Camera Calibration and Augmented Reality

Python Modules used:

```
math
matplotlib
NumPy
skimage
sklearn
opencv
pickle
copy
```

Folder Structure:

⇒ ***Assignment 3:***

- Part1.ipynb
- Part2.ipynb
- Part3.ipynb
- findHomography2d.py
- normalise2dpts.py
- H_matrix.pkl
- K_matrix.pkl
- Rotation.pkl
- Translation.pkl
- ⇒ ***Assignment3files***
  - *All Images*
- ⇒ ***clipart***
  - All cliparts

Part I: Camera Calibration using 3D calibration object

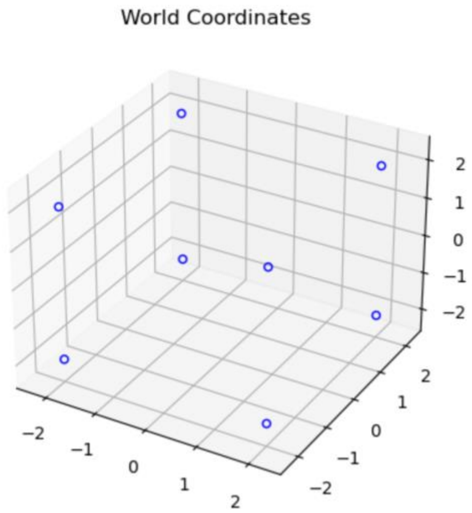
We are given world coordinates and pixel positions of a cube and we have to calibrate the camera using these details.

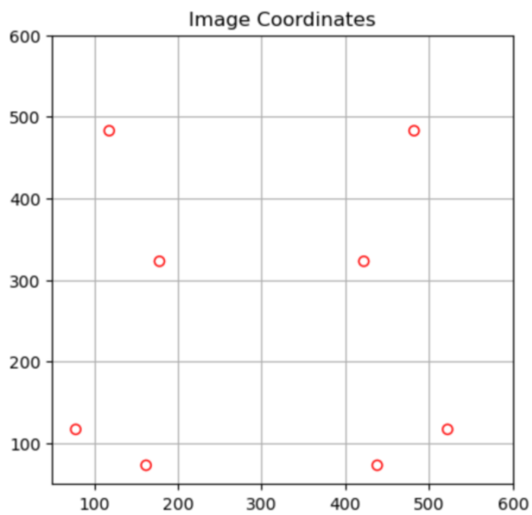
World Coordinates:

2 2 2,
-2 2 2,
-2 2 -2,
2 2 -2,
2 -2 2,
-2 -2 2,
-2 -2 -2,
2 -2 -2

Pixel Positions in Camera Image:

422 323; 178 323; 118 483; 482 483; 438 73; 162 73; 78 117; 522 117





Matrix P:

```
Shape of matrix P: (16, 12)
[[ 2.000e+00  2.000e+00  2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00 -8.440e+02 -8.440e+02 -8.440e+02 -4.220e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00  2.000e+00  2.000e+00 2.000e+00  1.000e+00 -6.460e+02 -6.460e+02 -6.460e+02 -3.230e+02]
 [-2.000e+00  2.000e+00  2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00  3.560e+02 -3.560e+02 -3.560e+02 -1.780e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00 -2.000e+00  2.000e+00 2.000e+00  1.000e+00  6.460e+02 -6.460e+02 -6.460e+02 -3.230e+02]
 [-2.000e+00  2.000e+00 -2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00  2.360e+02 -2.360e+02  2.360e+02 -1.180e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00 -2.000e+00  2.000e+00 -2.000e+00  1.000e+00  9.660e+02 -9.660e+02  9.660e+02 -4.830e+02]
 [ 2.000e+00  2.000e+00 -2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00 -9.640e+02 -9.640e+02  9.640e+02 -4.820e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00  2.000e+00  2.000e+00 -2.000e+00  1.000e+00 -9.660e+02 -9.660e+02  9.660e+02 -4.830e+02]
 [ 2.000e+00 -2.000e+00  2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00 -8.760e+02  8.760e+02 -8.760e+02 -4.380e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00  2.000e+00 -2.000e+00 2.000e+00  1.000e+00 -1.460e+02  1.460e+02 -1.460e+02 -7.300e+01]
 [-2.000e+00 -2.000e+00  2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00  3.240e+02  3.240e+02 -3.240e+02 -1.620e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00 -2.000e+00 -2.000e+00 2.000e+00  1.000e+00  1.460e+02  1.460e+02 -1.460e+02 -7.300e+01]
 [-2.000e+00 -2.000e+00 -2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00  1.560e+02  1.560e+02  1.560e+02 -7.800e+01]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00 -2.000e+00 -2.000e+00 -2.000e+00  1.000e+00  2.340e+02  2.340e+02  2.340e+02 -1.170e+02]
 [ 2.000e+00 -2.000e+00 -2.000e+00  1.000e+00  0.000e+00  0.000e+00 0.000e+00  0.000e+00 -1.044e+03  1.044e+03  1.044e+03 -5.220e+02]
 [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00  2.000e+00 -2.000e+00 -2.000e+00  1.000e+00 -2.340e+02  2.340e+02  2.340e+02 -1.170e+02]]
```

