SarahBrown - CV - HW4

April 30, 2021

0.1 In this assignment, we will get acquainted with essential matrix and fundamental matrix. And recover the essential matrix from matched points of two scene with OpenCV. Furthermore, we will try to localize these matched points in the 3D space

[106]:
```
# Read matched points from two scenes

import pickle
with open('data_points.pickle','rb') as handle:
    data=pickle.load(handle)
```

[107]:
```
# Find fundamental matrix from the match points using OpenCV
import cv2

pts1 = data['x1']
pts2 = data['x2']

F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_8POINT)
pts1 = pts1[mask.ravel()==1] # remove outliers by only keeping inliers
pts2 = pts2[mask.ravel()==1] #
```

[108]:
```
# "Find" essential matrix
import numpy as np

K = np.eye(3) # Take the camera intrisic matrix matrix K to be identity.
E = K.T @ F @ K # The essential matrix is just the same as the fundamental␣
 ↪matrix when K = I

#E, mask = cv2.findEssentialMat(pts1,pts2,K,cv2.RANSAC) # alternatively, we may␣
 ↪try to find essential matrix directly
```

[109]:
```
# Sanity check of the essential matrix
import numpy as np
hx1=np.hstack((pts1,np.ones((2000,1))))
hx2=np.hstack((pts2,np.ones((2000,1))))

rint=np.random.randint(hx1.shape[0])
hx2[rint].T @ E @hx1[rint] # this should be almost zero
```

[109]: 2.909400498296577e-11

1 Q1 (5 points) Compute a potential solution of R and t from E. Note that you will only get half of the points if you use cv2.decomposeEssentialMat

[110]:
```
import numpy as np

# this function should be helpful. You probably want to call the function below
↪instead of np.linalg.svd
def mySVD(E): # compute SVD E = U S V and enforcing det(U)=det(V)=1
    U,S,V = np.linalg.svd(E)
    detU=np.linalg.det(U)
    detV=np.linalg.det(V)
    U=U/detU
    V=V/detV
    S=S*detU*detV
    return U,S,V

def compute_one_R_and_t_from_E(E):
    # Input:
        # E: essential matrix
    # Output:
        # R: rotation matrix (3x3)
        # t: translation (3x1)
    W = np.array([[0,-1,0],
                  [1,0,0],
                  [0,0,1]])
    #from Simon J. D Prince CV Text book equations 16.18 and 16.19
    #U, L, VT = np.linalg.svd(E)
    #tx = U @ np.diag(L) @ W @ np.transpose(U)
    #t = np.transpose(np.array([tx[2,1],tx[0,2],tx[1,0]]))
    #R = U @ np.linalg.inv(W) @ (VT)

    #from slides
    U, S, VT = mySVD(E)
    tx = np.transpose(VT) @ (W) @ np.diag(S) @ VT
    t = np.transpose(np.array([((tx[1,2]-tx[2,1])/2),((tx[2,0]-tx[0,2])/
↪2),((tx[0,1]-tx[1,0])/2)]))
    R = U @ np.transpose(W) @ VT

    return R,t
```

1.1 Testing solution of Q.1

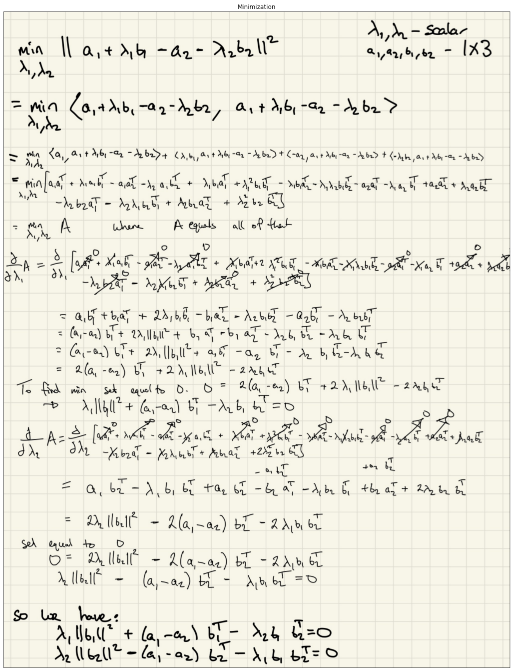
[111]:
```
R,t = compute_one_R_and_t_from_E(E)

tx= np.array([[0,t[2],-t[1]],
              [-t[2],0,t[0]],
              [t[1],-t[0],0]])
R@tx-E
```

