# CLab-3 Assignment

ENGN 4528

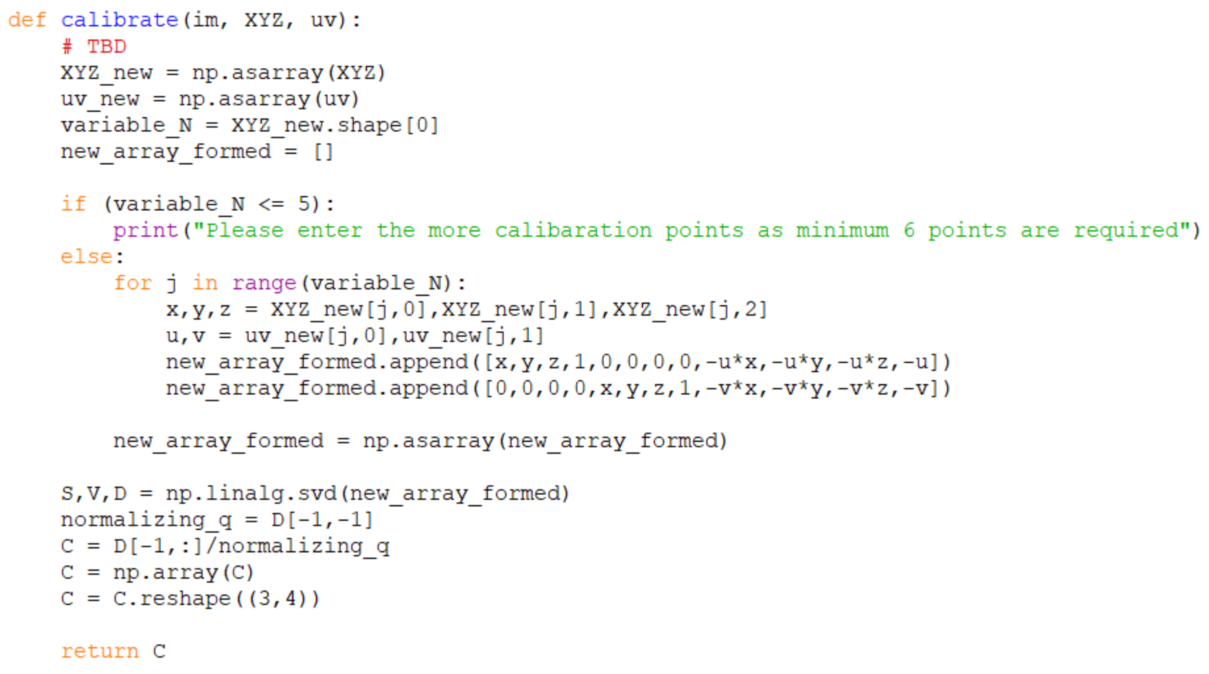
Australian National University

Prateek Arora

Student Id – u6742441

## Task 1: (3D – 2D Camera Calibration (6 marks))

### 1. List calibrate function in the pdf file[3]



```
def calibrate(im, XYZ, uv):
    # TBD
    XYZ_new = np.asarray(XYZ)
    uv_new = np.asarray(uv)
    variable_N = XYZ_new.shape[0]
    new_array_formed = []

    if (variable_N <= 5):
        print("Please enter the more calibaration points as minimum 6 points are required")
    else:
        for j in range(variable_N):
            x,y,z = XYZ_new[j,0],XYZ_new[j,1],XYZ_new[j,2]
            u,v = uv_new[j,0],uv_new[j,1]
            new_array_formed.append([x,y,z,1,0,0,0,0,-u*x,-u*y,-u*z,-u])
            new_array_formed.append([0,0,0,0,x,y,z,1,-v*x,-v*y,-v*z,-v])

        new_array_formed = np.asarray(new_array_formed)

    S,V,D = np.linalg.svd(new_array_formed)
    normalizing_q = D[-1,-1]
    C = D[-1,:]/normalizing_q
    C = np.array(C)
    C = C.reshape((3,4))

    return C
```

Direct linear transformation is an algorithm which solves a set of variables from the set of similarity relations. [1]

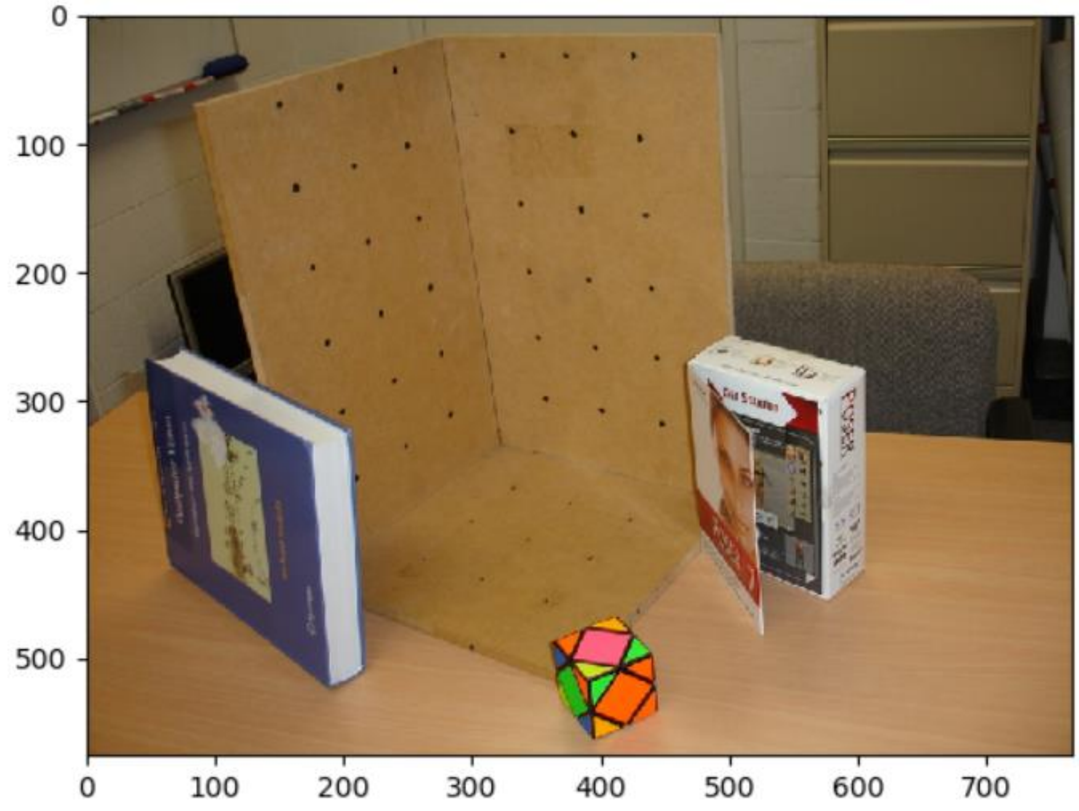
$x_k$ α A$y_k$ [1]
So, if we put this particular equation in another way which is shown below:

AX = Y
So, over, we assume that the value of X is unknown, so we find the value of X using the Moore pseudo equation, i.e.,

X = $(A^T\text{A})^{-1}A^T$Y

And if the equation is homogeneous, we use the SVD (single value decomposition) which is used in calibration function.

### 2. List which image you have chosen for your experiment, display the image in your pdf file[3]



This image is provided in the code. The name of this file is “stereo2012a”.

## 3. List 3x4 camera calibration matrix C that you have calculated[3]

So, in this function, the ginput takes the 6 inputs as specified in the code. We specify the points that we want to select.

Over here, the points that have been selected in the as xyz coordinate for the calibration function is

[[0,7,7,1], [0,7,14,1], [0,7,21,1], [7,7,0,1], [7,14,0,1], [14,0,14,1]]

uv points are selected from the ginput that we provide. In this algorithm, we have to make sure that minimum of 6 points are selected from the image as graphical input.
Then, we got the equation after normalization,

$$\begin{bmatrix} x & y & z & 1 & 0 & 0 & 0 & 0 & -ux & -uy & -uz & -u \\ 0 & 0 & 0 & 0 & x & y & z & 1 & -vx & -vy & -vz & -v \end{bmatrix}$$

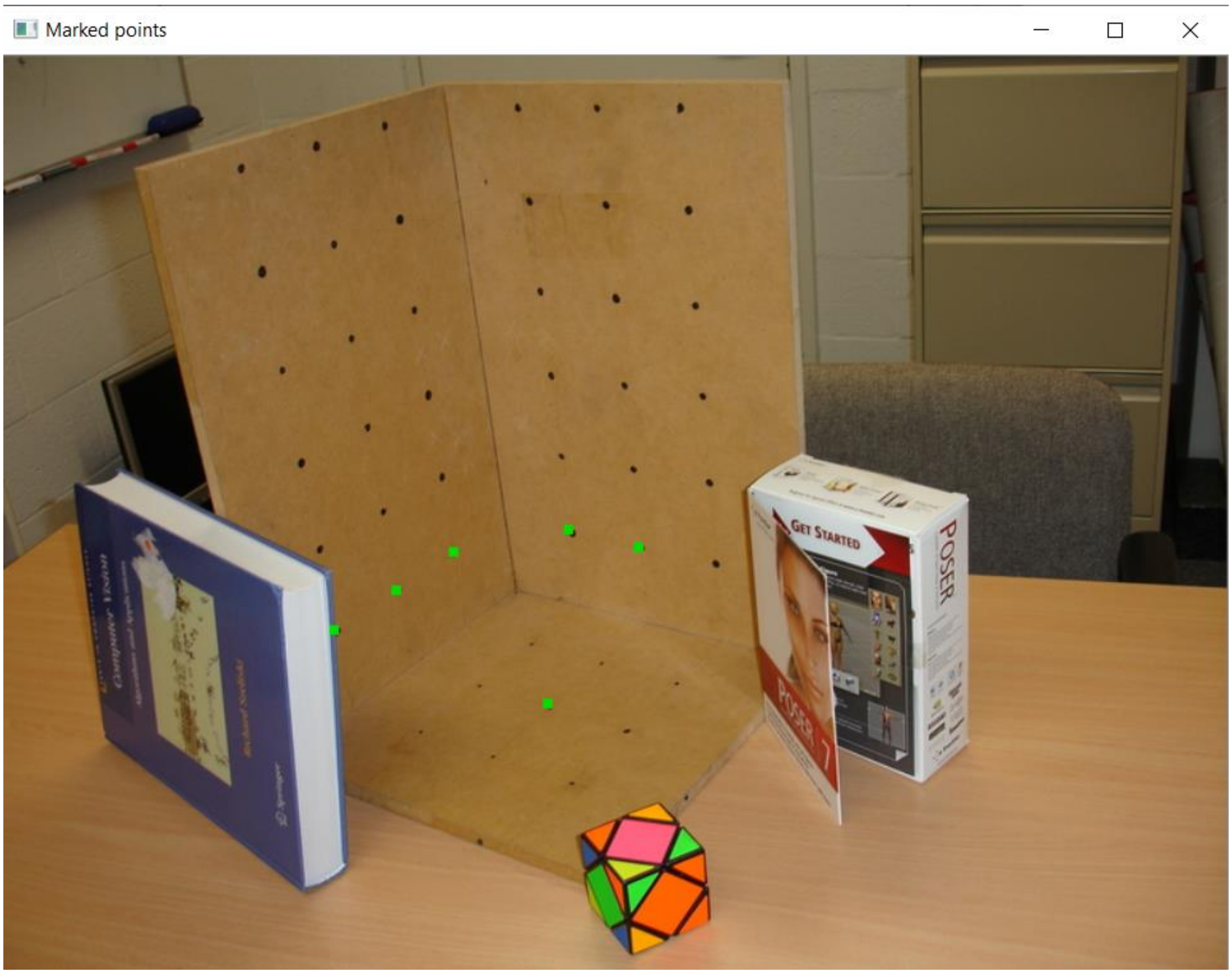
which forms the array. Now as we know that the equation formed is homogeneous, we used SVD to find the values which can be seen in the given code above. Then normalization is done on the matrices and then reshape function is used to get the suitable shaped array of size (3,4).