Matrix M:

```
Projection Matrix:
 [[-1.925e-01 -2.830e-02 -7.860e-02 -7.346e-01]
 [-0.000e+00 -2.044e-01 -1.000e-04 -6.120e-01]
 [-0.000e+00 -1.000e-04 -3.000e-04 -2.400e-03]]
```

Euclidean Coordinates:

```
Euclidean coordinates of camera center in the frame of reference of the cube:
[x, y, z] =  [-4.000000e-06 -2.991214e+00 -8.269539e+00]
```

Rescaled M matrix (M_prime):

```
Normalized M':
 [[7.34628863e+02 1.07895535e+02 2.99999893e+02]
 [9.25005842e-04 7.80144198e+02 2.64050222e-01]
 [3.71488286e-06 3.59651924e-01 1.00000000e+00]]
```

R_x, θ_x, N

```
Rx:
[[ 1.          0.          0.        ]
 [ 0.          0.94099175  0.33842949]
 [ 0.         -0.33842949  0.94099175]]

theta_x = -19.78121922876508

N:
[[ 7.34628863e+02 -3.67940535e-06  3.18812456e+02]
 [ 9.25005842e-04  7.34019893e+02  2.64272275e+02]
 [ 3.71488286e-06  0.00000000e+00  1.06270857e+00]]
```

R_z, θ_z

```
Rz:
[[ 1.00000000e+00  1.26019179e-06  0.00000000e+00]
 [-1.26019179e-06  1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]

theta_z = -7.220367084089655e-05
```

Calibration Matrix (K), Focal Lengths and Pixel Coordinates of Image Center of Camera

```
Calibration Matrix:
[[ 8.96681894e+02  2.47849337e+02  3.00000277e+02]
 [ 2.32905594e-03  8.96156575e+02 -3.21961928e+02]
 [ 5.08756351e-06  8.26168312e-01  1.00000000e+00]]

Focal Lengths: [896.68189361 896.15657474]

Pixel Coordinates of Image Center: [ 300.00027691 -321.96192818]
```

Part II: Camera Calibration using 2D calibration object

In this part, we implement camera calibration from multiple images of 2D planes.

Calibration Grid and Images:

Details Given about Grid:
- 9 squares in width and 7 in height
- Each square dimension is 30 mm x 30 mm

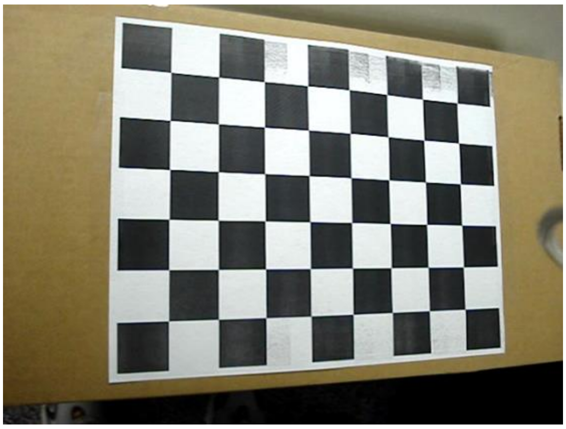
Images to use for calibration:
- images2.png, images9.png, images12.png, images20.png

