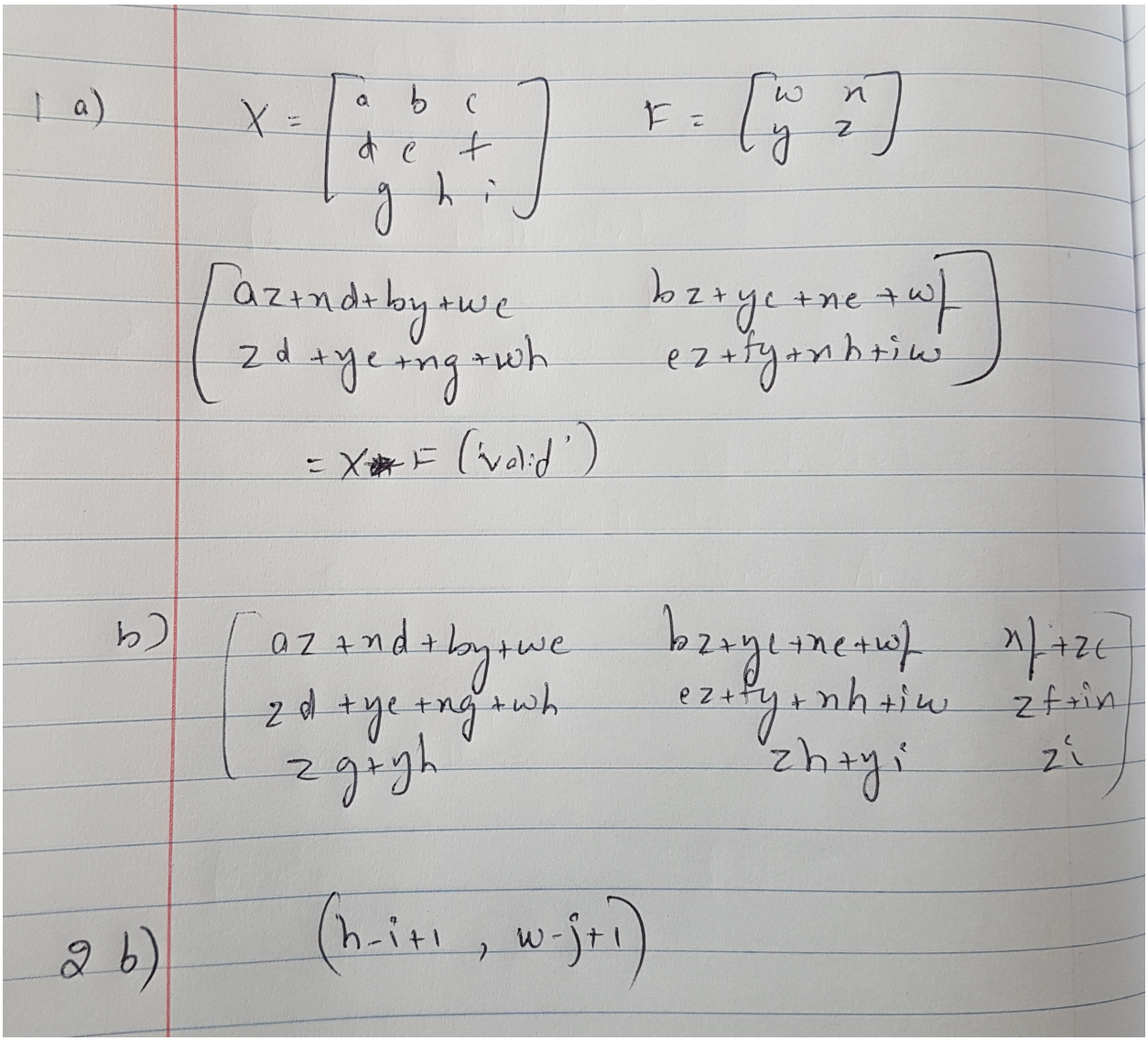
Report of Assignment 1

Srinidhi Goud Myadaboyina
N12449860

NOTE: The entire code could not be run on a common script but only part 1,3 & 4 are. Part 2 is written in python as with my version of Matlab and OS, I could not find a compatible Matlab code for finding SIFT feature extraction. So please excuse this and run the python code separately.



