# EEE: 515 - Machine Vision - Homework 3

Praveen Paidi
Student ID: 1225713099
Fall 2023

November 24, 2023

**Problem status**

1. **Problem 1 : Completed**
2. **Problem 2 : Completed**
3. **Problem 3 : Completed**
4. **Problem 4 : Completed**
5. **Problem 5 : Completed**

# 1 Problem 1

## 1.1 Part 1

The line is y = mx+b ;
Rewriting the equation in the form of -mx + y - b = 0

Converting equation into matrix $\begin{bmatrix}-m & 1 & -b\end{bmatrix}.\begin{bmatrix}x\\y\\1\end{bmatrix} = 0$ which is comparable to $l^T.X = 0$.

Heterogeneous coordinate of $X = \begin{bmatrix}x\\y\end{bmatrix}$ and Homogeneous coordinate of $X = \begin{bmatrix}x\\y\\1\end{bmatrix}$.

Matrix $l^T = \begin{bmatrix}-m & 1 & -b\end{bmatrix}$.

## 1.2 Part 2

Given point on 2D plane is $\begin{bmatrix}3\\5\end{bmatrix}$.

This is heterogeneous point on 2D plane, Converting into homogeneous point $\begin{bmatrix}3z\\5z\\z\end{bmatrix}$. Taking z ¿ 0, for example z = 1 , Homogeneous point is $\begin{bmatrix}3\\5\\1\end{bmatrix}$ Taking z ¡ 0, for example z = -1,Homogeneous point is $\begin{bmatrix}-3\\-5\\-1\end{bmatrix}$

Reconverting them would result in same heterogeneous point $\begin{bmatrix}3\\5\end{bmatrix}$.

## 1.3 Part 3

Two homogeneous lines are $l_1$ and $l_2$.Let's suppose the intersection point of the lines is x.
As the line can be written in homogeneous as following, $l_1^T.x = 0$, $l_2^T.x = 0$
Since $l_1^T.x = 0$ and $l_2^T.x = 0$ x has to be orthogonal to both the $l_1$ and $l_2$ and that is $l_1$ x $l_2$

By taking an example;
$l_1 = a_1x + b_1y + c_1$; $l_2 = a_2x + b_2y + c_2$
Taking the cross product would give

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} b_1 \cdot c_2 - b_2 \cdot c_1 \\ -a_1 \cdot c_2 + a_2 \cdot c_1 \\ a_1 \cdot b_2 - b_1 \cdot a_2 \end{bmatrix}$$

Substituting the point on both lines giving a result of 0, which says point lies on both lines intersection.

## 1.4 Part 4

The first line is x + y - 5 = 0 and the second line is 4x - 5y +7 = 0
The intersection is given by the cross product of the lines.

Given vectors $\mathbf{a} = \begin{bmatrix}1\\1\\-5\end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix}4\\-5\\7\end{bmatrix}$, the cross product $\mathbf{a} \times \mathbf{b}$ can be calculated as:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 1 \cdot (-5) - 1 \cdot 4 \\ 1 \cdot 7 - (-5) \cdot (-5) \\ 1 \cdot (-5) - 1 \cdot 4 \end{bmatrix} = \begin{bmatrix}-18\\-27\\-9\end{bmatrix}$$

The homogeneous coordinates are $\begin{bmatrix}-18\\-27\\-9\end{bmatrix}$ Then converting into Cartesian coordinates are $\begin{bmatrix}2\\3\end{bmatrix}$

### 1.5 Part 5

The standard form of the line is ax + by +c = 0. any parallel line to this one can be expressed in the form of same slope and different intercept. Let's say dax+dby+k=0 is the equation where k is a constant and d is scalar value multiplying to x and y coefficient making sure still slope is same.

Taking the cross product of the both the equations to find the intersection of the two lines.

$$\textbf{Givenline} \times \textbf{Parallelline} = \begin{bmatrix} b \cdot (k) - c \cdot (db) \\ a \cdot (k) - (c) \cdot (ad) \\ a \cdot (db) - (da) \cdot (b) \end{bmatrix} = \begin{bmatrix} ab[k - cd] \\ -ab[k - cd] \\ 0 \end{bmatrix}$$

The homogeneous coordinates are the $\begin{bmatrix} ab[k - cd] \\ -ab[k - cd] \\ 0 \end{bmatrix}$

Converting them into Cartesian space makes it points to meet infinity in x and y direction by making them as $\begin{bmatrix} ab[k - cd]/0 \\ -ab[k - cd]/0 \end{bmatrix}$

### 1.6 Part 6

For a line l to pass through x1 and x2, both x1 and x2 must lie on the line l.
As the line can be written in homogeneous as following, $l^T.X_1 = 0$, $l^T.X_2 = 0$

The vector l must be orthogonal to both $X_1 and X_2$, such a vector is $X_1 \times X_2$.
So, the line passing through points $X_1 and X_2$ is $l = X_1 \times X_2$.
Suppose the $X_1 = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$ and $X_2 = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$
The cross product would result in $\begin{bmatrix} y_1 \cdot 1 - 1 \cdot y_2 \\ x_1 \cdot 1 - x_2 \cdot 1 \\ x_1 \cdot y_2 - x_2 \cdot y_1 \end{bmatrix}$
The line equation would be a$(y_1 - y_2)$+ b$(x_1 - x_2)$ + c$(x_1.y_2 - x_2.y_1) = 0$

**Other method**:
Assuming both points to be on the same line, slope of the line will be $= \frac{y_2 - y_1}{x_2 - x_1}$
Two point line equation is $y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}.\ x - x_1$
Final equation is $(x_2 - x_1).y$ - $(y_2 - y_1).x$ +$(x_1.y_2 - x_2.y_1) = 0$

## 2 Problem 2

### 2.1 Part 1

Homogeneous rotation matrix for rotation of angle theta around the point (a,b).
**1 st step: Translation of Origin:**
Translate the system such that the point (a,b) becomes the new origin.
To translate (a,b) to origin the translation matrix would b e $\begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix}$
**2nd Step : Rotation of angle $\theta$ :**
The rotation matrix for angle $\theta$ is $\begin{bmatrix} cos(\ \theta) & -sin(\ \theta) & 0 \\ sin(\ \theta) & cos(\ \theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$
**3rd Step: Translate back to original origin:**
Translate the point by (a, b) w.r.t original origin with translation matrix $\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix}$

Putting up everything
Pre multiplying the rotation matrix got in step 2 with the translation matrix got in step 1 and then pre multiplying with back Translation mayrix got in step 3.

$$\begin{bmatrix}1 & 0 & a\\ 0 & 1 & b\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}cos(\ \theta) & -sin(\ \theta) & 0\\ sin(\ \theta) & cos(\ \theta) & 0\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}1 & 0 & -a\\ 0 & 1 & -b\\ 0 & 0 & 1\end{bmatrix}$$

Final rotation matrix is given by : $\begin{bmatrix}cos(\ \theta) & -sin(\ \theta) & -a.cos(\ \theta) + \text{b.sin}(\theta)\text{+a}\\ sin(\ \theta) & cos(\ \theta) & -a.sin(\ \theta) \text{ - b.cos}(\theta)\text{+b}\\ 0 & 0 & 1\end{bmatrix}$

## 2.2 Part 2

The four vertices of the square are : p1 = $\begin{bmatrix}1\\1\end{bmatrix}$ p2 = $\begin{bmatrix}2\\1\end{bmatrix}$ p3 = $\begin{bmatrix}2\\2\end{bmatrix}$ p4 = $\begin{bmatrix}1\\2\end{bmatrix}$
Rotation about p2 through 45 degrees.

Final matrix obtained from the part 1 is $\begin{bmatrix}cos(\ \theta) & -sin(\ \theta) & -a.cos(\ \theta) + \text{b.sin}(\theta)\text{+a}\\ sin(\ \theta) & cos(\ \theta) & -a.sin(\ \theta) \text{ - b.cos}(\theta)\text{+b}\\ 0 & 0 & 1\end{bmatrix}$

$(\theta) = 45$ ; a = 2, b = 1;

Substituting gives the matrix as $\begin{bmatrix}1/1.414 & -1/1.414 & (0.414.(2)+1)/1.414\\ 1/1.414 & 1/1.414 & (-2+0.414.(1))/1.414\\ 0 & 0 & 1\end{bmatrix}$

Multiplying the homogeneous coordinates with matrix to get the transformed points.