Corner Extraction and Homography Computation

images2.png

Homography H:

```
images2.png
 [[ 1.726400e+00  1.520000e-01  6.694090e+01]
 [ 3.050000e-02 -1.563400e+00  4.106456e+02]
 [-0.000000e+00  4.000000e-04  1.000000e+00]]
```



images9.png

Homography H:

```
images9.png
 [[ 2.198300e+00  6.870000e-02  1.319043e+02]
 [ 2.913000e-01 -1.881100e+00  4.205553e+02]
 [ 1.000000e-03  3.000000e-04  1.000000e+00]]
```

images12.png

Homography H:

```
images12.png
 [[ 1.132900e+00  7.230000e-02  1.032763e+02]
 [-2.760000e-01 -1.411900e+00  3.930284e+02]
 [-8.000000e-04  3.000000e-04  1.000000e+00]]
```

images20.png

Homography H:

```
images20.png
 [[ 1.684300e+00  5.197000e-01  1.275000e+02]
 [-1.460000e-02 -7.783000e-01  2.752134e+02]
 [ 0.000000e+00  1.600000e-03  1.000000e+00]]
```

Matplotlib's ginput function is used to select the four corners of the grid, starting from bottom left in anticlockwise fashion: Bottom-Left, Bottom-Right, Top-Right, Top-Left

Once we collect the grid corners, these along with the real coordinates calculated with the help of provided dimesnions are used to compute homography matrix H of each of the above images.

Computing the Intrinsic and Extrinsic Parameters

Having calculated the four homographies corresponding to four images, we now compute extrinsic and intrinsic parameters.

Computed Matrix B:

```
matrix B:
 [[-0.e+00  0.e+00  5.e-04]
 [ 0.e+00 -0.e+00  3.e-04]
 [ 5.e-04  3.e-04 -1.e+00]]
```

We can compute intrinsic matrix containing intrinsic parameters with the help of method specified in the paper Flexible Camera Calibration By Viewing a Plane From Unknown Orientation by Zhengyou Zhang.

$$
\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}} \\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
c &= -B_{12}\alpha^2\beta/\lambda \\
u_0 &= cv_0/\alpha - B_{13}\alpha^2/\lambda \,.
\end{aligned}
$$

Intrinsic Matrix:

```
Intrinsic Matrix (K)
 [[745.0844   1.3561 331.815 ]
 [  0.     723.2172 227.0186]
 [  0.       0.       1.    ]]
```

Computed R, R^TR, t, new R and new R^TR:

Images2.png

```
** matrix R **
[[ 0.9998  0.0126  0.0012]
 [ 0.0189 -0.9863 -0.1732]
 [-0.0021  0.1732 -0.9864]]
```

```
** vector t **
[-153.4523  109.4569  431.0973]
```

```
** matrix RTR **
[[ 1.     -0.0064 -0.    ]
 [-0.0064  1.0029  0.    ]
 [-0.      0.      1.0029]]
```

```
** rec_R **
[[ 0.9999  0.0158  0.0012]
 [ 0.0157 -0.9848 -0.1729]
 [-0.0016  0.1729 -0.9849]]
```

```
** rec_RTR **
[[ 1. -0. -0.]
 [-0.  1. -0.]
 [-0. -0.  1.]]
```

Images9.png

```
** matrix R **
[[ 0.9242 -0.0084  0.3821]
 [ 0.0298 -0.9955 -0.0952]
 [ 0.3808  0.0996 -0.9198]]
```

```
** vector t **
[-99.6518  99.2114 370.7377]
```

```
** matrix RTR **
[[ 1.0000e+00  5.0000e-04 -0.0000e+00]
 [ 5.0000e-04  1.0011e+00  0.0000e+00]
 [-0.0000e+00  0.0000e+00  1.0011e+00]]
```

```
** rec_R **
[[ 0.9242 -0.0086  0.3819]
 [ 0.0301 -0.995  -0.0952]
 [ 0.3808  0.0995 -0.9193]]
```

```
** rec_RTR **
[[ 1.  0. -0.]
 [ 0.  1.  0.]
 [-0.  0.  1.]]
```

Images12.png

```
** matrix R **
[[ 0.915  -0.0096 -0.402 ]
 [-0.0597 -0.9876 -0.1161]
 [-0.3991  0.1311 -0.9042]]
```

```
** vector t **
[-148.9032  111.2817  484.7956]
```

```
** matrix RTR **
[[ 1.      -0.0021  0.    ]
 [-0.0021  0.9927 -0.    ]
 [ 0.     -0.      0.9927]]
```

```
** rec_R **
[[ 0.915  -0.0086 -0.4035]
 [-0.0608 -0.9913 -0.1166]
 [-0.399   0.1312 -0.9075]]
```

```
** rec_RTR **
[[ 1. -0. -0.]
 [-0.  1. -0.]
 [-0. -0.  1.]]
```