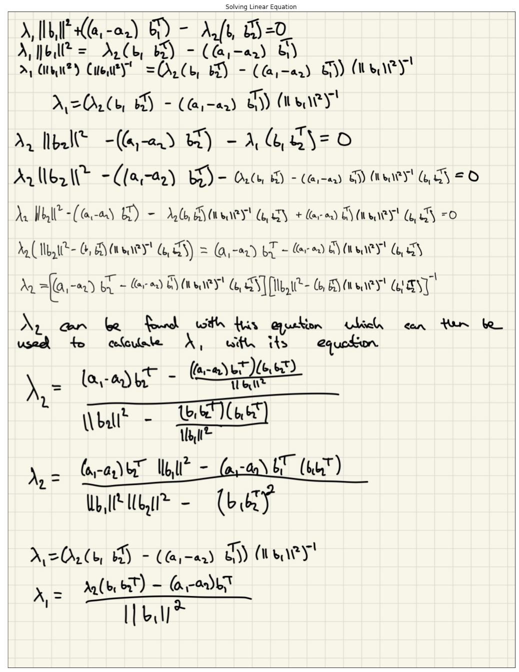
[111]:
```
array([[-1.03683028e-06,  1.28662268e-05,  1.28662304e-05],
       [-1.28662298e-05, -5.18415111e-07, -5.18415249e-07],
       [ 1.28662273e-05,  5.18415023e-07,  5.18415160e-07]])
```

2 Q2.a (5 points) For two lines $\mathbf{a_1} + \lambda_1\mathbf{b_1}$ and $\mathbf{a_2} + \lambda_2\mathbf{b_2}$ in the 3-D space parametrized by λ_1 and λ_2 ($\mathbf{a_1}, \mathbf{a_2}, \mathbf{b_1}, \mathbf{b_2}$ are length-3 vectors). Find the intersecting point between the two lines (the mid point between the closest points of the two lines) by derivating the expressions of the optimum λ_1 and λ_2.

[112]:
```
from matplotlib import pyplot as plt
import cvui
%matplotlib inline

math1 = cv2.imread('2a-1.jpg')
math2 = cv2.imread('2a-2.jpg')

math1 = cv2.cvtColor(math1, cv2.COLOR_BGR2RGB)
math2 = cv2.cvtColor(math2, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(40,40))
plt.subplot(121), plt.imshow(math1), plt.title("Minimization"), plt.xticks([]),␣
 ↪plt.yticks([])
plt.subplot(122), plt.imshow(math2), plt.title("Solving Linear Equation"), plt.
 ↪xticks([]), plt.yticks([])
```

[112]:
```
(<AxesSubplot:title={'center':'Solving Linear Equation'}>,
 <matplotlib.image.AxesImage at 0x7fd777063460>,
 Text(0.5, 1.0, 'Solving Linear Equation'),
 ([], []),
 ([], []))
```





3 Q2.b (5 points) Implement the solution of Q2.a by completing the function below

[113]:
```
def computeIntersection(a1,b1,a2,b2):
    # Input:
        # a1: Nx3 matrix  (a1[i] = ith a1)
        # b1: Nx3 matrix  (b1[i] = ith b1)
        # a2: Nx3 matrix  (a2[i] = ith a2)
        # b2: Nx3 matrix  (b2[i] = ith b2)
        # N.B. for the ith pair of lines, line 1: a1[i]+lambda1[i] b1[i] and␣
↪line 2: a2[i]+lambda2[i] b2[i]
    # Output:
        # points: Nx3 matrix (points[i] = the intersecting point for ith pair)
    points = np.empty(shape=np.shape(a1))

    n = len(a1)
    for i in range(n):
        a1i = a1[i]; a2i = a2[i]; b1i = b1[i]; b2i = b2[i]
        b1normsq = np.dot(b1i,b1i); b2normsq = np.dot(b2i, b2i)
        a1a2 = (a1i-a2i)

        lambda2 = (np.dot(a1a2,b2i)*b1normsq - np.dot(a1a2,b1i)*np.
↪dot(b1i,b2i))/(b1normsq*b2normsq - np.dot(b1i,b2i)**2)
        lambda1 = (lambda2*(np.dot(b1i,b2i))-np.dot(a1a2,b1i)) / ((b1normsq))
```

```
        point = (a1i + (lambda1 * b1i) + a2i + (lambda2 * b2i))/2.0
        points[i] = point
    return points
```

3.1 Testing solution of Q2.b

We will take camera center of view 1 as origin, so $\mathbf{a_1} = \mathbf{0}$, $\mathbf{b_1} = [x_1[0], x_1[1], 1]^\top$