The final rotated p1 is $\begin{bmatrix}0.707 & -0.707 & 1.292\\ 0.707 & 0.707 & -1.121\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}1.292\\0.293\\1\end{bmatrix}$

The final rotated p2 is $\begin{bmatrix}0.707 & -0.707 & 1.292\\ 0.707 & 0.707 & -1.121\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}2\\1\\1\end{bmatrix} = \begin{bmatrix}2\\1\\1\end{bmatrix}$

The final rotated p3 is $\begin{bmatrix}0.707 & -0.707 & 1.292\\ 0.707 & 0.707 & -1.121\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}2\\2\\1\end{bmatrix} = \begin{bmatrix}1.292\\1.707\\1\end{bmatrix}$

The final rotated p4 is $\begin{bmatrix}0.707 & -0.707 & 1.292\\ 0.707 & 0.707 & -1.121\\ 0 & 0 & 1\end{bmatrix} . \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}0.585\\1\\1\end{bmatrix}$

## 2.3 Part 3

ax + by + c = 0 is the equation of the line. To reflect about this line, aligning it with x axis and then reversing the parameters to attain the solution.

**First case when b not equal to 0.**
**1st Step:**
Translate to align with the x-axis: Using the point where the line intersects the x-axis to translate the entire coordinate system. This point has coordinates $\begin{bmatrix}-c/a\\0\end{bmatrix}$ which is the x-intercept

The translation matrix for this step is: $\begin{bmatrix}1 & 0 & c/a\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$

**2nd Step:**
Then perform the rotation with respect to the angle between the line and the x axis to align with x axis.
The angle $\theta$ that the line makes with the x-axis can be found using slope of the line:
$\theta = \arctan(-a/b)$

Then rotation matrix for the angle is $\begin{bmatrix}cos(\ \theta) & -sin(\ \theta) & 0\\ sin(\ \theta) & cos(\ \theta) & 0\\ 0 & 0 & 1\end{bmatrix}$.

**3rd Step:**

Now reflecting around the x axis can be obtained by substituing theta as 180,

Then reflection matrix for the angle is $\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}$.

**4th Step:**

Then restoring back it, Taking the inverse of the rotation matrix is equal to transpose of the rotation matrix.

$R^T = R^{-1}$ Then Inverse rotation matrix for the angle is $\begin{bmatrix}cos(\ \theta) & sin(\ \theta) & 0\\ -sin(\ \theta) & cos(\ \theta) & 0\\ 0 & 0 & 1\end{bmatrix}$

**5th Step:**

Then getting back the a translation of $\begin{bmatrix}-c/a\\ 0\end{bmatrix}$

The inverse translation matrix would be $\begin{bmatrix}1 & 0 & -c/a\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$

Putting up everything ;

$$\begin{bmatrix}1 & 0 & c/a\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}cos(\ \theta) & sin(\ \theta) & 0\\ -sin(\ \theta) & cos(\ \theta) & 0\\ 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}cos(\ \theta) & -sin(\ \theta) & 0\\ sin(\ \theta) & cos(\ \theta) & 0\\ 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}1 & 0 & -c/a\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

The final matrix is , $\begin{bmatrix}cos(2\theta) & -sin(\ 2\theta) & c/a(1-cos\theta)\\ -sin(\ 2\theta) & -cos(\ 2\theta) & c/a\\ 0 & 0 & 1\end{bmatrix}$

## 3 Problem 3

Affine + Second order warp.

$x^{'} = ax + by + tx + \alpha x^2 + \beta y^2$

$y^{'} = cx + dy + ty + \gamma x^2 + \theta y^2$

Converting above equations into matrix form

$$\begin{bmatrix}x\\ y\end{bmatrix} = \begin{bmatrix}a & b & t_x & \alpha.x^2 & \beta.y^2\\ c & d & t_y & \gamma.x^2 & .y^2\end{bmatrix}.\begin{bmatrix}x\\ y\\ 1\\ x^2\\ y^2\end{bmatrix}$$

Converting matrix suitable for more points in the form of X'=A.X,

$$\begin{bmatrix}x_1^{'}\\ y_1^{'}\\ x_2^{'}\\ y_2^{'}\\ .\\ .\\ x_n-1^{'}\\ y_n-1^{'}\\ x_n^{'}\\ y_n^{'}\end{bmatrix} = \begin{bmatrix}x_1 & y_1 & x_1^2 & y_1^2 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & x_1^2 & y_1^2 & 1\\ x_2 & y_2 & x_2^2 & y_2^2 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & x_2^2 & y_2^2 & 1\\ . & . & . & . & . & . & . & . & . & .\\ . & . & . & . & . & . & . & . & . & .\\ x_1 & y_1 & x_1^2 & y_1^2 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & x_1^2 & y_1^2 & 1\\ x_2 & y_2 & x_2^2 & y_2^2 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & x_2^2 & y_2^2 & 1\end{bmatrix}.\begin{bmatrix}a\\ b\\ \alpha\\ \beta\\ t_x\\ a\\ b\\ \alpha\\ \beta\\ t_y\end{bmatrix}$$

There are 6 linear column vectors. It requires at least 6 equations which makes 3 points in minimum. Linear least squares solution for that is $(A^T.A)^{-1}.A^T.X$ which gives the solution for the new points

Other method:

This can also be done using SVD as well which gives same result by picking the last column of V matrix. Applying SVD to the A matrix makes USV and picking as above mentioned gives same.

## 4 Problem 4

### 4.1 part 4.1

Construction of white irregular quadrilateral with 50 by 50 pixel size centred in the image.Translation and rotation of image by (30,100) and 45 degrees with respect to the original centre which is centre of the image.

Below is the code for the generation of the in listing 1,

Listing 1: Quadrilateral generation

```
# Create a new image with a black background
img = Image.new('RGB', (300, 300), 'black')
angles = np.sort(np.random.rand(4) * 2 * np.pi)
side_length = 50
x_coords = np.round(150 + side_length * np.cos(angles))
y_coords = np.round(150 + side_length * np.sin(angles))
centroid_x = np.mean(x_coords)
centroid_y = np.mean(y_coords)
x_coords = x_coords - centroid_x + 150
y_coords = y_coords - centroid_y + 150
points = list(zip(x_coords, y_coords))
draw = ImageDraw.Draw(img)
draw.polygon(points, fill='white')
```

Below is the code for the translation of (30, 100) and then rotation of 45 degrees by the centre of the image which is the same as the original centre in listing 2.

The translation matrix is $\begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix}$

Angle given to rotate is 45 degrees which cv2 takes as anti clock wise and consider the angle as -45 and performs the rotation matrix internally from cv2 function in listing 2 as $\begin{bmatrix} 0.707 & 0.707 & -62.13 \\ -0.707 & 0.707 & 150 \\ 0 & 0 & 1 \end{bmatrix}$

Reference for angle in cv2 automatic consideration of anti clockwise is :link

The final matrix for the transformation is multiplied matrix of rotation to translation matrix resulting in following shown : $\begin{bmatrix} 0.707 & 0.707 & 29.79 \\ -0.707 & 0.707 & 199.49 \\ 0 & 0 & 1 \end{bmatrix}$

Listing 2: Translation and Rotation

```
# Define the translation matrix
translation_matrix = np.float32([[1, 0, 30], [0, 1, 100]])
# Apply the translation
translated_image = cv2.warpAffine(original_array, translation_matrix, (300, 300))
# Define the rotation matrix
center = (150, 150)
rotation_matrix = cv2.getRotationMatrix2D(center, 45, 1)
# Apply the rotation
rotated_image = cv2.warpAffine(translated_image, rotation_matrix, (300, 300))
# Convert the result back to an Image object
result_image = Image.fromarray(rotated_image)
```

Random quadrilateral generated and the (rotated + translated ) quadrilateral images are shown in the figure 1 and the figure 2.