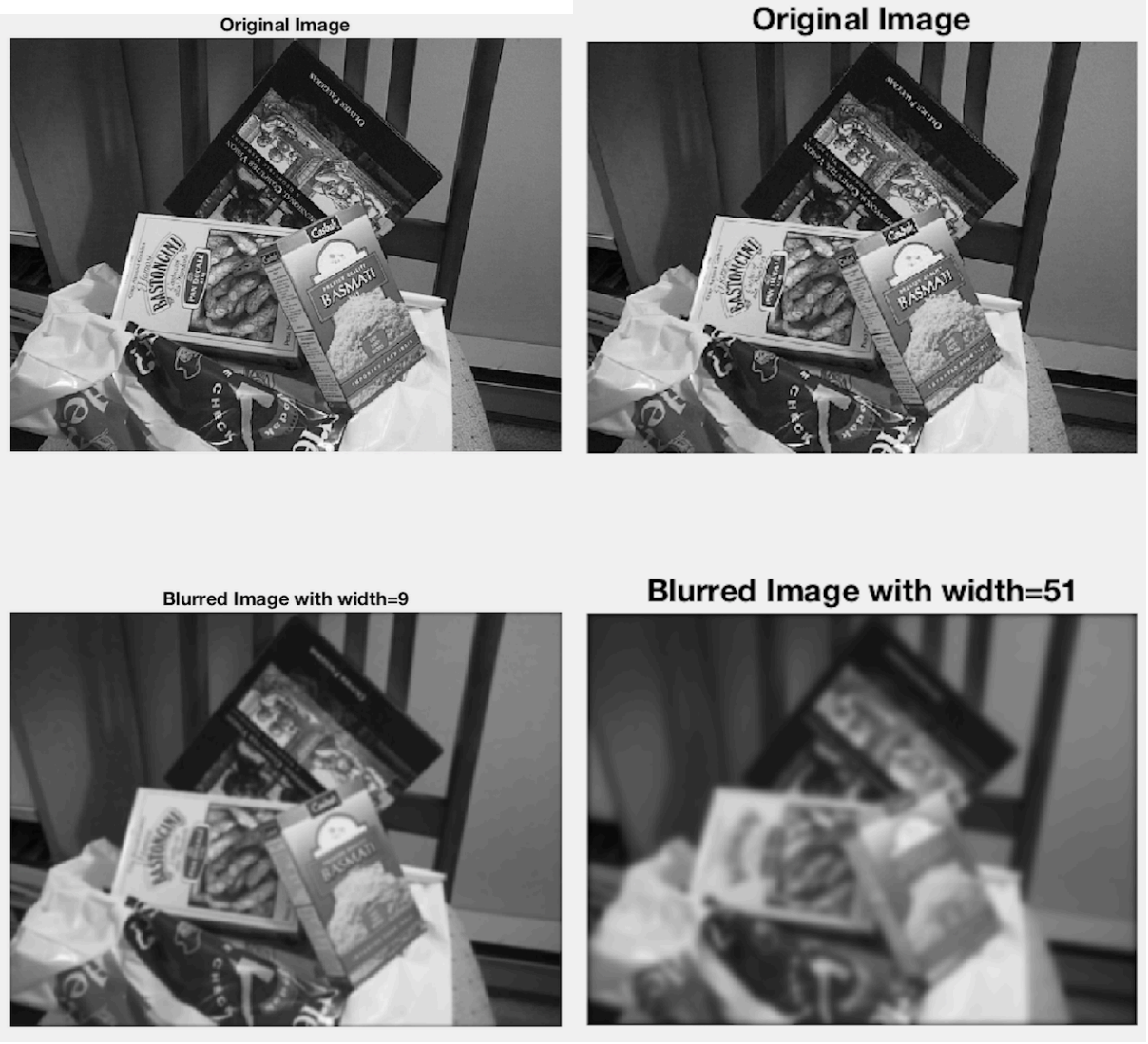
2c) CODE:

```
close all;
clear all;
clc;

%1)2 part c

filtk=[1/4 2/4 1/4];
filt2k=filtk;
prompt = 'What is the width of the kernel? ';
x=input(prompt);
prompt='Now select the image, after navigating to its folder';

while(mod(x,2)==0)%Enter only odd kernel width
    prompt = 'Enter odd value? ';
    x=input(prompt);
end
prompt='Now select the image, after navigating to its folder. Type
any integer and press return/enter to continue';
y=input(prompt);
%You need to enter any integer and press enter and then select any
image, that will be feeded into the system
sz=fix(x/2)-1;
for i=1:sz
    filt2k=conv2(filtk,filt2k);%Convolve for the given width
end
image=uigetfile(['*.*']);
RGB = (imread(image));
RGB2=imfilter(RGB,filt2k);%Convolving in x-axis
blurred=imfilter(RGB2.',filt2k);%Convolving in y-axis
blurred=blurred.';
figure
subplot(2,1,1)
imshow(RGB);
title('Original Image');

subplot(2,1,2);
imshow(blurred);
title(['Blurred Image with width=',num2str(x)]);
```