After that, we get the calibration matrix which is given as follows:

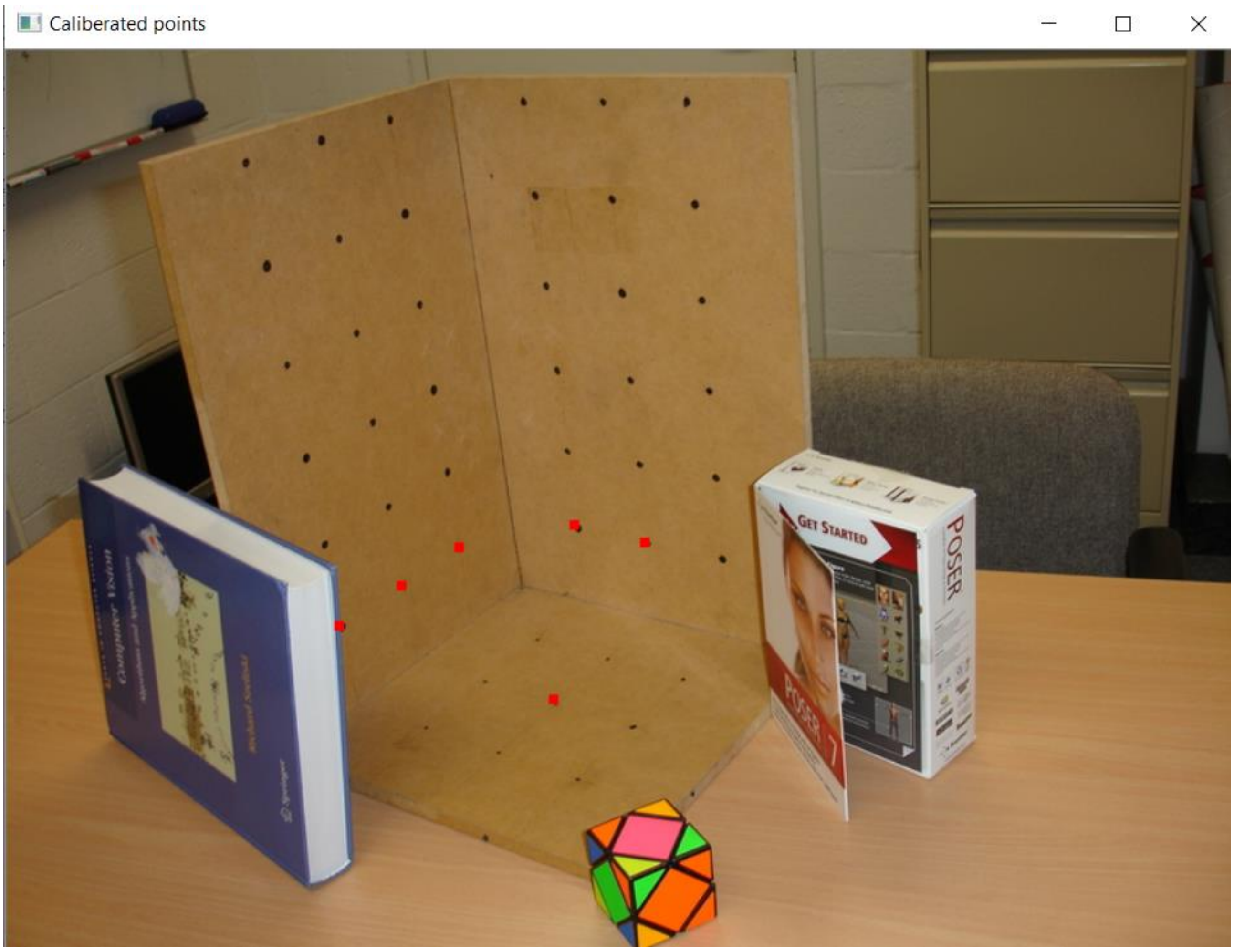
```
[[-4.32870755e+01, -5.70142520e-01, 6.63704679e+00, 3.10323502e+02]
 [-4.06735350e+01, -3.23403747e-01, 1.37639413e+00, 2.89001637e+02]
 [-1.40141067e-01, -1.56259708e-03, 5.13504437e-04, 1.00000000e+00]]
```

So, when the calibrated matrix is multiplied with the xyz points, we get new u and v values and those are the calibrated one's. So, we have to make sure that the values match with the initial uv points closely providing us with the least error of the calibration.

So, when we plot the uv values, we get



And the calibration is done, the new u and v is



The red boxes are the new u and v and we see that it matches with the ginput. So, we can say that the calibration works fine with very less error.

The error that we get is:

MSE error - 0.72949345027267

4. **Decompose the C matrix into K,R,t such that C = K[R|t], by using the following provided code. List the result in the pdf[3]**

So, when the C matrix is extracted from the calibration function, we put that calibrated matrix into the function named "vgg_KR_from_P" provided to us initially for the decomposition. So, when we put that matrix into the function, it returns the variables K, R and t.

```
K = [[ 59.12308996, -13.91435458, 320.57878884]
       [0, 32.13317456, 285.43871906]
       [0, 0, 1]]

R = [[ 0.72072732, -0.541218, -0.43316881]
      [ 0.28684872, -0.33602111, 0.89711071]
      [-0.63108633, -0.77082611, -0.0869319]]

T = [8.21066028, 9.55161806, 0.08873257]
```

5. **Answer the following questions:**

   i) **What is the focal length of the camera? [3]**

   So, $f$ = 59.12308996.

   This is because in the general K matrix,

   $$K = \begin{bmatrix} \alpha & \Upsilon & u \\ 0 & \beta & v \\ 0 & 0 & 1 \end{bmatrix}$$

   And $\alpha$ = focal length.

   So, from this we can deduce that the focal length is [0,0] index of K matrix

   ii) **What is the pitch angle of the camera with respect to the X-Z plane in the world coordinate system? (Assuming the X-Z plan is the ground plane, then the pitch angle is the angle between the camera's optical axis and the ground plane) [3]**

   Pitch angle can be calculated from the formula:

   $$\beta = \tan^{-1}(-r_{31}/\sqrt{r_{32}{}^2 + r_{32}{}^2})$$

   In this formula, r values are from the R matrix that was decomposed from the calibration matrix.