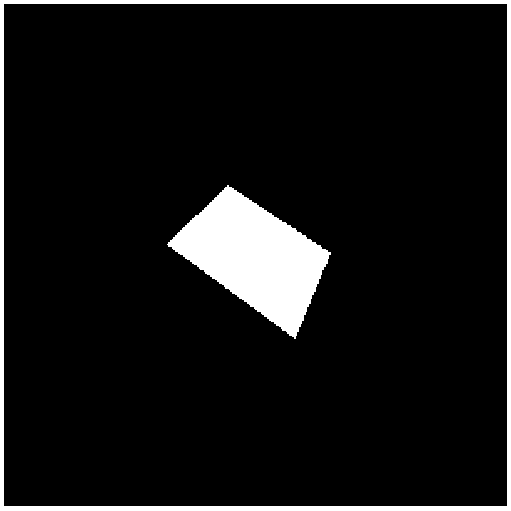
Figure 1: Random quadrilateral centred

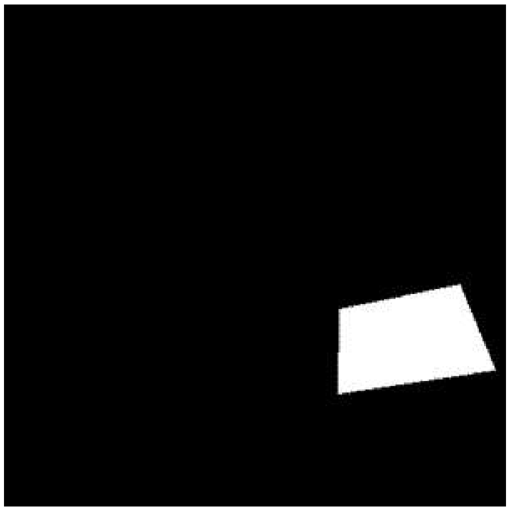
Figure 2: Translated and then rotated

### 4.2 part 4.2

Harris corner implemented for the detection of the corners in the quadrilateral in both the figure 1(Original quadrilateral) and the figure 2(Translated+ rotated image)

Below is the code for implementation using in 5 steps as sobel detection , removing mean and variance, computing covariance matrix, computing the eigen values and then applying the threshold at the final step in a nutshell.

Listing 3: Harris corners

```
I_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
I_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
''' 2nd step , taking the mean out '''
I_x -= np.mean(I_x)
I_y -= np.mean(I_y)
''' 3rd step , computing the covariance matrix '''
I_x2 = I_x ** 2; I_y2 = I_y ** 2; I_xy = I_x * I_y
''' 4th step calculting eigen values and eigen vectors for M matrix '''
height, width= image.shape;offset = 2
R = np.zeros_like(image, dtype=np.float32)
for y in range(offset, height - offset):
    for x in range(offset, width - offset):
        window_I_x2 = I_x2[y - offset:y + offset + 1, x - offset:x + offset + 1]
        window_I_y2 = I_y2[y - offset:y + offset + 1, x - offset:x + offset + 1]
        window_I_xy = I_xy[y - offset:y + offset + 1, x - offset:x + offset + 1]
        M = np.array([[np.sum(window_I_x2), np.sum(window_I_xy)],
                      [np.sum(window_I_xy), np.sum(window_I_y2)]])
        eigvals = np.linalg.eigvals(M)
        R[y, x] = min(eigvals)*max(eigvals) -k* ((min(eigvals) + max(eigvals))**2)
'''5th step corners finding my applying the thresshold as 1000 '''
corners = (R > threshold).nonzero()
```

Applied the threshold on harris corners to able to detect the corners in the both images as shown in the following figure 3 and figure 4.

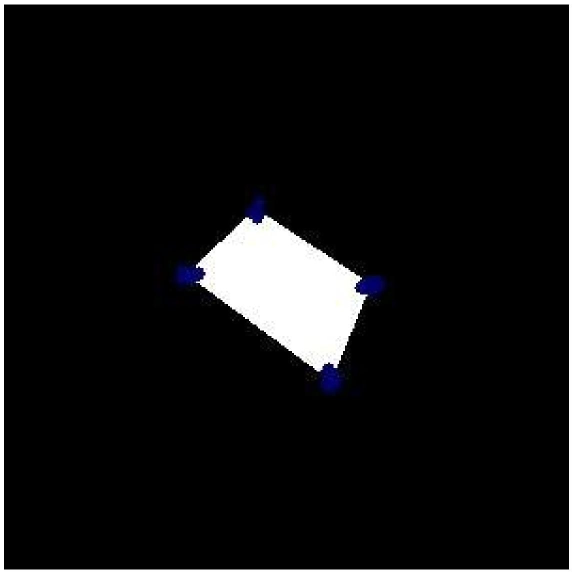

Figure 3: Harris corners for original image

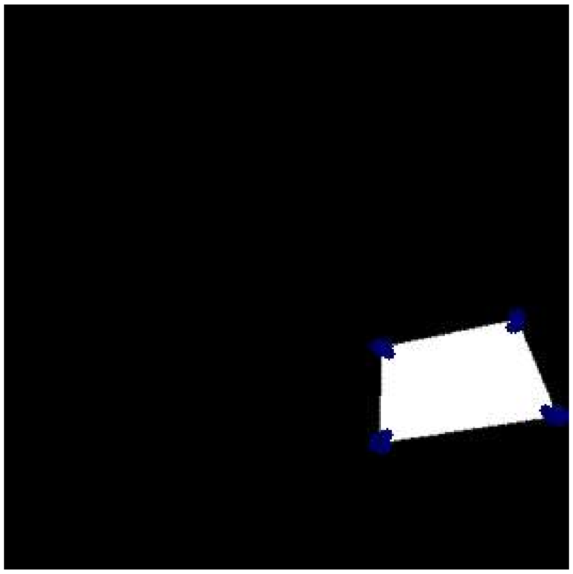

Figure 4: Harris corners of rotated image

From the corners achieved in the both images, Those corners are passed to SIFT algorithm defined below in listing 4 , to develop the feature descriptors.Some of the hyper parameters considered for the SIFT are :
Patch size = 22 around each corner
Gaussian blur kernel size = 7
Descriptor vector size = (patch size * patch size) without down sampling.

Listing 4: SIFT Algorithm

```
n_descriptors = []
x = corners[:,1]
y = corners[:,2]
for i in range(n_best):
    y_i = x[i]
    x_i = y[i]
    gray = np.pad(img, ((patch_size,patch_size), (patch_size,patch_size)))
    x_start = int(x_i + patch_size/2)
    y_start = int(y_i + patch_size/2)
    # creating feature descriptor 40x40 descriptor of one point
    descriptor = gray[x_start:x_start+patch_size, y_start:y_start+patch_size]
    # Applying gaussian blur on the descriptor
    descriptor = cv2.GaussianBlur(descriptor, (7,7), cv2.BORDER_DEFAULT)
    descriptor1 = descriptor.reshape((patch_size*patch_size,1))
    std = descriptor1.std()
    # to remove illumination invariance.
    descriptor_standard = (descriptor1 - descriptor1.mean())/std
    n_descriptors.append(descriptor_standard)
```

From the feature vectors derived from the SIFT algorithm, Lowe's algorithms applied for feature matching as shown in listing 5. Match ratio for the Lowe's algorithm is 0.8.This 0.8 is chosen as hyperparameter for better matching of the corners from original to rotated image. Each descriptor is taken from original image and euclidean distance is measured with all other descriptors in transformed image.The ratio between best match to second best match according to algorithm is hyperparameter. Corresponding matches in lowe's algorithm are taken by sorting them in ascending order of error.

With the output of the feature matching list of corners in both original and translated+rotated image are taken and corresponding matches are drawn using cv2 function mentioned in listing 6.

Listing 5: Feature Matching by Lowe's Algorithm

```
f1 = Descriptors_image1
f2 = Descriptors_image2
matched_pairs = []
for i in range(0, len(f1)):
    sqr_diff = []
    for j in range(0, len(f2)):
        # comparing each corner to every other corner in next image by taking the
           difference between the descriptors
        diff = np.sum((f1[i] - f2[j])**2)
        sqr_diff.append(diff)
    # converting  into array
    sqr_diff = np.array(sqr_diff)
    diff_sort = np.argsort(sqr_diff)
    sqr_diff_sorted = sqr_diff[diff_sort]
    if (sqr_diff_sorted[1])==0:
        sqr_diff_sorted[1] = 0.00001
    # applying lowe's algorithm to check the matching
    ratio = sqr_diff_sorted[0]/(sqr_diff_sorted[1])
    if ratio < match_ratio :
        matched_pairs.append((corners1[i,1:3], corners2[diff_sort[0],1:3]))
```

I got 15 matches from original to transformed quadrilateral.Those are shown in figure 5.