END CODE:

To run this code, just enter any integer but it only runs after you enter odd integers. The code reads in Image as input interactively. So, when prompted, press any integer and press enter, browse to the folder containing image and select it. The code blurs it.

RESULTS:


Original Image
Original Image
Blurred Image with width=9
Blurred Image with width=51

2)

This code is in python file CV_@.py

To run the part 2 code, give the inputs in command line itself, like this:

```
python ./CV_2.py book.pgm scene.pgm
```

CODE:

```
import cv2
import sys
import os.path
import numpy as np
import random

def is_invertible(a):
        return a.shape[0] == a.shape[1] and np.linalg.matrix_rank(a) == a.shape[0]
def compare(filename1, filename2):
   img1 = cv2.imread(filename1)        # queryImage
   img2 = cv2.imread(filename2)        # trainImage
   rows,cols,ch = img1.shape
   # pts1 = np.float32([[50,50],[200,50],[50,200]])
   # pts2 = np.float32([[10,100],[200,50],[100,250]])

   # M = cv2.getAffineTransform(pts1,pts2)
   # print(M)
   # Initiate SIFT detector
   sift = cv2.xfeatures2d.SIFT_create()

   # find the keypoints and descriptors with SIFT
   kp1, des1 = sift.detectAndCompute(img1,None)
   kp2, des2 = sift.detectAndCompute(img2,None)
   img3 = cv2.drawKeypoints(img1,kp1,None,(255,255,0),4)
   img4 = cv2.drawKeypoints(img2,kp2,None,(255,255,0),4)
   cv2.imshow('with key points for image 1',img3)
   cv2.waitKey(0)
   cv2.destroyWindow('with key points for image 1')
   cv2.imshow('with key points for image 2',img4)
   cv2.waitKey(0)
   cv2.destroyWindow('with key points for image 2')
   # BFMatcher with default params
   bf = cv2.BFMatcher()
   matches = bf.knnMatch(des1,des2,k=2)
```