   As, we know that R matrix is

   R = [[ 0.72072732, -0.541218, -0.43316881]
   [ 0.28684872, -0.33602111, 0.89711071]
   [-0.63108633, -0.77082611, -0.0869319]]

   So, using the formula and the matrix, β = 39.13

## Task2: Two view DLT based homography estimation

1. **List your source code for homography estimation and display the two images and the location of six parts of selected parts. [3]**

```python
def homography(u2Trans, v2Trans, uBase, vBase):

    new_H_array = []
    variable_n = 6

    for i in range(variable_n):
        new_H_array.append([0,0,0,-u2Trans[i],-v2Trans[i],-1,vBase[i]*u2Trans[i], vBase[i]*v2Trans[i], vBase[i]]) #Appending the first line of the array required
        new_H_array.append([u2Trans[i],v2Trans[i],1,0,0,0,-uBase[i]*u2Trans[i],-uBase[i]*v2Trans[i],-uBase[i]]) #Appending the second line of the array required

    new_H_array = np.asarray(new_H_array) # Converting the list to the array
    S,V,D = np.linalg.svd(new_H_array) #Getting the values using single value decomposition
    normalizing = D[-1,-1] #  Normalizing the values
    H = D[-1,:]/normalizing ## Getting the H
    H = np.asarray(H) # Convert H to an array
    H = H.reshape((3,3)) # Rehaping the H array to the required shape
    return H
```

Any two images of the same planar surface in the space are related by a homography [2]. In homography, we have 2 viewpoints whereas in the calibration, we just have one view point.
So, in this, we select the same points from the different views. For example, in the given below images, we can see that the images have the same points selected and these points are selected using ginput. After that for each image, we get the u and v value respectively which then becomes the parameters to be passed to the homography function. So, in this algorithm, we should select at least 4 points so as to calculate the homography matrix. After that, as we assemble the needed rows and append in the list. The formula which is used in the list is:

$$\begin{bmatrix} 0 & 0 & 0 & -u\_trans & -v\_trans & -1 & v_{base} * u\_trans & v_{base} * v\_trans & v_{base} \\ -u\_trans & -v\_trans & -1 & 0 & 0 & 0 & u_{base} * u\_trans & u_{base} * v\_trans & u_{base} \end{bmatrix}$$

The values of uv are for the “Right” image and those points are splitted into uBase and vBase and the values of uv1 are for the “Left” image and those points are splitted into u2Trans and v2Trans. The selected values are given below:

uBase:

[127.37229437, 130.37554113, 355.61904762, 357.62121212, 221.47402597,
264.52056277]

vBase:

[147.05952381, 248.16883117, 248.16883117, 146.05844156, 179.09415584,
177.09199134]

uTrans:

[129.37445887, 132.37770563, 306.56601732, 313.57359307, 187.43722944,
216.46861472]

vTrans:

[109.01839827, 206.12337662, 261.18290043, 118.02813853, 147.05952381,
151.06385281]

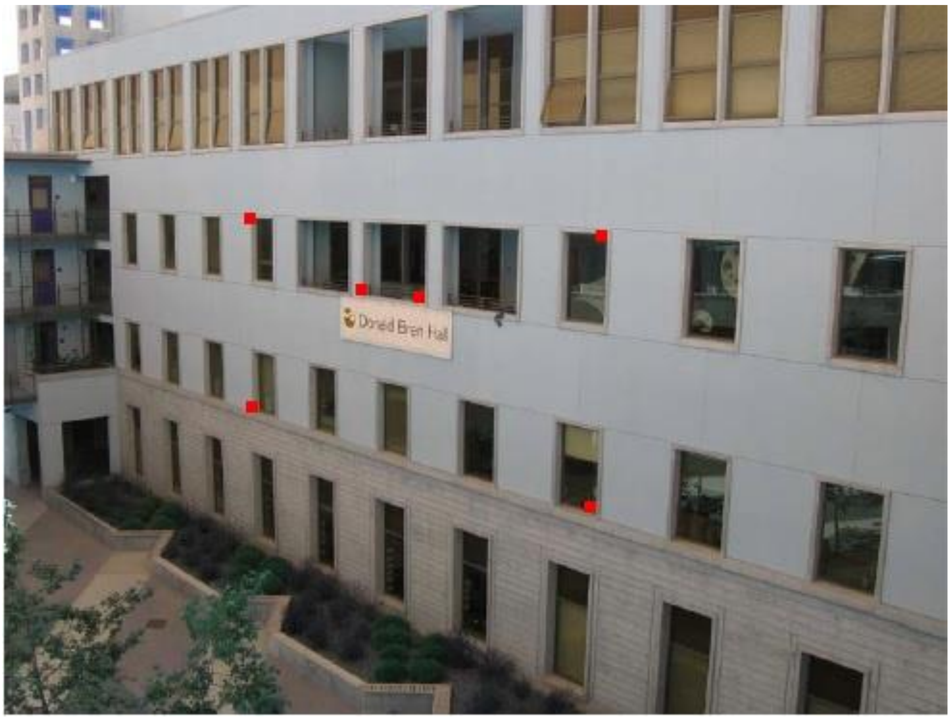


Fig 2.1
(Selected trans image (Left image))

