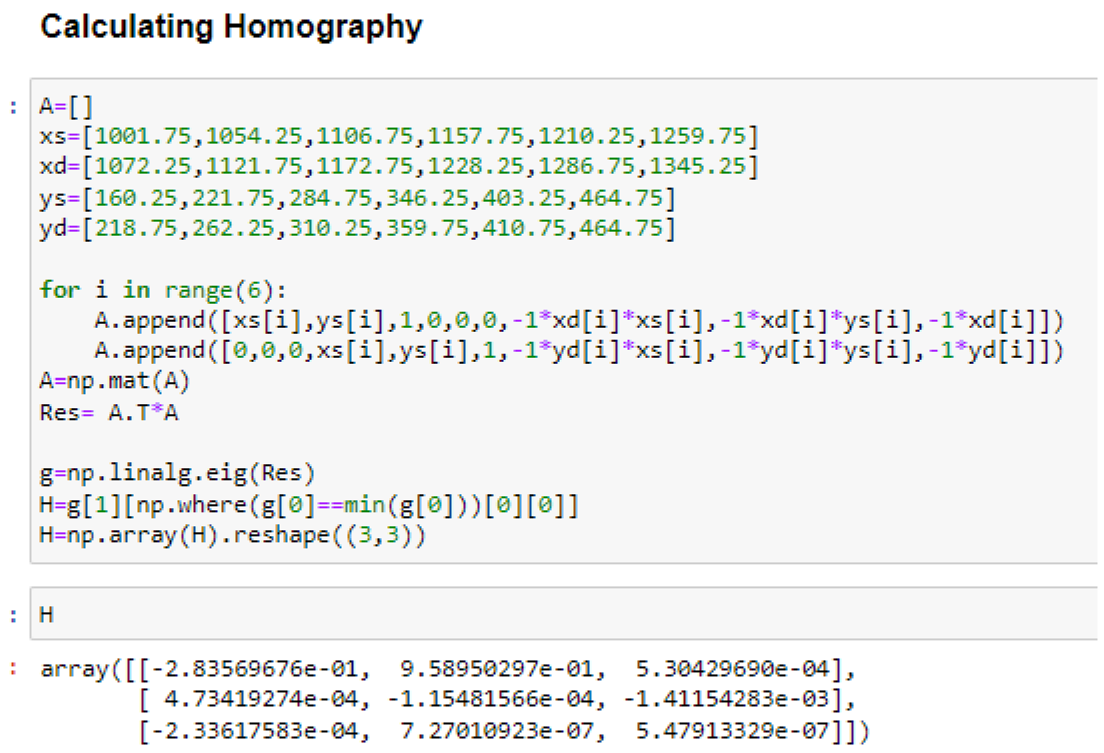
Calculating Homography



Source Image



Destination Image



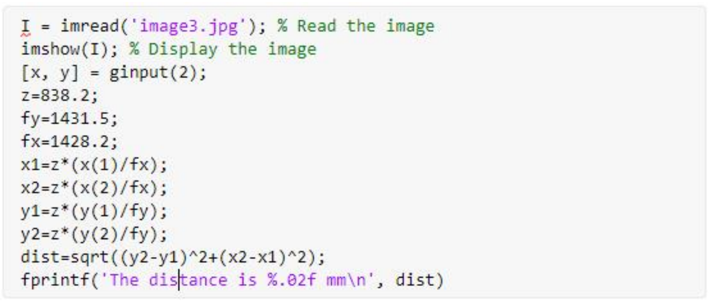
Calculating Homography

```
: A=[]
  xs=[1001.75,1054.25,1106.75,1157.75,1210.25,1259.75]
  xd=[1072.25,1121.75,1172.75,1228.25,1286.75,1345.25]
  ys=[160.25,221.75,284.75,346.25,403.25,464.75]
  yd=[218.75,262.25,310.25,359.75,410.75,464.75]

  for i in range(6):
      A.append([xs[i],ys[i],1,0,0,0,-1*xd[i]*xs[i],-1*xd[i]*ys[i],-1*xd[i]])
      A.append([0,0,0,xs[i],ys[i],1,-1*yd[i]*xs[i],-1*yd[i]*ys[i],-1*yd[i]])
  A=np.mat(A)
  Res= A.T*A

  g=np.linalg.eig(Res)
  H=g[1][np.where(g[0]==min(g[0]))[0][0]]
  H=np.array(H).reshape((3,3))
```

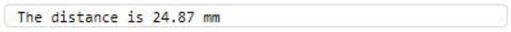
```
: H
```

```
: array([[-2.83569676e-01,  9.58950297e-01,  5.30429690e-04],
         [ 4.73419274e-04, -1.15481566e-04, -1.41154283e-03],
         [-2.33617583e-04,  7.27010923e-07,  5.47913329e-07]])
```

PART-B



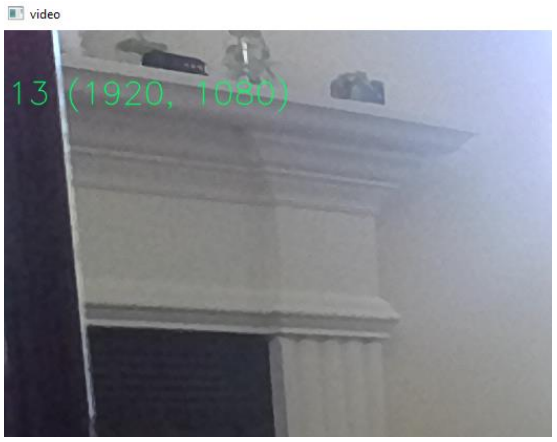
```
I = imread('image3.jpg'); % Read the image
imshow(I); % Display the image
[x, y] = ginput(2);
z=838.2;
fy=1431.5;
fx=1428.2;
x1=z*(x(1)/fx);
x2=z*(x(2)/fx);
y1=z*(y(1)/fy);
y2=z*(y(2)/fy);
dist=sqrt((y2-y1)^2+(x2-x1)^2);
fprintf('The distance is %.02f mm\n', dist)
```