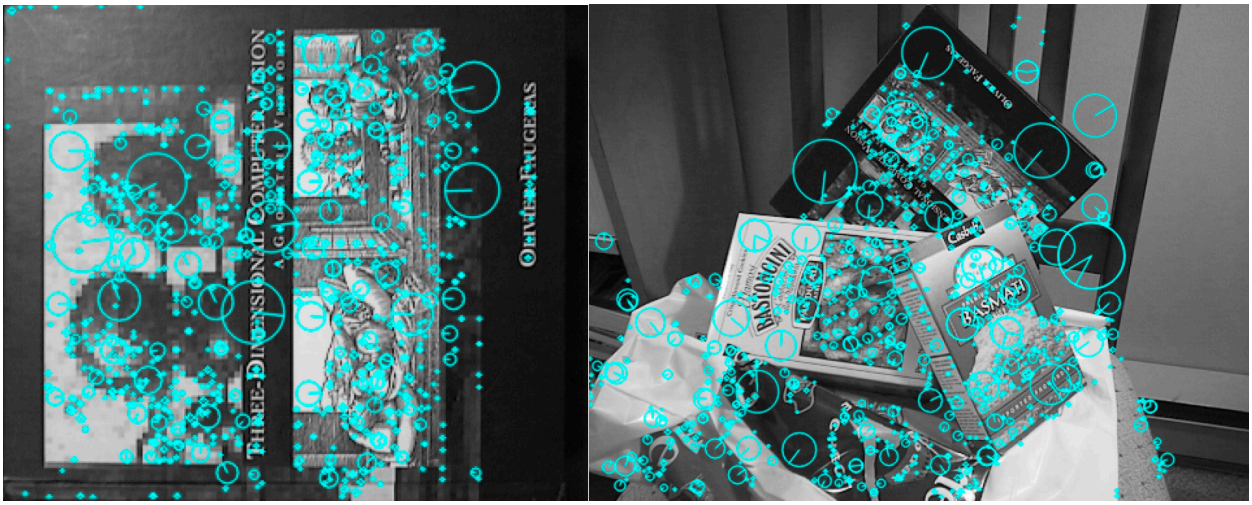
```
good = []
toogood=[]
for m,n in matches:
        if m.distance < 0.9*n.distance:
                good.append([m])
                toogood.append(m)
img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)
# print(kp1[1].pt)
# Show the image
cv2.imshow('Matched Features', img3)
cv2.waitKey(0)
cv2.destroyWindow('Matched Features')
src_pts = np.uint32([np.round((kp1[m.queryIdx]).pt) for m in toogood ]).reshape(-1,1,2)
dst_pts = np.uint32([ np.round(kp2[m.trainIdx].pt) for m in toogood ]).reshape(-1,1,2)
buf0,buf1,buf2=src_pts.shape
buf1=buf0-1
sum=0
Y=[]
Z=[]
for l in range(0,100):
        #
           (p1,p2,p3)=random.sample(range(0, buf0), 3)

A=np.array([src_pts[p1][0][0],src_pts[p1][0][1],0,0,1,0,0,0,src_pts[p1][0][0],src_pts[p1][0][1],0,
1,src_pts[p2][0][0],src_pts[p2][0][1],0,0,1,0,0,0,src_pts[p2][0][0],src_pts[p2][0][1],0,1,src_pts[p
3][0][0],src_pts[p3][0][1],0,0,1,0,0,0,src_pts[p3][0][0],src_pts[p3][0][1],0,1]).reshape(6,6)

B=np.array([dst_pts[p1][0][0],dst_pts[p1][0][1],dst_pts[p2][0][0],dst_pts[p2][0][1],dst_pts[p3][
0][0],dst_pts[p3][0][1]]).reshape(6,1)
            B=np.asarray(B)
            A=np.asarray(A)
            if(is_invertible(A)):
                p1=p1
            else:
                continue

            X=np.linalg.lstsq(A,B)[0]

            if(l==0):
                Y.append(X)
            M=np.array([X[0][0],X[1][0],X[2][0],X[3][0]]).reshape(2,2)
            T=np.array([X[4][0],X[5][0]]).reshape(2,1)
            z_local=[]
            trpts=[]
```

```
            sum_local=0
            for i in range(0,buf0):
                trpts.append(np.round(np.add(np.dot(M,(np.array(src_pts[i][:][0]))), T.T)))
                if((np.linalg.norm(trpts[i][:][0]-dst_pts[i][:][0]))<=10):
                    sum_local=sum_local+1
                    z_local.append(i)
            if(sum<sum_local):
                sum=sum_local
                Y[0]=X
                Z=np.array(z_local)

    A=[]
    B=[]
    alpha=0
    for it in Z:
        alpha=alpha+1
        A.append(np.array([src_pts[it][0][0],src_pts[it][0][1],0,0,1,0]))
        A.append(np.array([0,0,src_pts[it][0][0],src_pts[it][0][1],0,1]))
        B.append(np.array(dst_pts[it][0][0]))
        B.append(np.array(dst_pts[it][0][1]))
    X1=np.linalg.lstsq(A,B)[0]
    avg=0


    H=np.array([X1[0],X1[1],X1[4],X1[2],X1[3],X1[5]]).reshape(2,3)
    print(H)
    dst = cv2.warpAffine(img1,H,(int(cols*1.5),rows))
    cv2.imshow('Input', img1)
    cv2.waitKey(0)
    cv2.imshow('Output', dst)
    cv2.waitKey(0)
    cv2.destroyWindow('Input')
    cv2.destroyWindow('Output')

if len(sys.argv) != 3:
    sys.stderr.write("usage: compare.py <queryImageFile> <sourceImageFile>\n")
    sys.exit(-1)

compare(sys.argv[1], sys.argv[2])
```