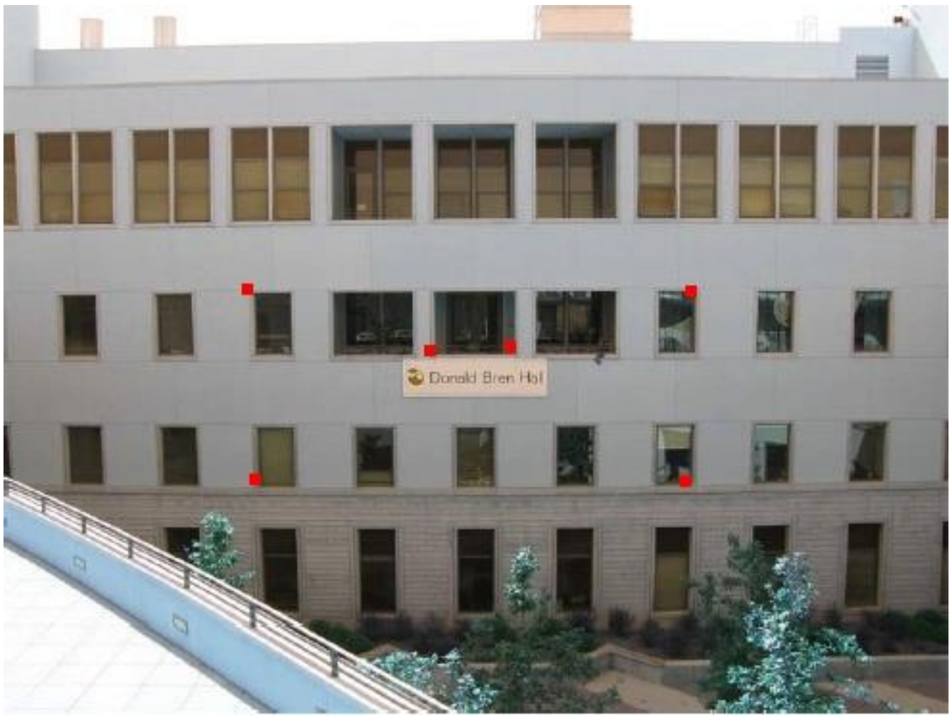


Fig 2.2
(Selected base image (Right image))

**2. List the 3x3 camera homography matrix, H that you have calculated [3]**

Homography matrix have calculated using the function provided above. So, the result that we get is:

```
[[ 3.18436600e+00, -5.23981777e-02, -2.15470773e+02]
 [ 5.12862968e-01, 1.36622794e+00, 3.33629909e+00]
 [ 3.86402059e-03, -3.64123247e-04, 1.00000000e+00]]
```

3. **Wrap the left image according to the calculated homography. Study the factors that affect the rectified results, e.g., the distance between the selected points. [3]**

In this, we use the “left” image for the warping. Warp perspective function has been used to warp the source image and 450 x 450 is defined as the shape of the image formed.

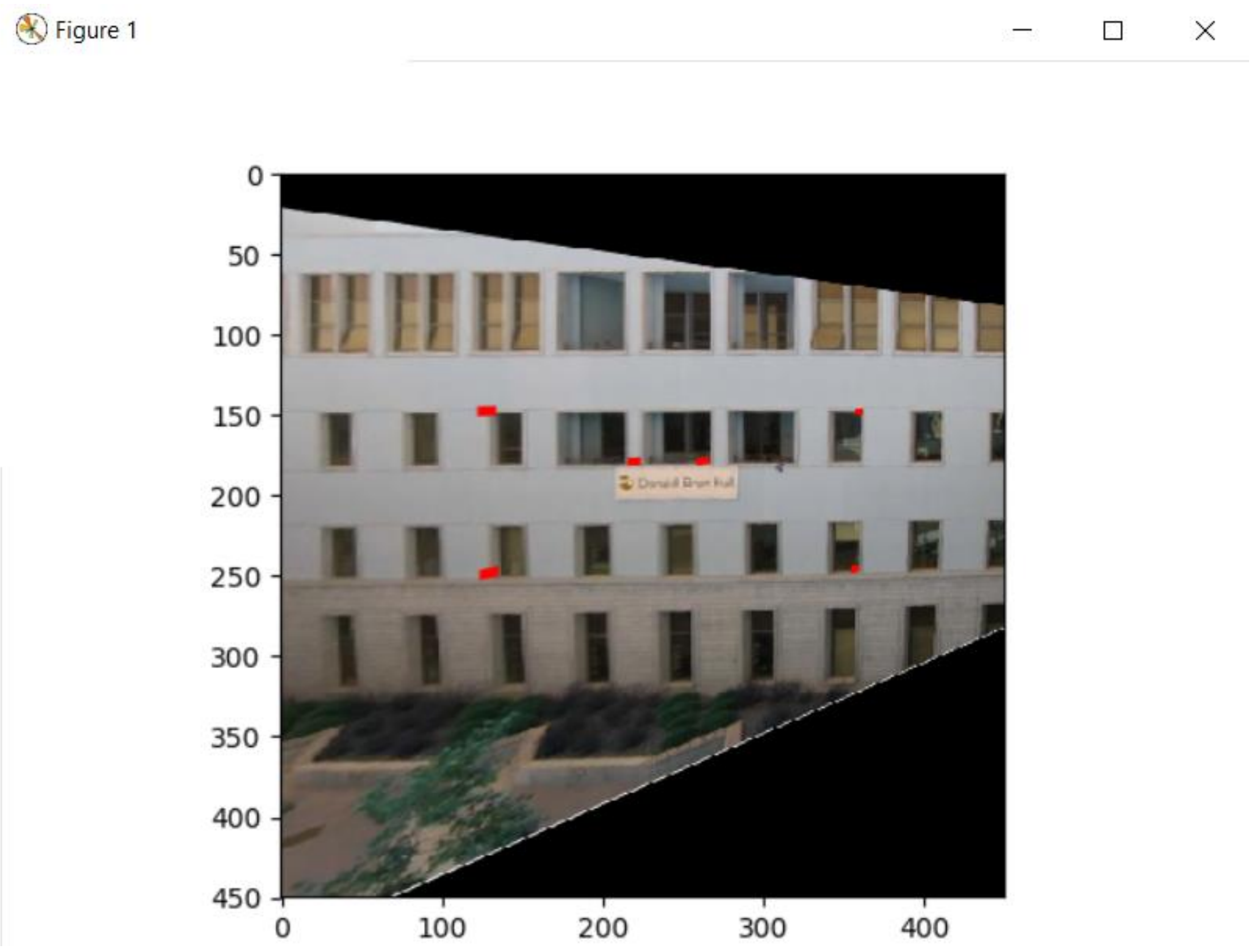


Fig 2.3
(Warped image with the points displayed)

Factors that affect the rectified results are:

**The number of points we choose or the number points used as ginput.**

So, when the number of points is changed, then it gives different results. So, to prove this point, I have taken 4 points as the ginput instead of 6 points. Those four points which are selected are shown below:

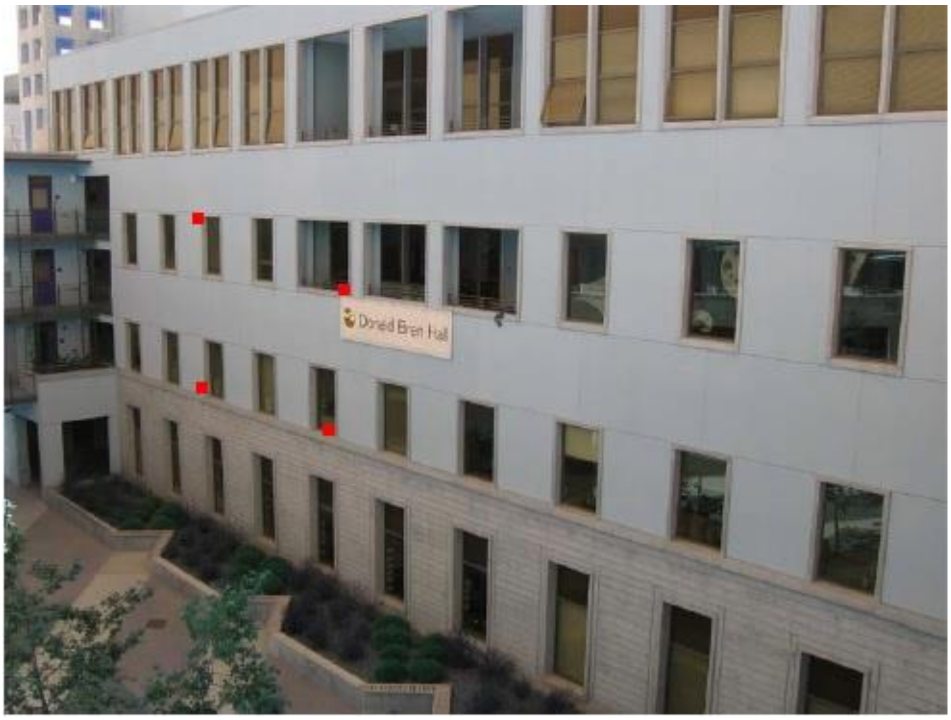


Fig 2.4

(Selected trans image with 4 ginputs)

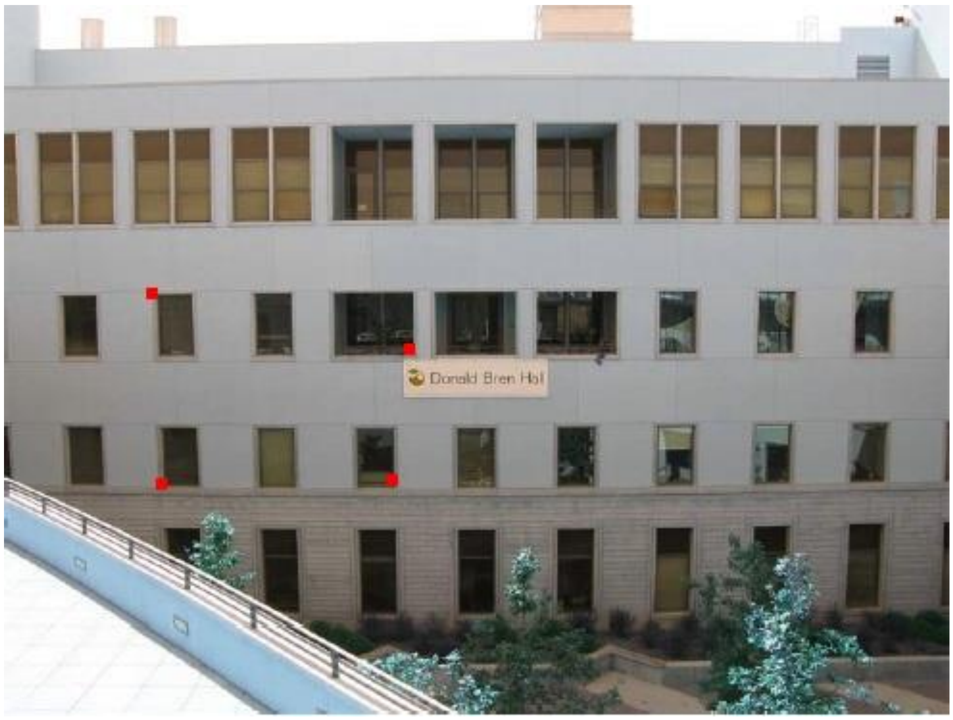


Fig 2.5

(Selected base image with 4 points)

So, after giving the ginputs into the homography function, we get a homography matrix which is shown below:

```
[[ 3.80263427e+00, -2.32928073e-02, -2.69359243e+02]
 [ 7.36930741e-01, 1.61939025e+00, -2.76692593e+01]
 [ 5.48306168e-03, -4.04264824e-04, 1.00000000e+00]]
```

