Homography - PART A

Name: Nikhil Mukund Karve

$$ \\ $$
Here we are computing homography matrix from one image to another by using the pixel coordinates of the corresponding points in the image. We are creating a homography matrix 'h' that will convert the image pixel coordinates from the source image to the destination image by projecting the points in the source image plane to the destination image plane

In [32]:

```
import numpy as np
```

Creating image points list

Here we create 4 lists each consisting of the corresponding points of the same features from image1 to image2

- xs - X coordinates of the image pixel position from source image
- ys - Y coordinates of the image pixel position from source image
- xd - X coordinates of the image pixel position from the destination image
- yd - Y coordinates of the image pixel position from the destination image

In [33]:

```
xs = [368.1937,366.9331,365.6942,364.2767,362.9518,361.6201,360.1071,419.5753,418.3222,416.9176,415.6446,
ys = [188.3461,240.6366,292.5464,344.1300,395.6568,447.0891,498.7948,189.8007,242.0051,293.9554,345.5727,
xd = [319.0576, 348.5998, 377.6357, 406.4299, 435.1823, 463.8352, 492.4751, 361.0885, 390.4941, 419.5286,
yd = [370.0124, 412.3830, 454.3980, 496.2708, 537.9025, 579.5414, 621.3035, 341.5682, 383.8370, 426.0496,
```

Sanity check so that we make sure we have 25 points from each image as we have taken 25 features

Even though there are only 9 unknowns in the equation the more points we have the more accurate we wil have our homography matrix as we are setting up the problem as an overdetermined problem.

In [34]:

```
len(xs)
```

Out[34]:

```
25
```

In [35]:

```
len(ys)
```

Out[35]:

```
25
```

In [36]:

```
len(xd)
```

Out[36]:

```
25
```

In [37]:

```
len(yd)
```

Out[37]:

```
25
```

Once we have the image points now we will solve for homography

$$\begin{bmatrix} xd \\ yd \\ 1 \end{bmatrix} \equiv \begin{bmatrix} xd \\ yd \\ zd \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} xs \\ ys \\ 1 \end{bmatrix}$$

To get actual points from homogenous representation you need to do $$ \frac{xd^{(i)}}{zd^{(i)}} and \frac{yd^{(i)}}{zd^{(i)}}$$

So to solve for this we will have A.h=0

$$\begin{bmatrix} xs^{(i)} & ys^{(i)} & 1 & 0 & 0 & 0 & -xd^{(i)}xs^{(i)} & -xd^{(i)}ys^{(i)} & -xd^{(i)} \\ 0 & 0 & 0 & xs^{(i)} & ys^{(i)} & 1 & -yd^{(i)}xs^{(i)} & -yd^{(i)}ys^{(i)} & -yd^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} =\begin{bmatrix} 0 \\ 0 \end{bmatrix} $$

This equation is for one point like this we need to repeat for all sets of points

$$\begin{bmatrix} xs^{(1)} & ys^{(1)} & 1 & 0 & 0 & 0 & -xd^{(1)}xs^{(1)} & -xd^{(1)}ys^{(1)} & -xd^{(1)} \\ 0 & 0 & 0 & xs^{(1)} & ys^{(1)} & 1 & -yd^{(1)}xs^{(1)} & -yd^{(1)}ys^{(1)} & -yd^{(1)} \\ xs^{(2)} & ys^{(2)} & 1 & 0 & 0 & 0 & -xd^{(2)}xs^{(2)} & -xd^{(2)}ys^{(2)} & -xd^{(2)} \\ 0 & 0 & 0 & xs^{(2)} & ys^{(2)} & 1 & -yd^{(2)}xs^{(2)} & -yd^{(2)}ys^{(2)} & -yd^{(2)} \\ xs^{(3)} & ys^{(3)} & 1 & 0 & 0 & 0 & -xd^{(3)}xs^{(3)} & -xd^{(3)}ys^{(3)} & -xd^{(3)} \\ 0 & 0 & 0 & xs^{(3)} & ys^{(3)} & 1 & -yd^{(3)}xs^{(3)} & -yd^{(3)}ys^{(3)} & -yd^{(3)} \\ xs^{(4)} & ys^{(4)} & 1 & 0 & 0 & 0 & -xd^{(4)}xs^{(4)} & -xd^{(4)}ys^{(4)} & -xd^{(4)} \\ 0 & 0 & 0 & xs^{(4)} & ys^{(4)} & 1 & -yd^{(4)}xs^{(4)} & -yd^{(4)}ys^{(4)} & -yd^{(4)} \\ xs^{(5)} & ys^{(5)} & 1 & 0 & 0 & 0 & -xd^{(5)}xs^{(5)} & -xd^{(5)}ys^{(5)} & -xd^{(5)} \\ 0 & 0 & 0 & xs^{(5)} & ys^{(5)} & 1 & -yd^{(5)}xs^{(5)} & -yd^{(5)}ys^{(5)} & -yd^{(5)} \\ xs^{(6)} & ys^{(6)} & 1 & 0 & 0 & 0 & -xd^{(6)}xs^{(6)} & -xd^{(6)}ys^{(6)} & -xd^{(6)} \\ 0 & 0 & 0 & xs^{(6)} & ys^{(6)} & 1 & -yd^{(6)}xs^{(6)} & -yd^{(6)}ys^{(6)} & -yd^{(6)} \\ xs^{(7)} & ys^{(7)} & 1 & 0 & 0 & 0 & -xd^{(7)}xs^{(7)} & -xd^{(7)}ys^{(7)} & -xd^{(7)} \\ 0 & 0 & 0 & xs^{(7)} & ys^{(7)} & 1 & -yd^{(7)}xs^{(7)} & -yd^{(7)}ys^{(7)} & -yd^{(7)} \\ xs^{(8)} & ys^{(8)} & 1 & 0 & 0 & 0 & -xd^{(8)}xs^{(8)} & -xd^{(8)}ys^{(8)} & -xd^{(8)} \\ 0 & 0 & 0 & xs^{(8)} & ys^{(8)} & 1 & -yd^{(8)}xs^{(8)} & -yd^{(8)}ys^{(8)} & -yd^{(8)} \\ xs^{(9)} & ys^{(9)} & 1 & 0 & 0 & 0 & -xd^{(9)}xs^{(9)} & -xd^{(9)}ys^{(9)} & -xd^{(9)} \\ 0 & 0 & 0 & xs^{(9)} & ys^{(9)} & 1 & -yd^{(9)}xs^{(9)} & -yd^{(9)}ys^{(9)} & -yd^{(9)} \\ xs^{(10)} & ys^{(10)} & 1 & 0 & 0 & 0 & -xd^{(10)}xs^{(10)} & -xd^{(10)}ys^{(10)} & -xd^{(10)} \\ 0 & 0 & 0 & xs^{(10)} & ys^{(10)} & 1 & -yd^{(10)}xs^{(10)} & -yd^{(10)}ys^{(10)} & -yd^{(10)} \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ xs^{(25)} & ys^{(25)} & 1 & 0 & 0 & 0 & -xd^{(25)}xs^{(25)} & -xd^{(25)}ys^{(25)} & -xd^{(25)} \\ 0 & 0 & 0 & xs^{(25)} & ys^{(25)} & 1 & -yd^{(25)}xs^{(25)} & -yd^{(25)}ys^{(25)} & -yd^{(25)} \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 \end{bmatrix} $$