And $\mathbf{a_2} = \mathbf{t}$, $b_2 = R[x_2[0], x_2[1], 0]^\top$

[114]:
```
a1 = np.zeros((pts1.shape[0],3))
b1 = np.hstack((pts1,np.ones((pts1.shape[0],1))))
a2 = np.tile(t,(pts1.shape[0],1))
b2 = np.hstack((pts2,np.ones((pts2.shape[0],1))))
b2 = (R.T @ b2.T).T

Xs=(computeIntersection(a1,b1,a2,b2)) # 3D coordinates of points in the first
 ↪camera view
Xps=(R@(Xs-t).T).T # 3D coordinates of points in the second camera view
Xs,Xps
```

[114]:
```
(array([[   -7.43604951,    48.66761496, -2829.18397952],
        [   -8.15478157,    13.76484284, -3733.82471541],
        [  -12.49381847,    15.95569632, -3688.7495735 ],
        …,
        [  -40.50867914,    58.77384587, -7080.43115241],
        [  -52.5995618 ,    60.82071569, -6708.83273944],
        [  -36.81195278,    69.25851921, -5232.83036157]]),
 array([[   7.43603602,  -58.23522507, 2722.28181318],
        [   8.15476354,  -93.13821581, 3626.92254063],
        [  12.49380066,  -90.94735145, 3581.84739928],
        …,
        [  40.50864514,  -48.13002166, 6973.52898867],
        [  52.5995296 ,  -46.0830621 , 6601.93057625],
        [  36.81192776,  -37.64490182, 5125.92820034]]))
```

4 Chirality

4.0.1 Note that Xs and Xps are the points in the 3D space with camera centers as origins and with z-axis pointing from the camera centers to the objects. So the z-component (third column of Xs and Xps) should be both non-negative because the object points suppose to be in front of the cameras. But with only 1/4 of the chance you would be lucky. Because there are four possible combinations of R and t and only one is correct (satisfies chirality).

5 Q2.c (4 points) Find the correct R and t by adjusting your solution in Q1. It is okay to provide a "buggy" solution that only works for the current dataset. Please redefine compute_one_R_and_t_from_E(E) below

[115]:
```
import numpy as np

# this function should be helpful. You probably want to function below instead␣
↪of np.linalg.svd
def mySVD(E): # compute SVD E = U S V and enforcing det(U)=det(V)=1
    U,S,V = np.linalg.svd(E)
    detU=np.linalg.det(U)
    detV=np.linalg.det(V)
    U=U/detU
    V=V/detV
    S=S*detU*detV
    return U,S,V

def compute_one_R_and_t_from_E(E):
    # Input:
        # E: essential matrix
    # Output:
        # R: rotation matrix (3x3)
        # t: translation (3x1)
    W = np.array([[0,-1,0],
                  [1,0,0],
                  [0,0,1]])

    #from slides
    U, S, VT = mySVD(E)
    tx = np.transpose(VT) @ np.transpose(W) @ np.diag(S) @ VT
    t = np.transpose(np.array([((tx[1,2]-tx[2,1])/2),((tx[2,0]-tx[0,2])/
↪2),((tx[0,1]-tx[1,0])/2)]))
    R = U @ (W) @ VT

    return R,t
```

[116]:
```python
# rerun everything below

R,t = compute_one_R_and_t_from_E(E)

tx= np.array([[0,t[2],-t[1]],[-t[2],0,t[0]],[t[1],-t[0],0]])
rint=np.random.randint(hx1.shape[0])
hx2[rint].T @ R@tx @hx1[rint] # this should be almost zero
```

[116]: 4.630454680421177e-07

[117]:
```python
a1 = np.zeros((pts1.shape[0],3))
b1 = np.hstack((pts1,np.ones((pts1.shape[0],1))))
a2 = np.tile(t,(pts2.shape[0],1))
b2 = np.hstack((pts2,np.ones((pts2.shape[0],1))))
b2 = (R.T @ b2.T).T

Xs=computeIntersection(a1,b1,a2,b2)
Xps=(R@(Xs-t).T).T
Xs,Xps # note that third third columns should be positive
```