Listing 6: Drawing matches

```
out = cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches1to2, None, flags
    =2)
```

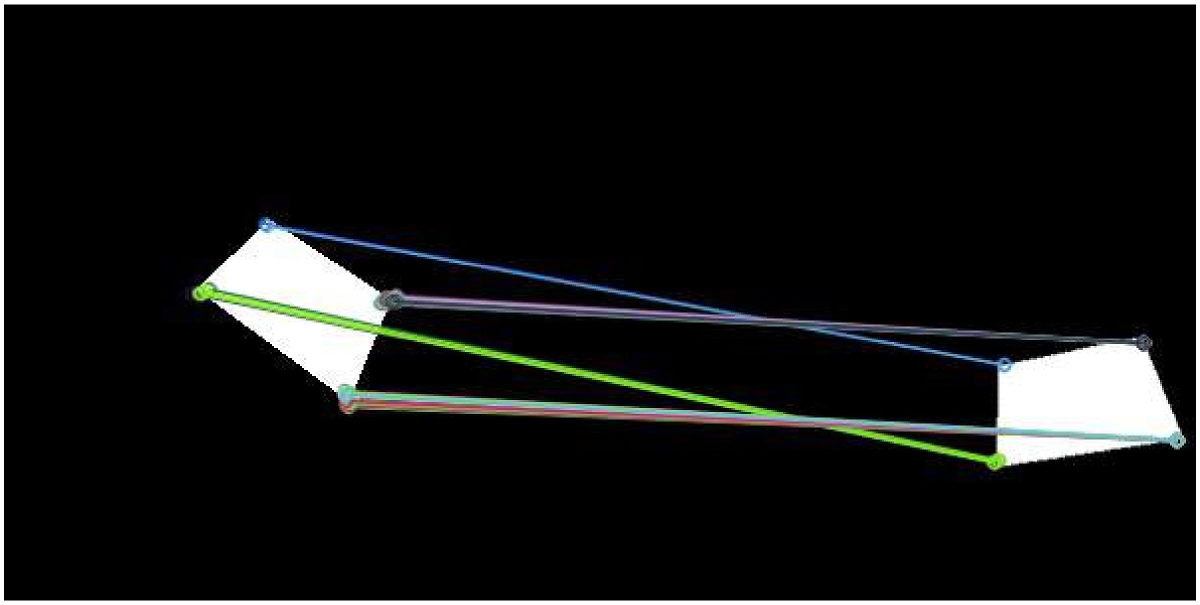

Figure 5: Matched features drawing in both images

**OTHER ASPECTS:**

Hyper parameters can be changed for better performance. Harris corners threshold can be further tuned with respect to each image and corners can be detected in better number.

Patch size can be varied for better making feature descriptor in SIFT algorithm.Down sampling can be done to detect better as well.

Instead of Lowe's algorithm Brute force algorithm can be used as well for better matching, Although I tried I'm getting very similar results.

## 4.3 Part 4.3

Picking a few samples from the matched pairs of 15 and then forming for the least squares.

The translation matrix derived from the 4.1 part is $\begin{bmatrix} 0.707 & 0.707 & 29.79 \\ -0.707 & 0.707 & 199.49 \\ 0 & 0 & 1 \end{bmatrix}$

The minimum number of points required for the least squares are derived from the degrees of freedom.

The linear independent columns in the matrix are 3 (3 DOF ).Therefore the minimum points required are:2

Least squares application in order to retrieve the translation and rotation matrix applied.

$$x^{'} = x.cos(\theta) + y.sin(\theta) + tx.cos(\theta) + ty.sin(\theta)$$

$$y^{'} = -x.sin(\theta) + y.cos(\theta) + tx.sin(\theta) + ty.cos(\theta)$$

Formulating the least square from the above equations for n number of points. X'=A.X,

$$\begin{bmatrix} x_1^{'} \\ y_1^{'} \\ x_2^{'} \\ y_2^{'} \\ . \\ . \\ x_n - 1^{'} \\ y_n - 1^{'} \\ x_n^{'} \\ y_n^{'} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ . & . & . & . & . & . \\ & . & . & . & . & . \\ x_n - 1 & y_n - 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n - 1 & y_n - 1 & 1 \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} . \begin{bmatrix} cos\theta \\ sin\theta \\ t_x.cos\theta + t_y.sin\theta \\ -sin\theta \\ cos\theta \\ t_y.cos\theta - t_x.sin\theta \end{bmatrix}$$

Linear least squares solution for that is $(A^T.A)^{-1}.A^T.X$

Below is the code formulated for the setup of the least squares solution shown in listing 7.

Listing 7: Least sqaures

```
for i in range(0,len(Points1)):
    A[2*i, :] = [Points1[i][0], Points1[i][1], 1, 0, 0, 0]
    A[j, :] = [0, 0, 0, Points1[i][0], Points1[i][1], 1]
    b[2*i] = Points2[i][0]
    b[j ] = Points2[i][1]
    j=j+2
np.dot(np.dot(np.linalg.inv(np.dot(A.T, A)),A.T),b)
```

I picked a subset of 20 points to obtain better solution.

The matrix obtained from the least square solution is $\begin{bmatrix} 0.72 & 0.684 & 30.84 \\ -0.682 & 0.714 & 194.2 \\ 0 & 0 & 1 \end{bmatrix}$

This is close to the original matrix in part 4.1.

OTHER ASPECTS:

Although number of points required are only 2. The more number of points the solution is more robust.It can be also solved using SVD as well.

There are marginal error in solution due to corner point detection threshold, match ratio and patch size hyper parameters used above. It can be further done better by tuning a bit more.

## 5 Problem 5

### 5.1 Part 5.1

The following are the four images chosen for the task.