Now we need to solve for h.

In [38]:

```
#xs, ys, 1, 0, 0, 0, -xdxs, -xdys, -xd
#0, 0, 0, xs, ys, 1, -ydxs, -ydys, -yd
xdxs=-np.multiply(np.array(xd),np.array(xs))
xdys =-np.multiply(np.array(xd),np.array(ys))
ydxs=-np.multiply(np.array(yd),np.array(xs))
ydys=-np.multiply(np.array(yd),np.array(ys))
```

In [39]:

```
A=[]
for i in range(0,25):
    mat1 = [xs[i],ys[i],1,0,0,0,xdxs[i],xdys[i],-xd[i]]
    mat2 = [0,0,0,xs[i],ys[i],1,ydxs[i],ydys[i],-yd[i]]
    A.append(mat1)
    A.append(mat2)
```

In [40]:

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

In [41]:

```
A.shape
```

Out[41]:

```
(50, 9)
```

To solve for H

Solve for **h** such that:
$$A.h = 0 \space and \space \lvert\lvert h \rvert\rvert = 0$$

Loss function L(h):

$$L(h,\lambda) = h^TA^TAh - \lambda(h^Th-1)$$
On taking first derivative:

$$ 2A^TAh - 2\lambda h = 0$$
We can set it up as a Eigen Value Decomposition of the inner product of matrix A:

$$A^TAh = \lambda h$$
The Eigenvector *h* belongs to the smallest eigenvalue of λ of matrix A^TA minimizes the loss function $L(h)$.

In [42]:

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

In [43]:

```
inner_product.shape
```

Out[43]:

```
(9, 9)
```

In [44]:

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

In [45]:

```
w
```

Out[45]:

```
array([3.37836847e+12, 8.85307300e+10, 7.52578625e+06, 2.70784459e+05,
       1.25346927e+05, 1.96727109e+04, 4.78534174e+03, 1.05764484e-02,
       5.47935846e-07])
```

In [46]:

```
v = v.T # take transpose as the solution exists in the null space
```

In [47]:

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

Out[47]:

```
array([1.00000000e+00, 2.62051729e-02, 2.22763926e-06, 8.01524348e-08,
       3.71027993e-08, 5.82313951e-09, 1.41646531e-09, 3.13063791e-15,
       1.62189486e-19])
```

In [48]:

```
v_prime = v[-1]
```

In [49]:

```
h = v_prime.reshape((3,3))
```

We have now taken the last column and reshaped it as a 3X4 matrix which is our homography matrix *h*

In [50]:

```
h
```

Out[50]:

```
array([[-1.83609191e-03, -1.33610055e-03,  2.00183150e-01],
       [ 1.30622399e-03, -1.83013316e-03, -9.79750660e-01],
       [-1.95411188e-08, -3.02895676e-08, -2.26695356e-03]])
```

Now let's check if our homography matrix obtained is right by using it to project a point from source image plane to destination image plane

In [51]:

```
product = np.dot(h, np.array([368.1937, 188.3461, 1]).T)
```

In [52]:

```
product
```

Out[52]:

```
array([-0.72750365, -0.84350566, -0.00227985])
```

In [53]:

```
product = product / product[2]
```

In [54]:

```
product
```

Out[54]:

```
array([319.10106592, 369.98241231,   1.        ])
```

We can see that the points obtained are close to the actual points in the destination image plane which confirms that the obtained homography matrix *h* is correct

Part B:

$$ \\ $$
Output Video & Code :- Github Link: Click Here

Part C:

$$ \\ $$
Output Video & Code :- Github Link: Click Here