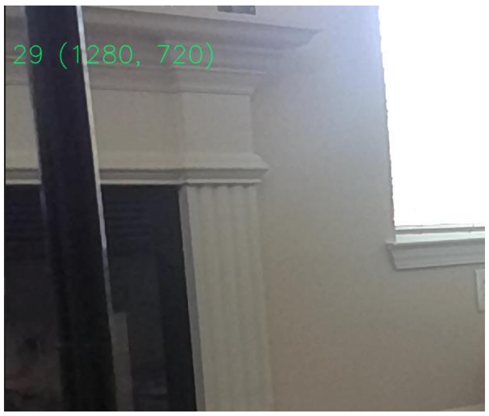


```
The distance is 24.87 mm
```

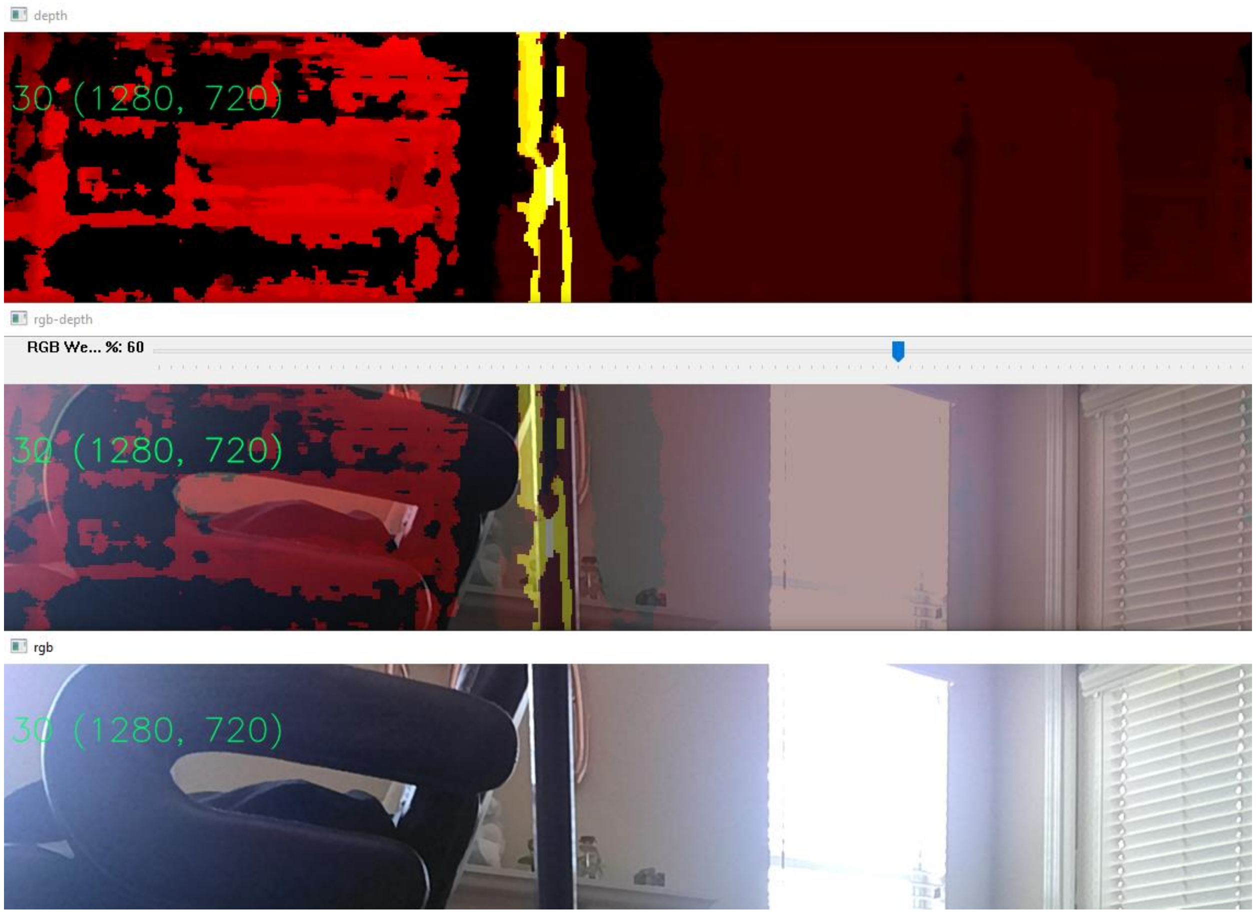
This is the distance between each diagonal of a square. The original value is somewhere around 25 mm.

Part-C

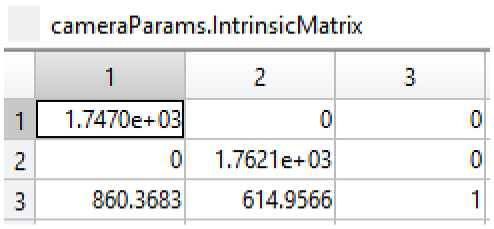




From the Images the fps of the camera with 1080p resolution is 15fps and the same RGB lens with 720p is 30fps.



Fps and resolution of RGB and Depth aligned which is 30fps.



cameraParams.IntrinsicMatrix

	1	2	3
1	1.7470e+03	0	0
2	0	1.7621e+03	0
3	860.3683	614.9566	1

My calibration value differs by 1mm for fx and fy which is a bit minimal but not accurate. I hope this is considerable.