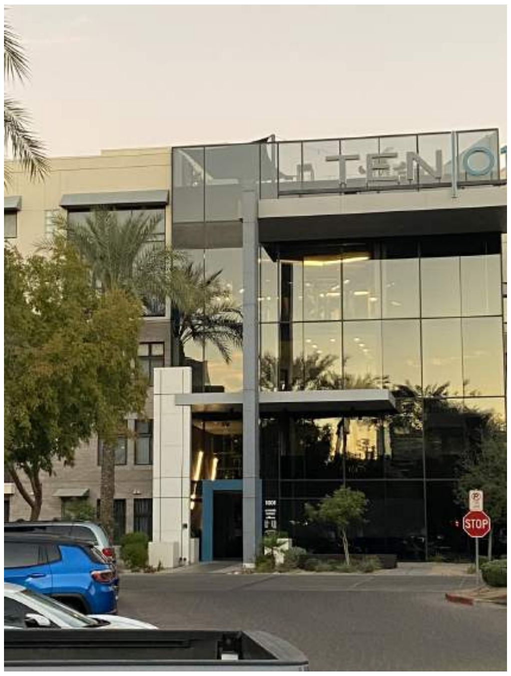

Figure 6: First image

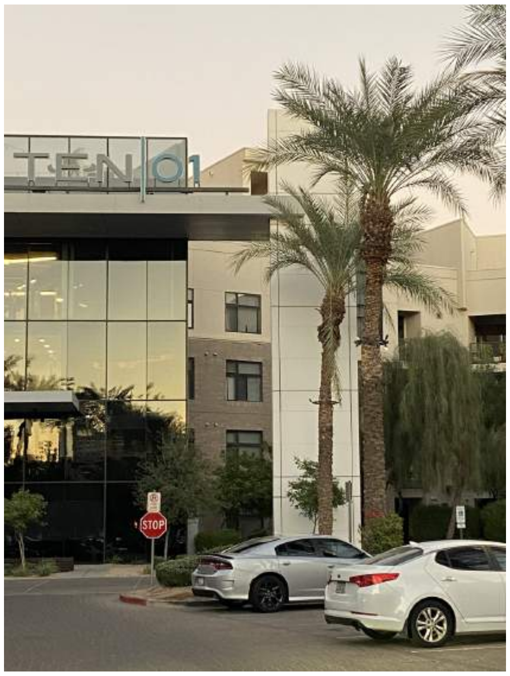

Figure 7: Second image

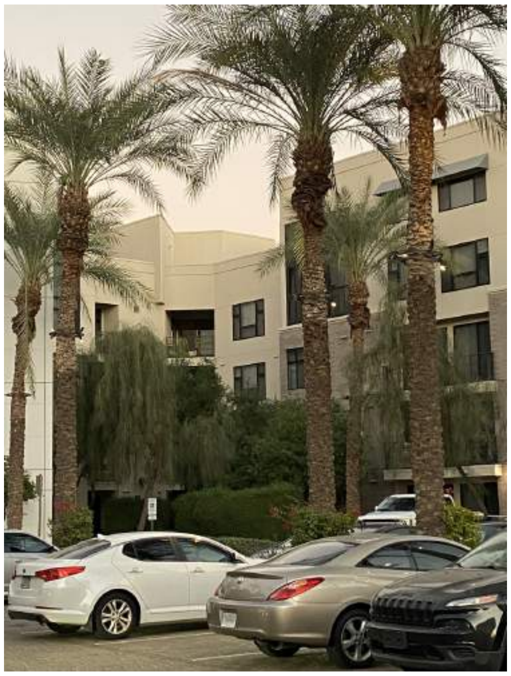

Figure 8: Third image

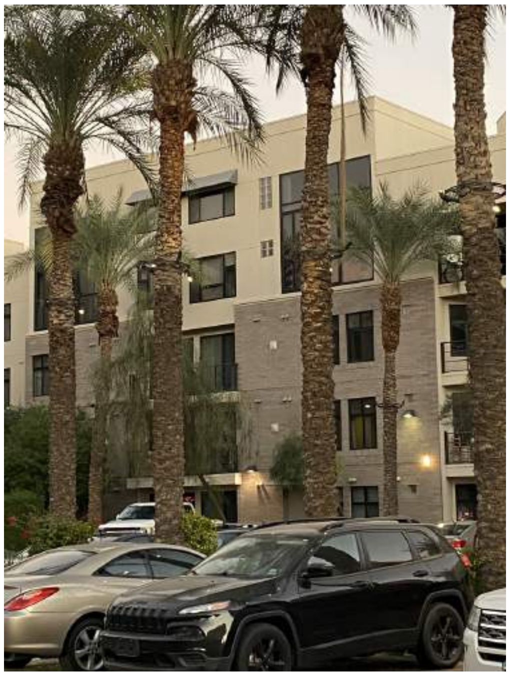

Figure 9: Fourth image

## 5.2 Part 5.2

I chose this image as the reference image.

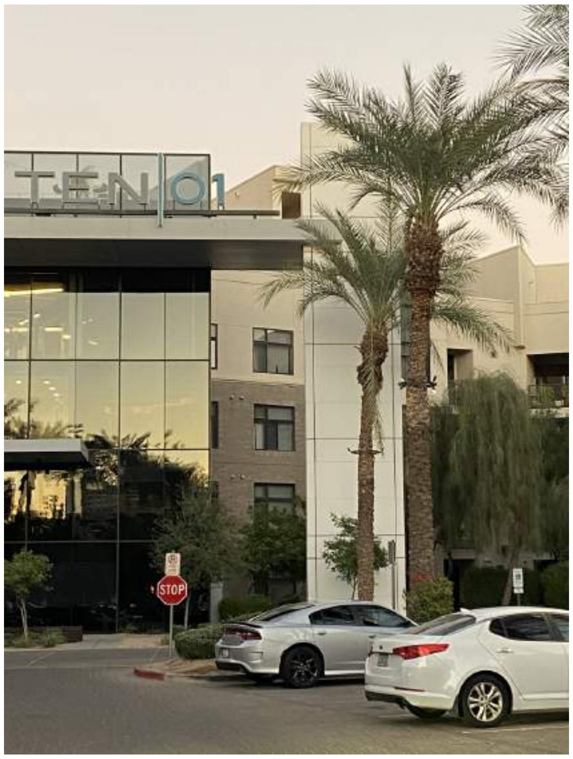

Figure 10: Refernece middle image

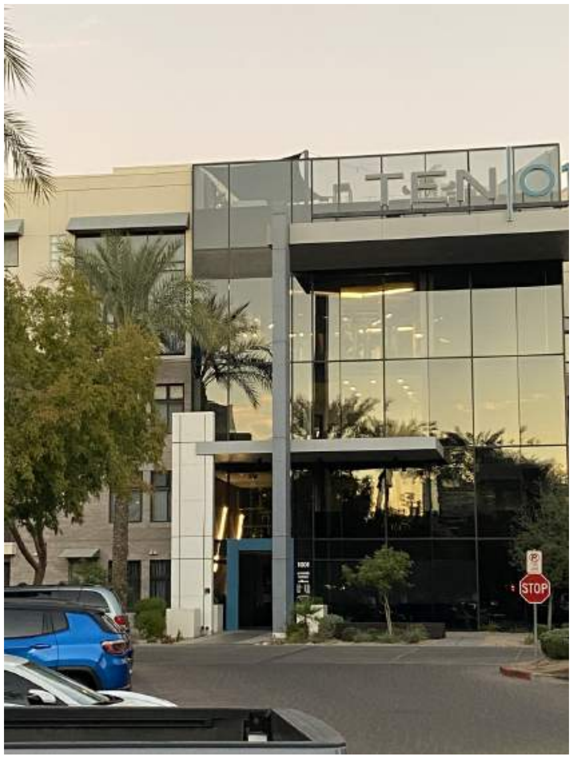

Figure 11: Trail reference image

The 2nd image in sequence is taken as reference point for the better homographic mosiacs shown in figure 10. For the trail purpose I took fig 11 which is a corner image as well to check.

The respective outputs are shown in part 5.5 at the end.

The minimum points required for the calculation of the homography is 4.

## 5.3 Part 5.3

Feature matching

I created the feature descriptors using SIFT around the corners. But before applying SIFT to the corners obtained from harris,It is important that interest points are spatially well distributed over the image. Adaptive Non maximal Suppression applied as mentioned in paper.Reference paper link
I applied Adaptive Non maximal Suppression on the output of harris corners.

It considers all the corners from harris corners output and runs to find the corners with highest strength and only those that are a maximum in a neighbourhood of radius r pixels are retained.T

The code for harris corners is same as mentioned in part 4.2 above in listing 3.

The Adaptive Non maximal Suppression is as follows in listing 8.

Listing 8: Adaptive Non maximal Suppression

```
num = len(coords); inf = sys.maxsize; r = inf * np.ones((num,3)); ED = 0
for i in range(num):
    for j in range(num):
        x_i = coords[i][0]
        y_i = coords[i][1]
        neighbours_x = coords[j][0]
        neighbours_y = coords[j][1]
        if img_h[y_i,x_i] > img_h[neighbours_y,neighbours_x]:
            ED = (neighbours_x - x_i)**2 + (neighbours_y - y_i)**2
        if ED < r[i,0]:
            r[i,0] = ED
            r[i,1] = x_i
            r[i,2] = y_i
arr = r[:,0]
feature_sorting = np.argsort(-arr)
feature_cord = r[feature_sorting]
Nbest_corners = feature_cord[:n_best,:]
```

I used SIFT descriptors used with following hyper parameters, The full code is same as shown in part 4.2, but the crux changes in code are in listing 9 below.

Listing 9: SIFT for feature descripors

```
descriptor = gray[x_start:x_start+patch_size, y_start:y_start+patch_size]
descriptor = cv2.GaussianBlur(descriptor, (7,7), cv2.BORDER_DEFAULT)
sub =5
descriptor = descriptor[::sub,::sub]
descriptor1 = descriptor.reshape((64,1))
descriptor_standard = (descriptor1 - descriptor1.mean())/std
n_descriptors.append(descriptor_standard)
```

The following is the comparison of output of harris corner and the ANMS output of same image for each of the selected images showing corners.