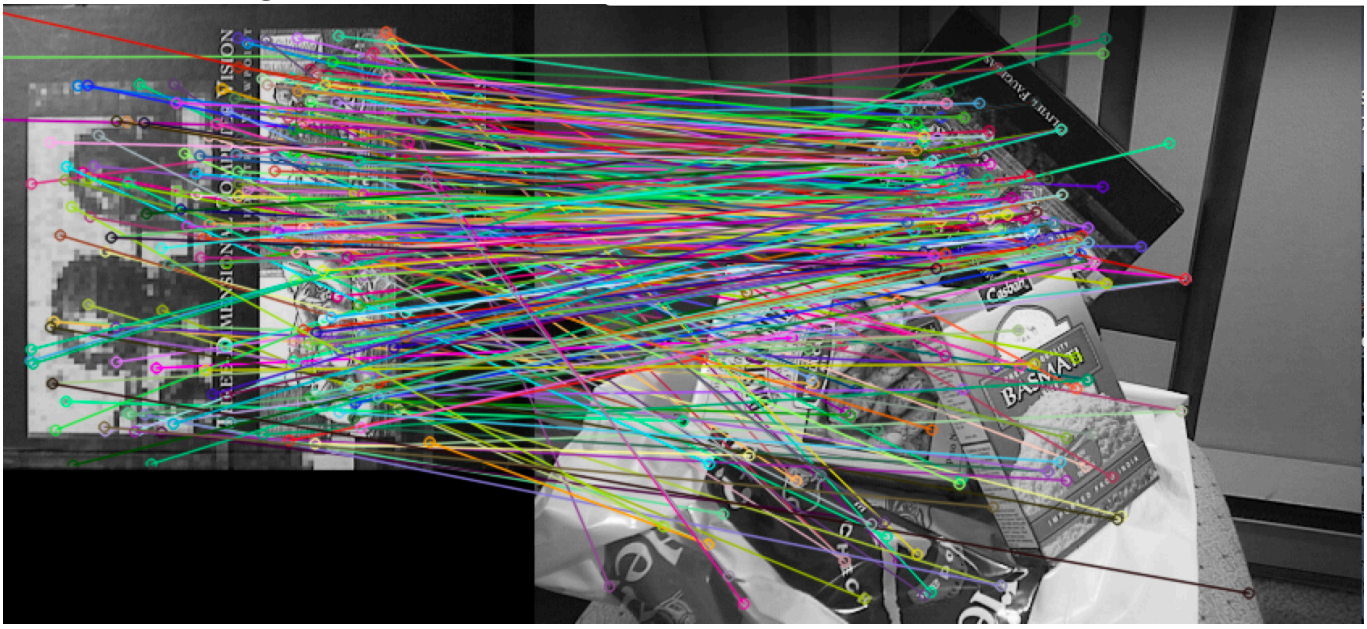
END CODE:

RESULTS:

Local Image Regions:



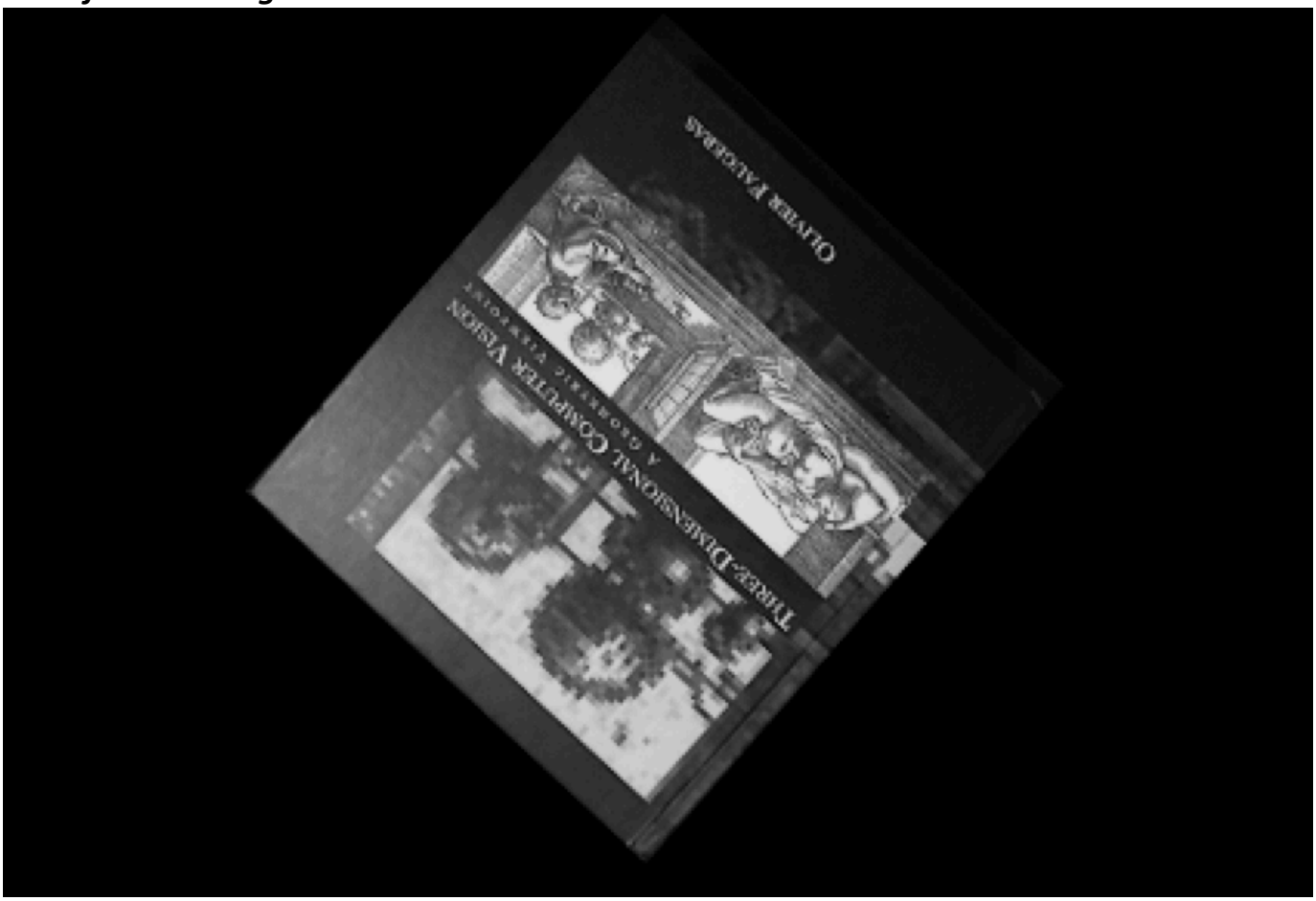
Feature Matching:



Transformation Matrix :

```
[[   0.39999353    0.44917999  135.63981701]
 [  -0.45888482    0.41802308  155.92421455]]
```

Transformed Image:



3)CODE:

```
% clear all;
% close all;
% clc;
clear all;
close all;
clc;
fileID                                                              =
fopen('/Users/srinidhigoud/Desktop/assignment1/world.txt','r');
formatSpec = '%lf';
sizeA = Inf;
C = fscanf(fileID,formatSpec,sizeA);
buf=ones(10,1,'double');
X=[C; buf];%Homogeneous form
X=reshape(X,10,4);
fclose(fileID);
fileID                                                              =
fopen('/Users/srinidhigoud/Desktop/assignment1/image.txt','r');
formatSpec = '%lf';
sizeA = Inf;
B = fscanf(fileID,formatSpec,sizeA);
buf=ones(10,1,'double');
x=[B; buf];%Homogeneous form
x=reshape(x,10,3);
fclose(fileID);
A=zeros(20,12,'double');
j=1;
%Filling matrix A
for i=1:10
    A(j,[5:8])=-1*x(i,3)*X(i,[1:4]);
    A(j,[9:12])=x(i,2)*X(i,[1:4]);
    A(j+1,[1:4])=x(i,3)*X(i,[1:4]);
    A(j+1,[9:12])=-1*x(i,1)*X(i,[1:4]);
    j=j+2;
end
%Finding SVD to find P
[U,S,V]=svd(A,'econ');
P=-reshape(V([1:12],12),4,3);
P=P.';

x2=(P*X.').';%Projection of world points using matrix P
X2=zeros(10,1,'double');
X2=x2([1:10],3);
for i=1:3
    for j=1:10
        x2(j,i)=x2(j,i)/X2(j);
    end
end
```

```
x_actual_points=x
P_projection_matrix=P
x_points_projected_by_P=x2
[U,S,V]=svd(P);%To get nullspace vector of P
C_H=V([1:4],4);
C_homogeneous_form=C_H
for i=1:4;
    C_H(i)=C_H(i)/C_H(4);
end
C=C_H([1:3]);
C_null_space_of_P_inhomogeneous=C
[R Q]=rq(P(:,[1:3]));%RQ decomposition of matrix P for K*(R|-RC)
%We get RQ of -KR thats why only first three columns of P are taken
%R*Q=-KR
%-KRC is last column of P
%-inv(RQ)*P(last column)=C'
C_2=(-1*inv(R*Q)*P(:,4));%RQ decomposition of matrix
C_from_RQ_decomposition_inhomogeneous=C_2



function [R Q]=rq(A)

[m n]=size(A);
if m>n
    error('RQ: Number of rows must be smaller than column');
end

[Q R]=qr(flipud(A).');
R=flipud(R.');
R(:,1:m)=R(:,m:-1:1);
Q=Q.';
Q(1:m,:)=Q(m:-1:1,:);

end
```