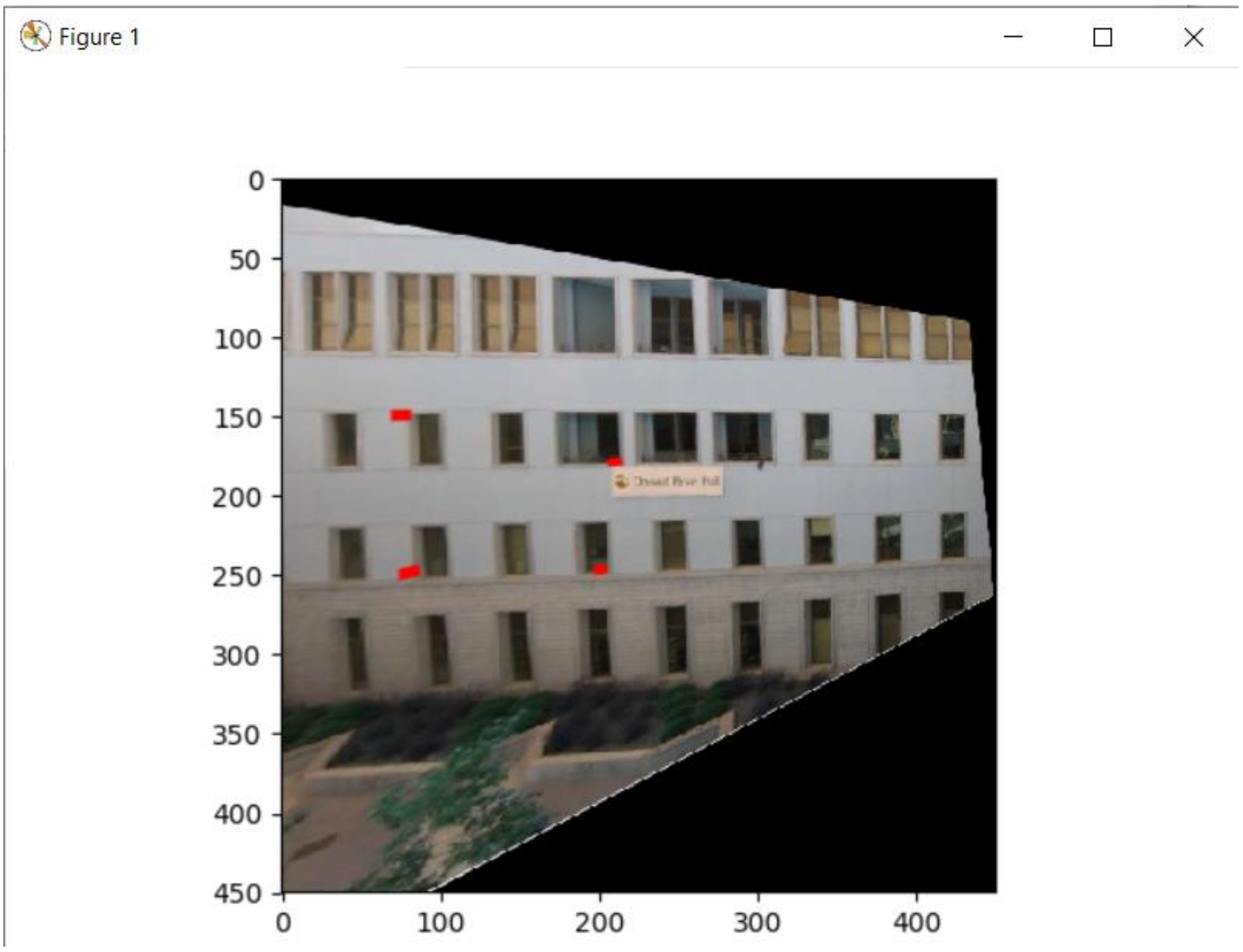


Fig 2.6

(Wrapped image for 4 ginputs)

So, we can say that the results differ when the different number of points are used.

**The distance between the selected points.**
So, when we change the distance between the points, the warp image as well as the homography matrix.
In the below image, we can see that the distance between the points have been decreased.

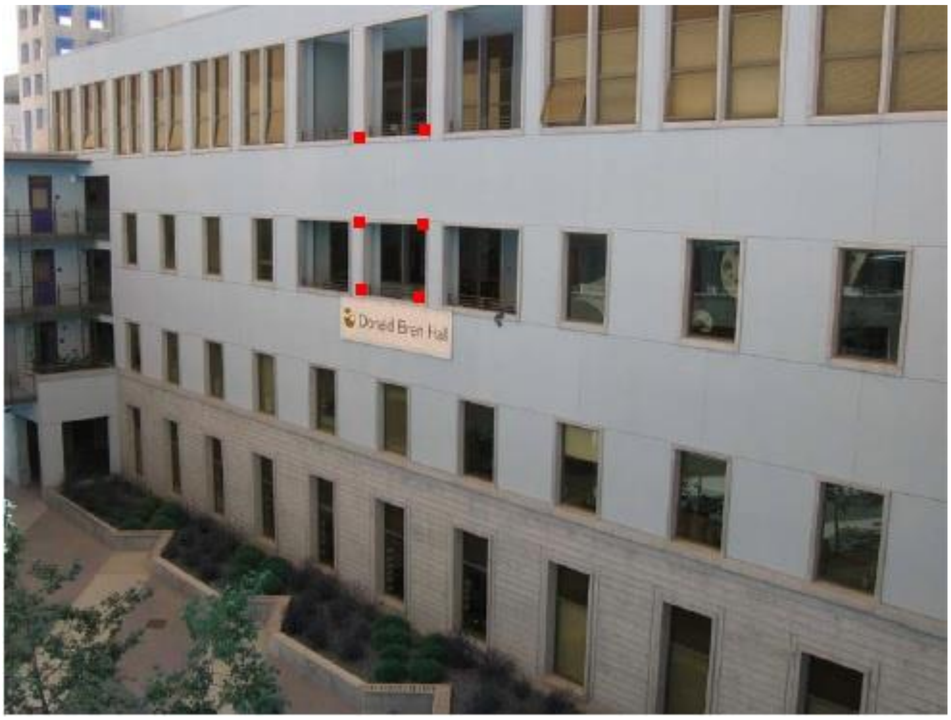


Fig 2.7

(Selected “Left” image with 6 ginputs with short distance)