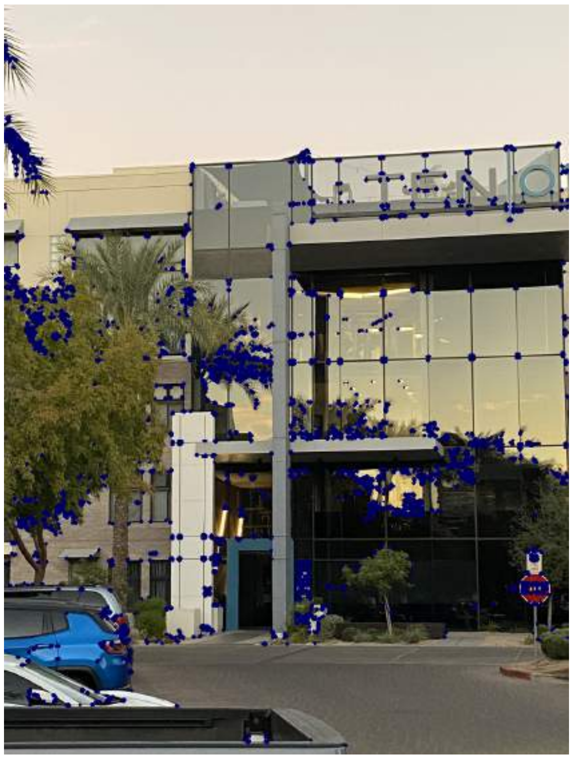
Figure 12: Harris corners image 1

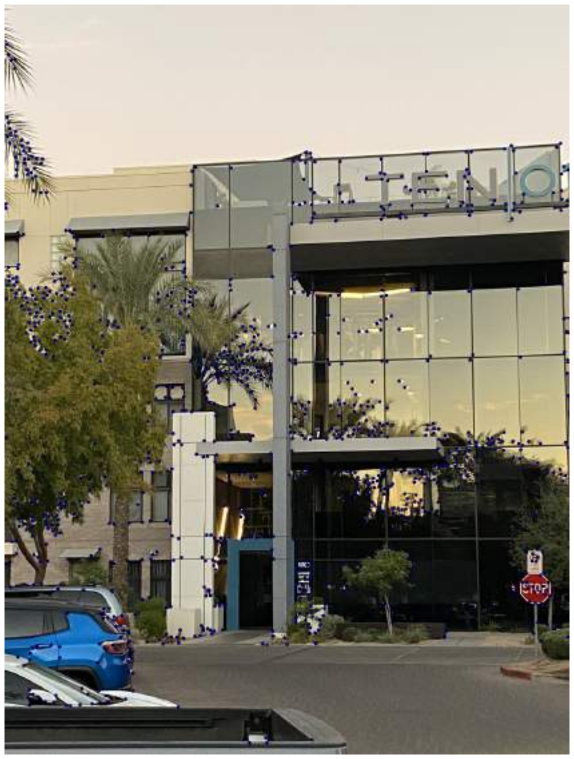
Figure 13: ANMS corners image1

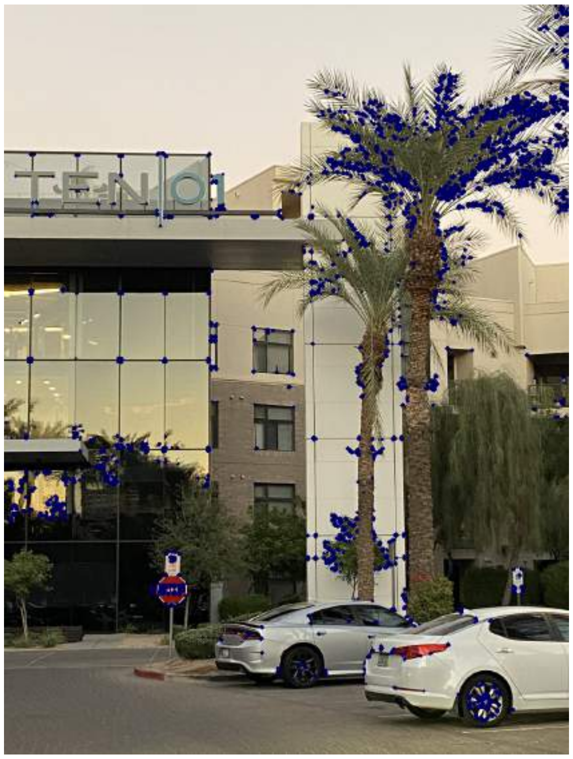

Figure 14: Harris corners image 2

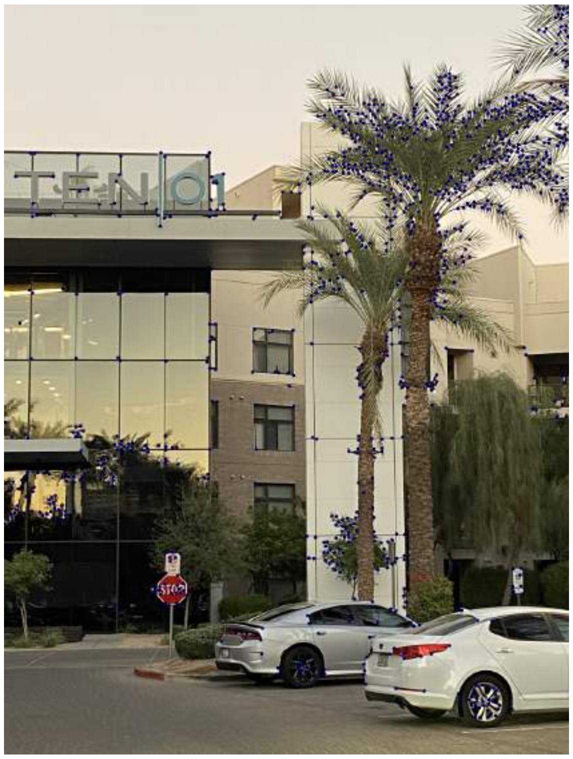

Figure 15: ANMS corners image 2

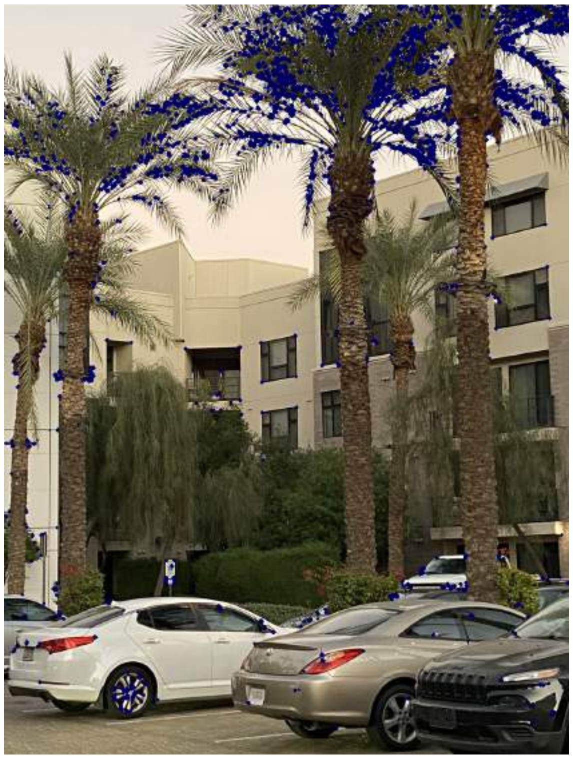

Figure 16: Harris corners image 3

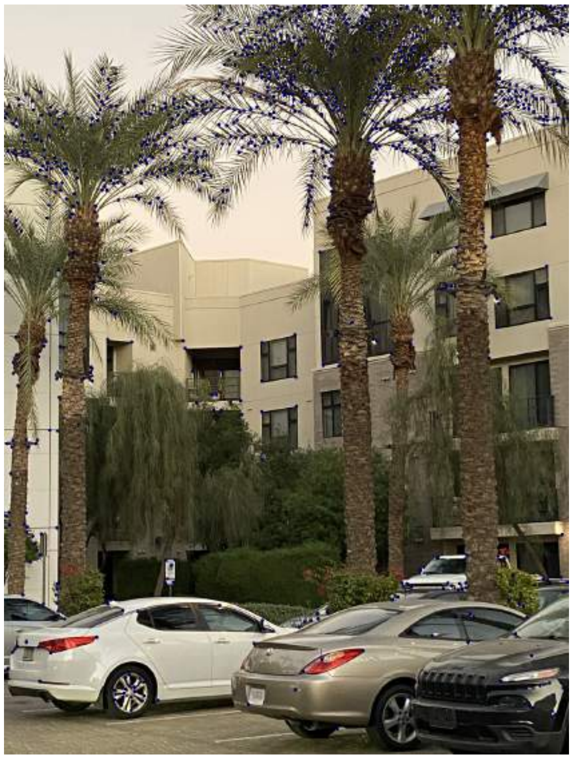

Figure 17: ANMS corners image 3

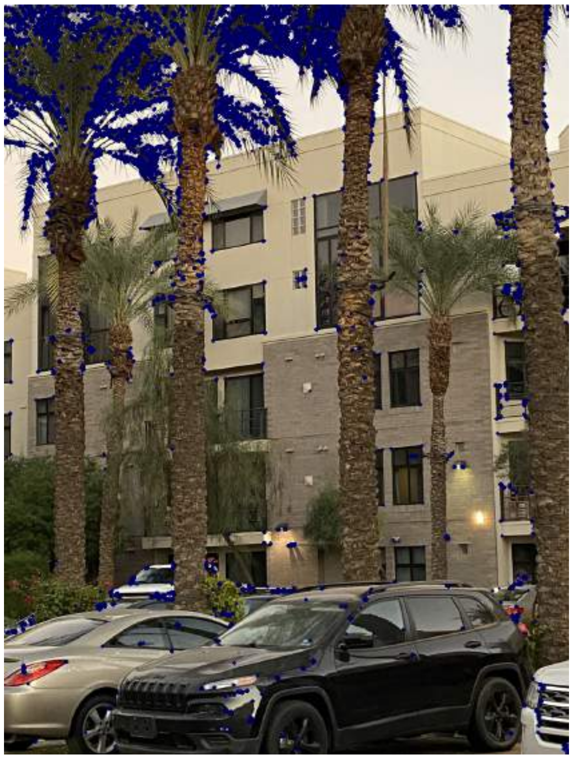

Figure 18: Harris corners image 4

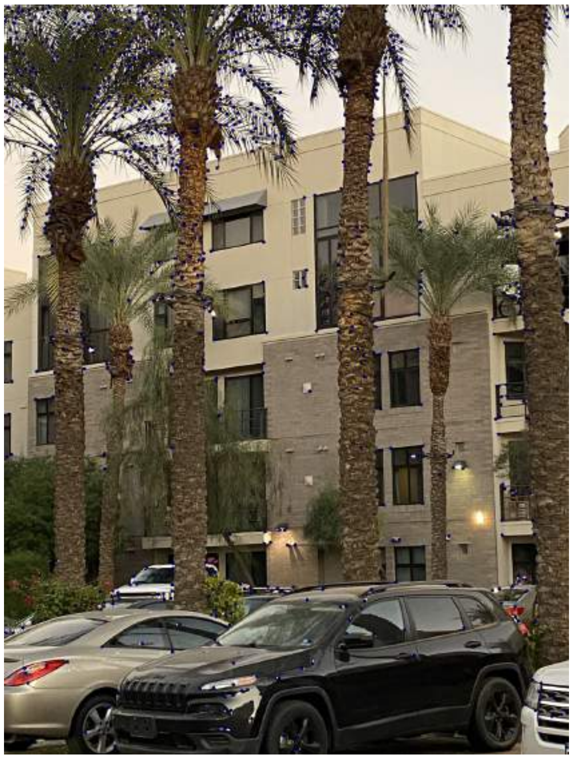

Figure 19: ANMS corners image 4

Feature matching :

I established feature matches using same code mentioned in part 4.2 by the Lowe's algorithm in listing 5 and cv2.draw in listing 6 function to make the tentative matches drawings as well.

The image in figure beside is the feature tentative matching of the reference image shown in figure 7 and the 3rd image in shown in figure 8 with lines being drawn using listing 6.Same pipeline is applied to get the tentative matches.

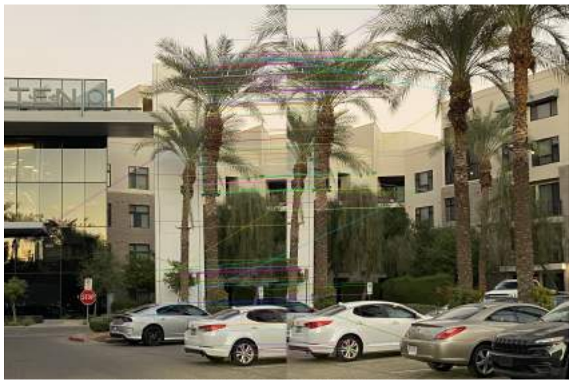

Figure 20: Matches of the fig 7(REF), 8

The image in figure beside is the feature tentative matching of the warped image ( reference image shown in figure 7 and the 3rd image in shown in figure 8) to 4th image figure 9.

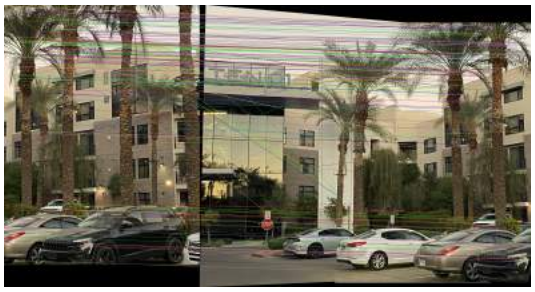

Figure 21: Matches of the fig 7(REF)+8, 9

The image in figure beside is the feature tentative matching of the (reference image shown in figure 7 and the 3rd image in shown in figure 8 and 4th shown in fig 9 ) to 1st image in fig 6.

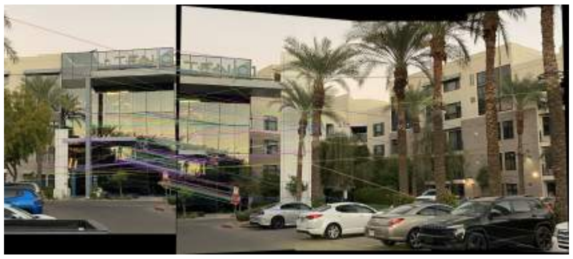
Figure 22: Matches of the fig 7(REF)+8 +9 to 6

## 5.4 Part 5.4

Although the feature matches are good, if we apply warping by finding the homography between the images, it would pose artifacts and some errors.

To remove the outliers and keep the strong matches only ransac is proposed.

Homography estimation with RANSAC

RANSAC algorithm ;

The re projection error is calculated after picking 4 random points from the best matches corners achieved from feature matching.

If the reprojection error is below the threshold reprojection error, we are keeping it as an inlier otherwise considering it as outlier.

The re-projectionerror for each pair of images is varied to get the best possible matches and remove the outliers.