END CODE:

RESULTS:

x_actual_points =

```
   5.1177    4.7654    1.0000
   5.5237    3.8703    1.0000
   7.1631    7.3594    1.0000
   5.2222    4.4280    1.0000
   5.6048    4.6748    1.0000
  13.5949   10.0522    1.0000
   8.7345    5.5642    1.0000
   6.2243    3.9082    1.0000
   9.7476    6.9042    1.0000
   5.0903    4.5509    1.0000
```

P_projection_matrix =

```
   0.1270    0.2540    0.3810    0.5080
   0.5080    0.3810    0.2540    0.1270
   0.1270         0    0.1270   -0.0000
```

x_points_projected_by_P =

```
   5.1177    4.7654    1.0000
   5.5237    3.8703    1.0000
   7.1631    7.3594    1.0000
   5.2222    4.4280    1.0000
   5.6048    4.6748    1.0000
  13.5949   10.0522    1.0000
   8.7345    5.5642    1.0000
   6.2243    3.9082    1.0000
   9.7476    6.9042    1.0000
   5.0903    4.5509    1.0000
```

C_homogeneous_form =

-0.5000
0.5000
0.5000
-0.5000

C_null_space_of_P_inhomogeneous =

1.0000
-1.0000
-1.0000

C_from_RQ_decomposition_inhomogeneous =

1.0000
-1.0000
-1.0000

4)

CODE:

```
clear all;
close all;
clc;
S=load('sfm_points.mat');
W=zeros(20,600);
t=zeros(10,2);
l=1;
X=zeros(2,600);
figure
%Just to see if the data will look like a cube finally
for k=1:10
    for i=1:600
        X([1:2],i)=S.image_points(:,i,k);
    end
    subplot(5,2,k)
    plot(X(1,:),X(2,:))
end
%Recentering the data
```

```
for i=1:10
    sum=zeros(2,1);
    for j=1:600
        y=S.image_points(:,j,i);
        sum=sum+y;
    end

    sum=sum/600;
    t(i,[1:2])=sum;%Matrix t
    for j=1:600
        W([l:l+1],j)=S.image_points(:,j,i)-sum;
    end
    l=l+2;
end
t
[U S V]=svd(W);
M_of_2x3_elements_for_10_images=U(:,[1:3])*S([1:3],[1:3])%Matrix
M
rotate3d on
figure
plot3(V(:,1),V(:,2),V(:,3))
```

END CODE:

RESULTS:

t^i is printed in 10x2 matrix with columns corresponding to x y coordinates of x^i

```
t =

  1.0e-15 *

   0.0550  -0.0070
   0.0331   0.0040
  -0.1061   0.0125
   0.0427  -0.0186
  -0.0009  -0.0399
  -0.0775   0.0884
   0.0122   0.0637
   0.0050   0.0027
  -0.0094  -0.0215
  -0.0021   0.0386
```

NOTE: M^i below is 10 2x3 matrices concatenated column wise:

