Q1-Final

March 1, 2020

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import random
from glob import glob
import time

# get_ipython().magic('matplotlib inline')
```

In [2]:
```
def construct_H_matrix(x, xs):
    '''
        Construct the correspondance matrix for estimating Homography

        Keyword Arguments:
            x  -- Image points from the First image
            xs -- Image points from the Second Image

        Return Values:
            A  -- Constructed correspondance matrix
    '''
    A = np.zeros((0, 9))
    for i in range(len(x)):
        a = np.array([
            [x[i][0], x[i][1], 1, 0, 0, 0, -xs[i][0]*x[i][0], -xs[i][0]*x[i][1], -xs[i][
            [0, 0, 0, x[i][0], x[i][1], 1, -xs[i][1]*x[i][0], -xs[i][1]*x[i][1], -xs[i][
            ])
        A = np.concatenate((A, a))
    return A
```

In [3]:
```
def transform_points(H, x):
    '''
        Given homography transforms points into the homography frame of reference

        Keyword Arguments:
            H  -- 3*3 homography matrix
            x  -- N*2 Points to be transformed

        Return Values:
```

```
        xs  -- N*3 Transformed Points
    '''
    # Convert points into homogeneous coordinates
    x_homogeneous = np.concatenate((x, np.ones((len(x), 1))), axis=1)

    # Calculate the transformed points and normalize them
    xs = H @ x_homogeneous.T
    xs[0, :] = xs[0, :] / xs[2, :]
    xs[1, :] = xs[1, :] / xs[2, :]
    xs[2, :] = xs[2, :] / xs[2, :]
    xs = xs.T

    return xs
```

```
In [4]: def calculate_transformation_error(H, x, xs):
    '''
        Calculate the transformation error between the transformed points and the ground

        Keyword Arguments:
            H                       -- 3*3 homography matrix
            x                       -- N*2 points to be transformed
            xs                      -- N*2 ground truth points

        Return Values:
            transformation_error  -- Return the L2 norm of the error
    '''
    # Convert points into homogeneous coordinates
    xs_homogeneous = np.concatenate((xs, np.ones((len(xs), 1))), axis=1)

    # Calculate the projected points
    transformed_coords = transform_points(H, x)

    # Calculate the projection error
    transformation_error =  np.linalg.norm(xs_homogeneous - transformed_coords, axis=1)

    return transformation_error
```

0.1 RANSAC

- For better estimation of the homography matrix, we perform RANSAC and try to get the homography matrix with maximum inliers.
- We calculate the inliers based on the transformation error, which is required to be below a certain preset threshold that we choose.

```
In [5]: def RANSAC(img_coords_1, img_coords_2, num_points, max_iterations=30000, thresh=0.1):
    '''
        Get the best estimate of the Homography Matrix using RANSAC
```

```
    Keyword Arguments:
        img_coords_1   -- Coordinates of the first image after matching
        img_coords_2   -- Coordinates of the first image after matching
        num_points     -- Number of matched points
        max_iterations -- Maximum Number of iterations to run RANSAC for (default=20
        thresh         -- Threshold to compute inliers (default=0.1)

    Return Values:
        H              -- Estimated Homography matrix using RANSAC
'''
min_transform_error = 999999999
max_inliers = -999999999

# Best estimate of Projection matrix by far
_H = np.zeros((3, 3))

inliers = []

for i in range(max_iterations):

    # Randomly select 4 world points and the corresponding image points
    idx = random.sample(range(0, num_points), 4)
    x  = img_coords_1[idx]
    xs = img_coords_2[idx]

    # Perform DLT and get the Transformation Matrix
    H = DLT(x, xs)

    # Calculate projection error
    transformation_error = calculate_transformation_error(H, img_coords_1, img_coord


    inliers = np.sum(transformation_error<thresh)

    if inliers > max_inliers:
        max_inliers = inliers
        _H = H

    # Repeat for a maximum number of iterations

return _H
```

0.2 Estimating the Homography Matrix

- To estimate the homography matrix we perform a DLT like estimation
- $x_2 = H_{21} \cdot x_1$