The below is the code for RANSAC in listing 11, I ran it for 1000 iterations.

For the homography calculation, the internal function of cv2 is being used mentioned in listing 10.

Listing 10: Homography

```
h_matrix  = cv2.getPerspectiveTransform(np.float32(point1), np.float32(point2))
h_final_matrix, status =cv2.findHomography(np.float32(pts_1),np.float32(pts_2))
```

Listing 11: RANSAC Algorithm

```
for i in range(1000):    #Nmax iterations
    keypoints_1 = [x[0] for x in matched_pairs]
    keypoints_2 = [x[1] for x in matched_pairs]
    length = len(keypoints_1)
    randomlist = random.sample(range(0, length), 4)
    points_1 = [keypoints_1[idx] for idx in randomlist]
    points_2 = [keypoints_2[idx] for idx in randomlist]

    h_matrix = homography(points_1, points_2)
    points = []
    count_inliers = 0
    for i in range(length):
        a = (np.array(keypoints_2[i]))
        ssd = np.linalg.norm(np.expand_dims(np.array(keypoints_2[i]), 1) -
            dot_product(h_matrix, keypoints_1[i]))
        if ssd < threshold: # REPROJECTION ERROR
            count_inliers += 1
            points.append((keypoints_1[i], keypoints_2[i]))
    COUNT.append(-count_inliers)
    inliers.append((h_matrix, points))
max_count_idx = np.argsort(COUNT)
max_count_idx = max_count_idx[0]
final_matched_pairs = inliers[max_count_idx][1]
pts_1 = [x[0] for x in final_matched_pairs]
pts_2 = [x[1] for x in final_matched_pairs]
h_final_matrix, status = cv2.findHomography(np.float32(pts_1),np.float32(pts_2))
```

Final matches after filtering the outliers are as follows :

The final homography matrix between reference and 3rd image is

$$\begin{bmatrix} 1.1045 & -0.023 & -668.275 \\ 0.09 & 1.08 & -92.5076 \\ 0.00016 & 0.000000049 & 1 \end{bmatrix}$$

and the reprojection error chosen is 10, It can be kept a bit lesser as well. But to keep more inliers, I chose this. With the above homography matrix my average reprojection error is 5.6.

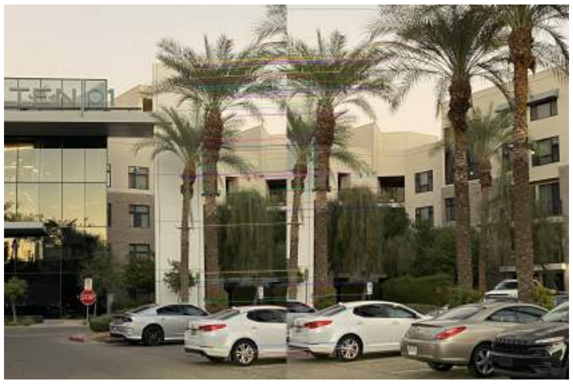

Figure 23: Fourth image

The final homography matrix between (reference+3rd) and 4th image is

$$\begin{bmatrix} 0.843 & 0.031 & 1194.334 \\ -0.101 & 0.979 & 95.98 \\ -0.000009 & 0.000000084 & 1 \end{bmatrix}$$

and the reprojection error chosen is 15, It can be kept a bit lesser as well. But to keep more inliers, I chose this. My average reprojection error is 7.2 for above parameters.