Images20.png

```
** matrix R **
[[ 0.9999 -0.0059 -0.0019]
 [-0.012  -0.7015 -0.7119]
 [ 0.0095  0.7119 -0.7014]]
```

```
** vector t **
[-121.8509   29.5988  444.1624]
```

```
** matrix RTR **
[[ 1.      0.0092  0.    ]
 [ 0.0092  0.9989 -0.    ]
 [ 0.     -0.      0.9988]]
```

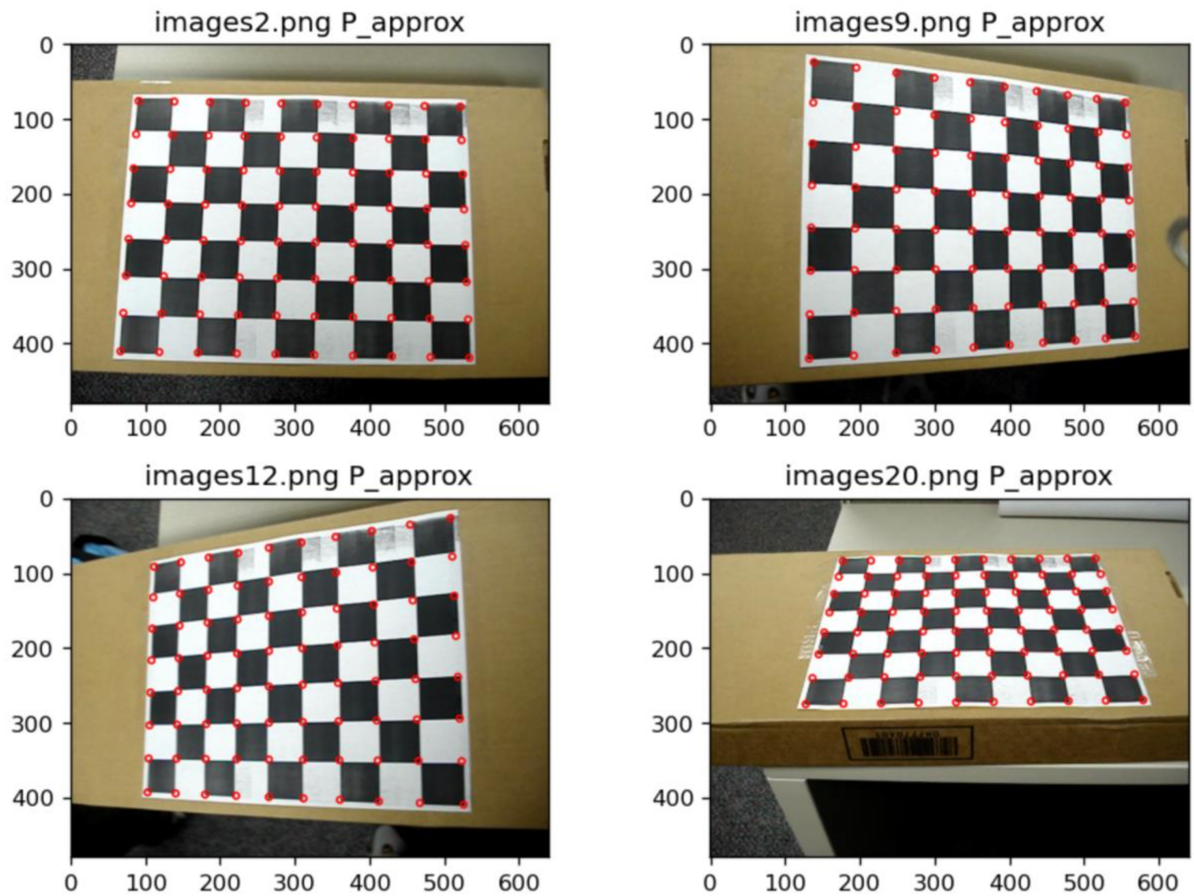
```
** rec_R **
[[ 0.9999 -0.0105 -0.0019]
 [-0.0087 -0.7018 -0.7123]
 [ 0.0062  0.7123 -0.7019]]
```

```
** rec_RTR **
[[ 1. -0. -0.]
 [-0.  1. -0.]
 [-0. -0.  1.]]
```