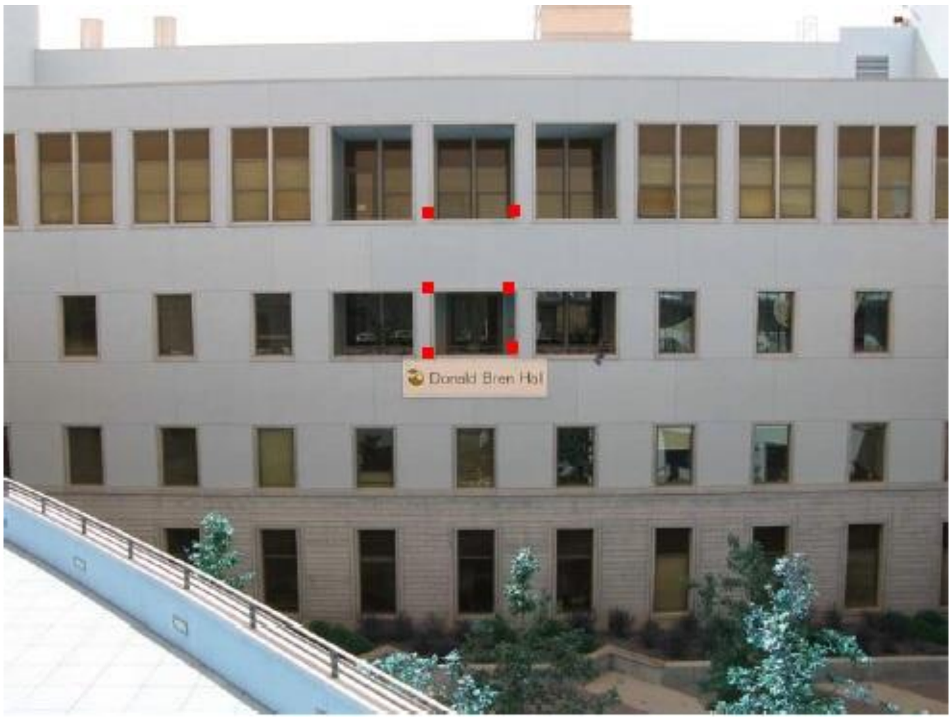


Fig 2.8

(Selected “Right” image with 6 ginputs with short distance)

When these points are used, then the we get the following homography matrix

[[-1.47084825e+01, 3.67814638e-01, 1.66672853e+03]

 [-4.24273220e+00, -4.33840927e+00, 5.89545841e+02]

 [-3.14722668e-02, 1.64586901e-03, 1.00000000e+00]]

This is homography matrix is different from the homography matrix when we considered distant points.

Moreover, the image that we get after warping is shown below which is different from the image that we have got from the above

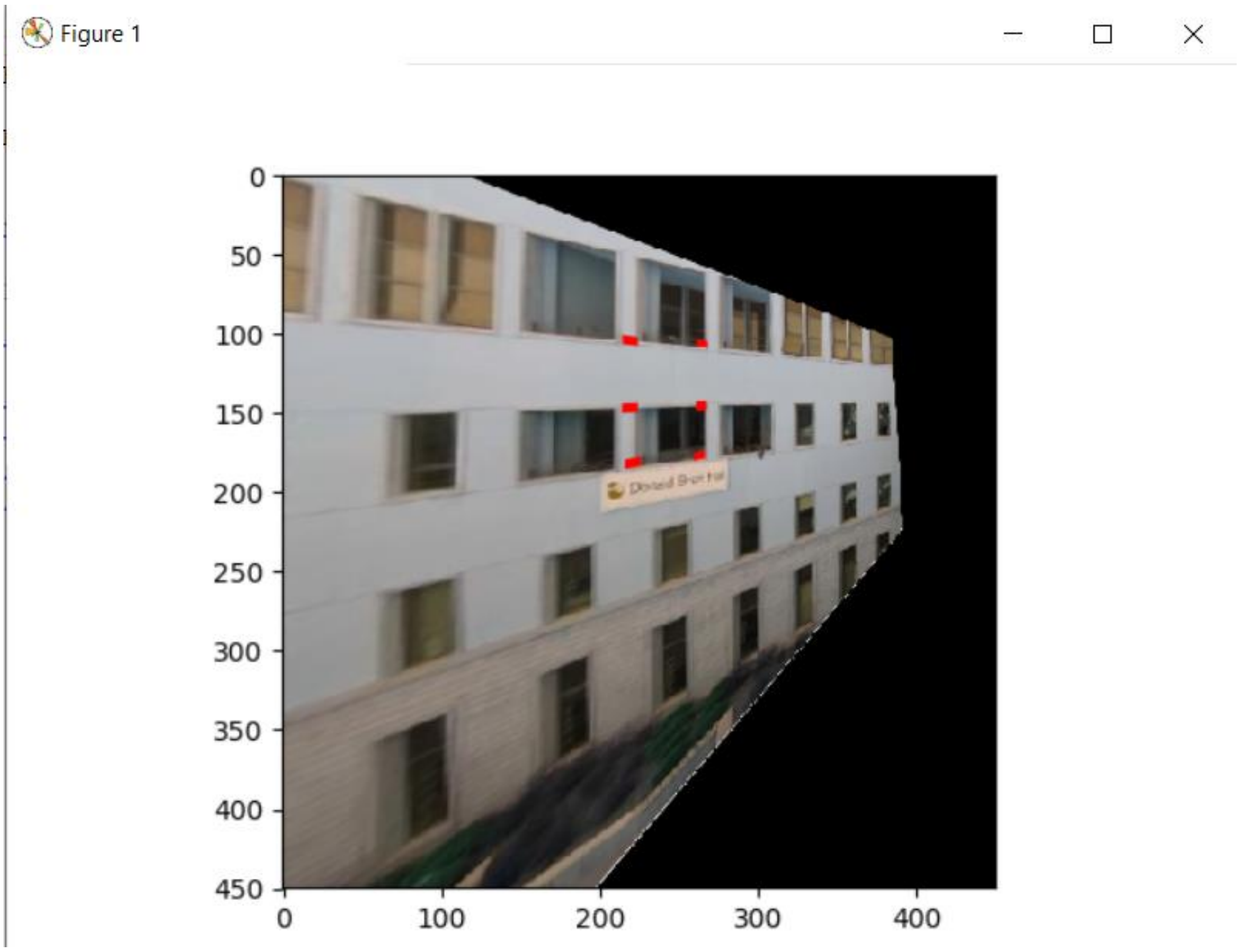


Fig 2.9

(Wrapped image with small distance keeping the ginput constant)

So, from this, we can conclude that the distance between the points changes the warping image as well as homography matrix

**References:**

[1] https://en.wikipedia.org/wiki/Direct_linear_transformation

[2] https://en.wikipedia.org/wiki/Homography_(computer_vision)

[3] Assignment Pdf provided.