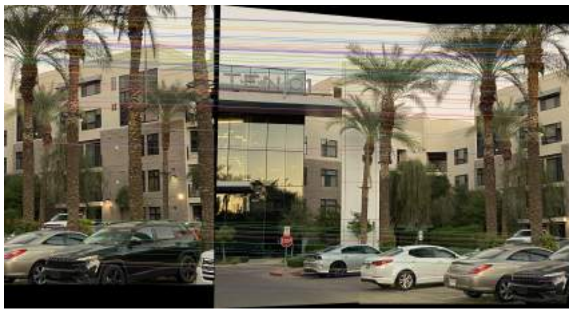

Figure 24: Fourth image

The final homography matrix between (reference+3rd+4th) and 1st image is

$$\begin{bmatrix} 1.240 & -0.05 & -901.78 \\ 0.246 & 1.13 & -171.79 \\ 0.000002 & 0.0000002 & 1 \end{bmatrix}$$

and the reprojection error chosen is 10, It can be kept a bit lesser as well. But to keep more inliers, I chose this. The average reprojection error for this is 2.6.

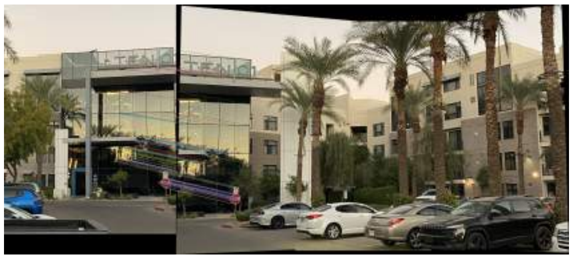

Figure 25: Fourth image

## 5.5 Part 5.5

Warping and compositing ;

The below is the crux code implemented for the warping of the images after performing the RANSAC. cv2.warpPerspective is been used in code below and setup is shown in listing 12.

Listing 12: Warping images

```
img1 = img1
img2 = img2
h1,w1 = img1.shape[:2]
h2,w2 = img2.shape[:2]
pts1 = np.float32([[0,0],[0,h1],[w1,h1],[w1,0]]).reshape(-1,1,2)
pts2 = np.float32([[0,0],[0,h2],[w2,h2],[w2,0]]).reshape(-1,1,2)
pts2_ = cv2.perspectiveTransform(pts2, H)
pts = np.concatenate((pts1, pts2_), axis=0)
[xmin, ymin] = np.int32(pts.min(axis=0).ravel())
[xmax, ymax] = np.int32(pts.max(axis=0).ravel())
t = [-xmin,-ymin]
Ht = np.array([[1,0,t[0]],[0,1,t[1]],[0,0,1]]) # translate
result = cv2.warpPerspective(img1, Ht.dot(H), (xmax-xmin, ymax-ymin), flags = cv2.
    INTER_LINEAR)
result[t[1]:h1+t[1],t[0]:w1+t[0]] = img2
```

The final image after warping and stitching the image is as follows:

The following sticthed image is when i take the reference image as figure 6 explained in part 5.2.

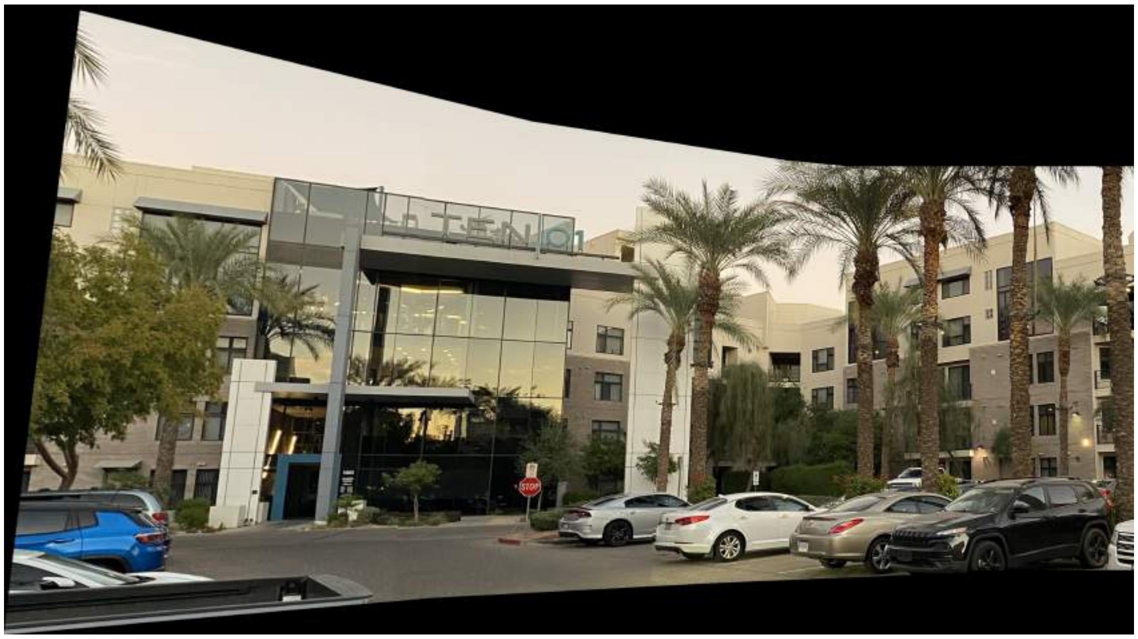

Figure 26: FINAL HOMOGRAPHY when corner image is taken as reference

The following sticthed image is when i take the reference image as figure 7 explained in part 5.2. It is the when reference image in the middle of the sequence. The mosiacs are much better. The results in the above parts are with respect to the middle image reference.

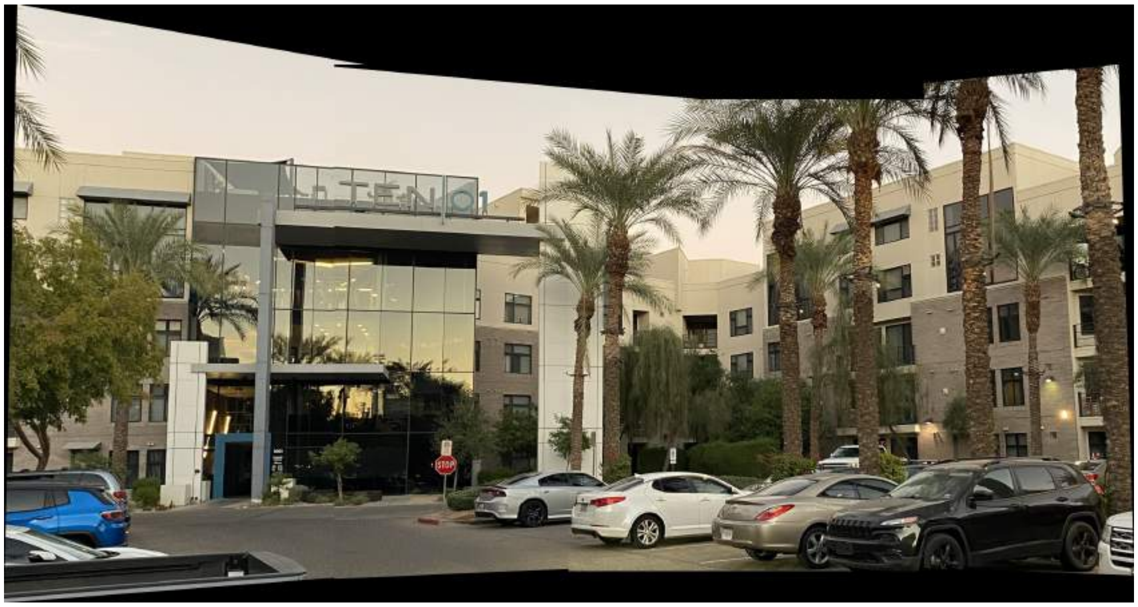

Figure 27: FINAL HOMOGRAPHY when middle image is taken as reference

OTHER ASPECTS:

Homography can be done better by checking the thresholding better in harris corners , match ratio change in feature matching as well. Taking better number of points from ANMS also helps. Thresholding the reprojection error through ransac is the crux which makes wrap better by getting the best y matrix.

Making reprojection better by changing the threshold may give better final homography matrix as well.