Improving Accuracy

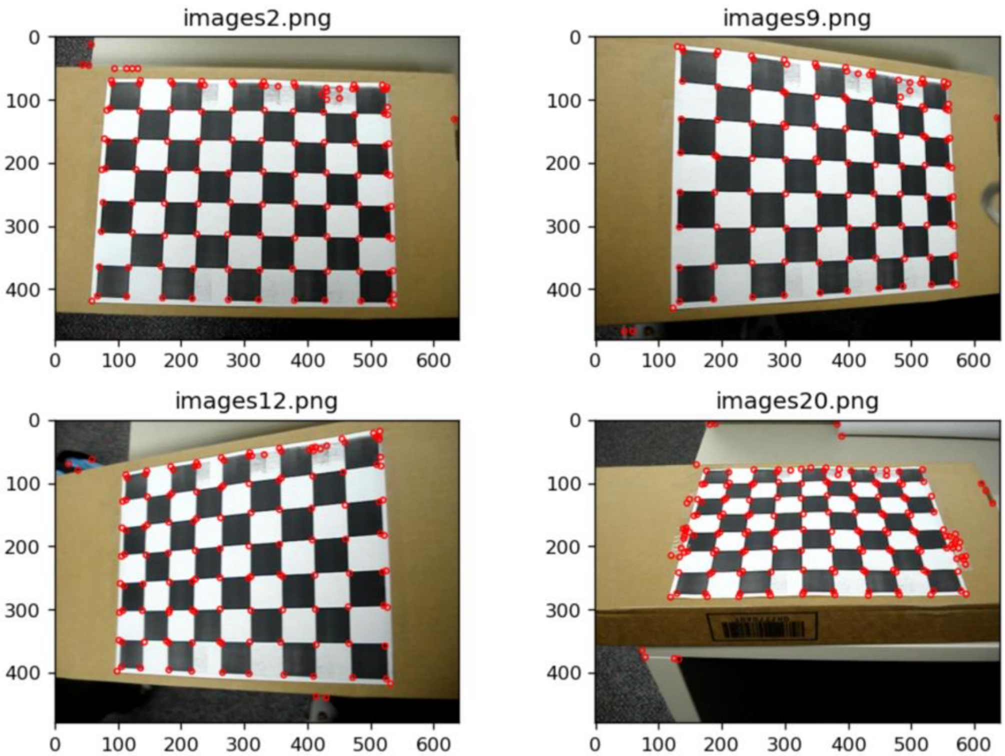
Approximate Grid Locations:

Figure 1: Projected Grid Corners



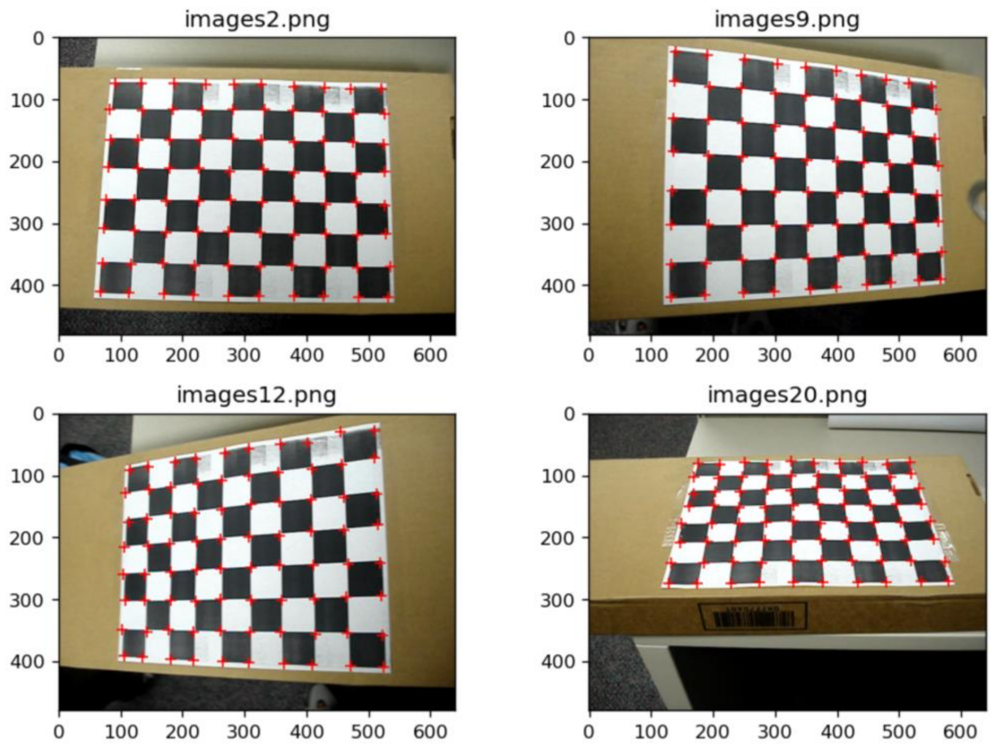
Harris Corners:

Figure 2: Harris Corners



Grid Points:

Figure 3: Grid Points



New Homography and Error Projection computed using p_correct:

```
images2.png
 new H matrix:
 [[ 1.74785175e+00  1.60020987e-01  6.34772260e+01]
 [ 2.98519688e-02 -1.59777967e+00  4.14284521e+02]
 [ 4.42425935e-06  4.06508770e-04  1.00000000e+00]]
 err_projection = 0.3551514652818974
```

```
images9.png
 new H matrix:
 [[ 2.25151945e+00  7.79611802e-02  1.29021712e+02]
 [ 3.10134180e-01 -1.92295727e+00  4.23885520e+02]
 [ 1.09615217e-03  2.80467950e-04  1.00000000e+00]]
 err_projection = 0.32584087064219236
```

```
images12.png
 new H matrix:
 [[ 1.12766194e+00  8.05031894e-02  1.01318206e+02]
 [-2.82614646e-01 -1.42207941e+00  3.93842295e+02]
 [-8.55036581e-04  2.78803574e-04  1.00000000e+00]]
 err_projection = 0.3504815088796816
```

```
images20.png
 new H matrix:
 [[ 1.67501676e+00  5.07950826e-01  1.26865090e+02]
 [-1.89767946e-02 -7.91447696e-01  2.75250117e+02]
 [-1.06926564e-05  1.57322817e-03  1.00000000e+00]]
 err_projection = 0.2876557430925058
```

Updated K, R and t for each image:

```
New B:
 [[-0.e+00  0.e+00  5.e-04]
 [ 0.e+00 -0.e+00  3.e-04]
 [ 5.e-04  3.e-04 -1.e+00]]
```

```
New Intrinsic Matrix (K)
 [[745.0844   1.3561 331.815 ]
 [  0.      723.2172 227.0186]
 [  0.        0.        1.    ]]
```

```
images2.png
 ** matrix R **
 [[ 0.988   0.016   0.0047]
 [ 0.0168 -0.9851 -0.1693]
 [ 0.0019  0.1714 -0.9735]]
 ** vector t **
 [-152.0129  109.1507  421.5375]
```

```
images9.png
 ** matrix R **
 [[ 0.9175 -0.0055  0.398 ]
 [ 0.0307 -0.9948 -0.0954]
 [ 0.397   0.1016 -0.9126]]
 ** vector t **
 [-98.7487  98.582  362.1541]
```

```
images12.png
 ** matrix R **
 [[ 0.914  -0.006  -0.4167]
 [-0.059  -0.9909 -0.1205]
 [-0.4125  0.1345 -0.906 ]]
 ** vector t **
 [-149.4544  111.2883  482.459 ]
```

```
images20.png
 ** matrix R **
 [[ 1.0078 -0.0072 -0.0106]
 [-0.0102 -0.7104 -0.7092]
 [-0.0048  0.7037 -0.716 ]]
 ** vector t **
 [-123.0983   29.8319  447.32  ]
```

Saving all matrices for each image:

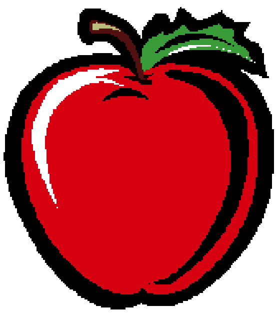
```
# save H, K, R and t
# index -> 0:'images2.png' 1:'images9.png' 2:'images12.png' 3:'images20.png']
open_file = open('H_matrix.pkl', "wb")
pickle.dump(H_rect_mat, open_file)
open_file.close()

open_file = open('K_matrix.pkl', "wb")
pickle.dump(K, open_file)
open_file.close()

open_file = open('Rotation.pkl', "wb")
pickle.dump(R_list, open_file)
open_file.close()

open_file = open('Translation.pkl', "wb")
pickle.dump(t_list, open_file)
open_file.close()
```

Ques. Can you suggest a way this can be done automatically (i.e without first letting the user manually select the 4 corners) ?