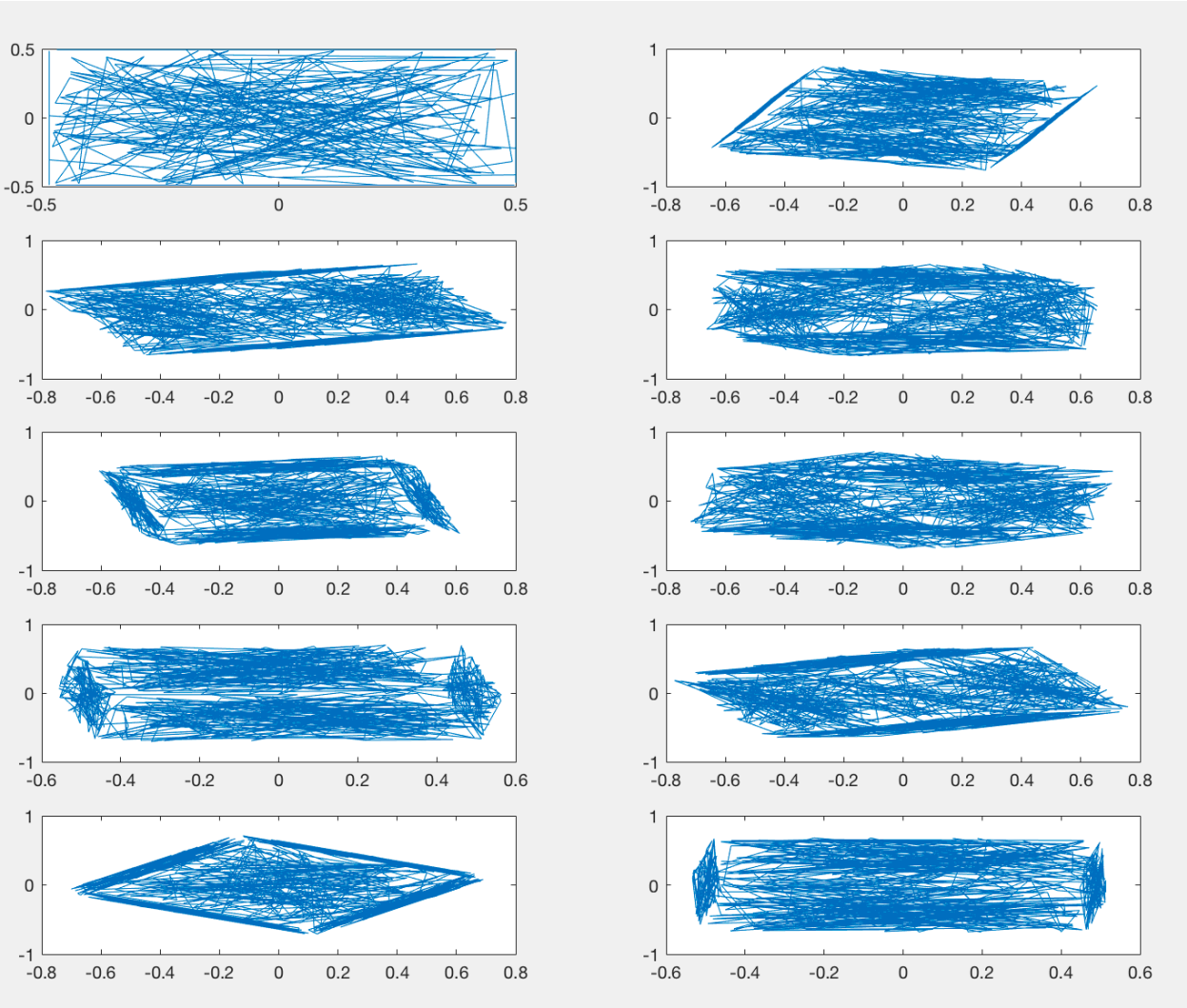
M_of_2x3_elements_for_10_images =

-7.5091	3.3084	-3.7176
-4.5375	-1.5777	7.7457
0.1786	-8.5662	-2.4759
9.0517	0.1260	0.7059
8.2531	2.1691	-3.4821
-0.1313	-7.6818	-4.3252
-3.7683	-8.3478	1.2009
8.2760	-3.5067	0.5700
-0.7346	-8.3978	-2.8898
-8.5004	1.6053	-2.5525
8.4569	-2.5653	-1.7939
-3.2895	-6.1037	-5.4464
-2.9667	-7.7884	-3.2299
8.4511	-1.6413	-2.7808
-1.4368	-8.6231	3.0768
-7.9514	-0.2371	-4.1743
8.6278	-2.1233	-1.6361
-0.4175	4.1054	-8.1481
7.4426	-3.7773	3.4002
-5.2285	-5.8248	5.1158

3D co-ordinates of first 10 points X Y Z

0.0058	0.0646	-0.0250
0.0006	0.0689	-0.0346
-0.0429	0.0633	0.0286
0.0475	0.0490	-0.0126
-0.0421	0.0679	0.0118
0.0596	0.0461	-0.0144
0.0091	0.0600	-0.0123
0.0104	0.0460	0.0353
-0.0259	0.0570	0.0334
0.0175	0.0405	0.0473

Data for all 10 frames are plotted to give us an idea of how the projected data might look.



Final Projected DATA:

Note: In Matlab to rotate the data turn 3D-on manually, even though it is included in the code. Click on the 3D-on button on the figure toolbar.