[117]:
```
(array([[  0.28687998,  -1.8775782 , 109.14884922],
        [  0.23945037,  -0.40417965, 109.63699108],
        [  0.37123112,  -0.47409453, 109.60449159],
        …,
        [  0.61579632,  -0.89345589, 107.63380911],
        [  0.84394685,  -0.97585321, 107.6415475 ],
        [  0.7574854 ,  -1.42514355, 107.67678137]]),
 array([[  0.28687946,  -2.24669299, 105.02460402],
        [  0.23944984,  -2.73483479, 106.49800259],
        [  0.37123059,  -2.70233531, 106.42808771],
        …,
        [  0.6157958 ,  -0.73165284, 106.00872628],
        [  0.84394633,  -0.73939123, 105.92632896],
        [  0.75748488,  -0.77462512, 105.47703862]]))
```

[118]:
```python
# Sanity check: the projection to cameras should get back the original

print(pts1[0],Xs[0]/Xs[0,2]) # the numbers should match if you did correctly
print(pts2[0],Xps[0]/Xps[0,2])
```

```
[ 0.00262833 -0.017202  ] [ 0.00262834 -0.017202    1.        ]
[ 0.00273155 -0.02139206] [ 0.00273155 -0.02139206  1.        ]
```

[119]:
```python
#!pip install open3d # install open3d if you don't have it installed before
# visualize your point cloud

import open3d as o3d
```

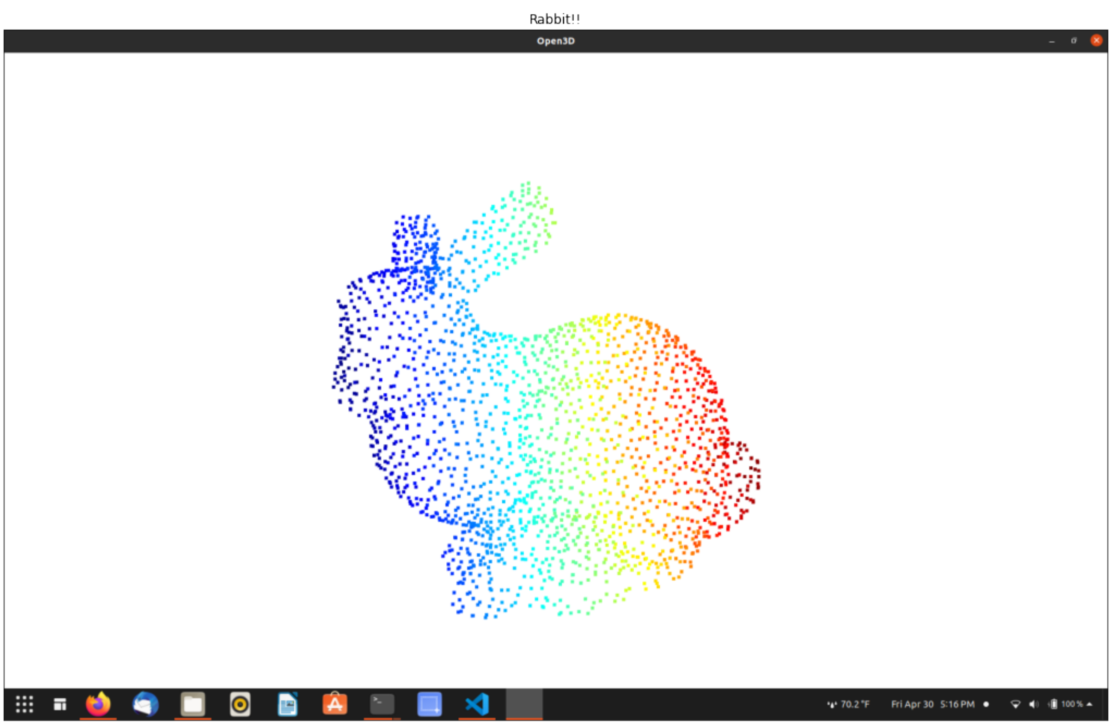
```
cloud=o3d.geometry.PointCloud(o3d.utility.Vector3dVector(Xps))
o3d.visualization.draw_geometries([cloud])
```

6 Q2.d (1 point): What is the object that you are seeing? Click the screen and move the mouse around and you can rotate it

It's a rabbit! The blocks are rainbow colored depending on location.

[120]:
```
rabbit = cv2.imread('rabbit.png')

rabbit = cv2.cvtColor(rabbit, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(40,40))
plt.subplot(121), plt.imshow(rabbit), plt.title("Rabbit!!"), plt.xticks([]),␣
↪plt.yticks([])
```

[120]:
```
(<AxesSubplot:title={'center':'Rabbit!!'}>,
 <matplotlib.image.AxesImage at 0x7fd776f23700>,
 Text(0.5, 1.0, 'Rabbit!!'),
 ([], []),
 ([], []))
```



7 Q3 (10 points, extra credit): Capture two images and try to reconstruct 3D shape with the tools you developed.

It does something, I couldn't get good matches on my pictures of things so I reused the pics I took for homework 3 and got some matches at least.