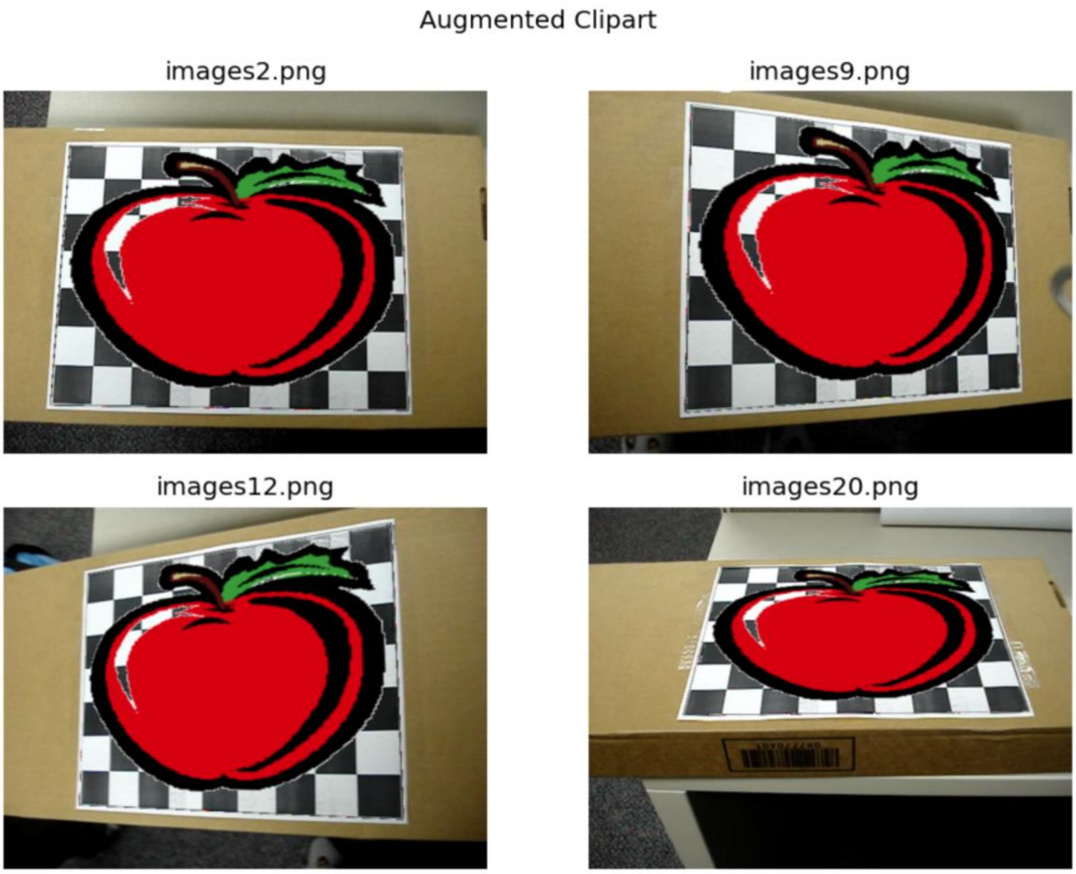
We can clearly observe here that Harris corners detects almost all the required points on the grid. It, however, detects few other edges as well. If we treat that as noise in the image, we can apply filters on the image to cover those up and then Harris corners will be able to precisely detect the points on the grid. From the collection of points on the grid, we can choose the four corners, as we already know the number of horizontal and vertical blocks on the grid. We also know the dimension of each block. We can therefore select the points based on the distance between them as four corners of the grid.

Part III: Augmented Reality

Augmenting an Image



1.gif



Augmenting an Object (3x3 grid)