[121]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm
import scipy.io as sio
%matplotlib inline


img1 = cv2.imread('File_000.jpg')
img2 = cv2.imread('File_001.jpg')

img1c = np.array(img1)
img2c = np.array(img2)

img1c=cv2.resize(img1c,(0,0),fx=0.5,fy=0.5)
img2c=cv2.resize(img2c,(0,0),fx=0.5,fy=0.5)

img1 = cv2.cvtColor(img1c, cv2.COLOR_RGB2GRAY)
img2 = cv2.cvtColor(img2c, cv2.COLOR_RGB2GRAY)
```

[122]:
```
sift = cv2.xfeatures2d.SIFT_create() #SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)

good = []
points1 = []
points2 = []

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.3*n.distance:
```

```
        good.append(m)
        points2.append(kp2[m.trainIdx].pt)
        points1.append(kp1[m.queryIdx].pt)
print(len(good))
```

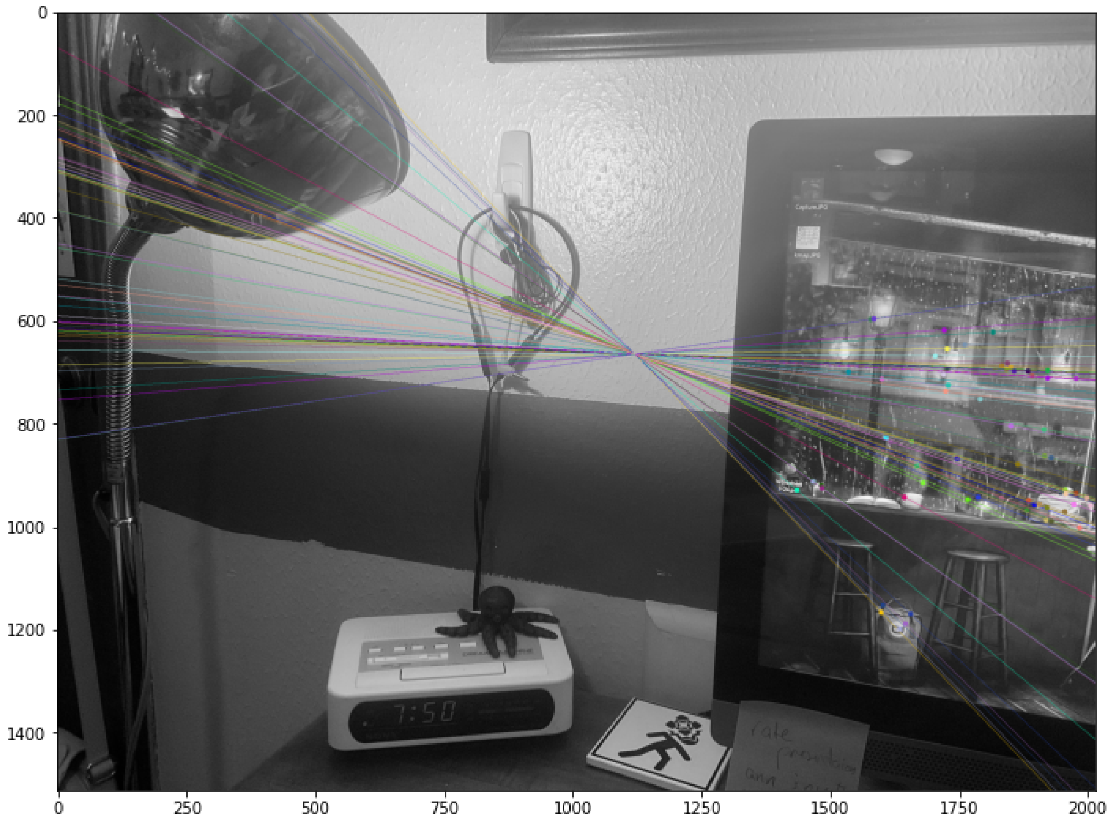
75

[123]:
```
points1 = np.int32(points1)
points2 = np.int32(points2)
F, mask = cv2.findFundamentalMat(points1,points2,cv2.FM_LMEDS)
# We select only inlier points
points1 = points1[mask.ravel()==1]
points2 = points2[mask.ravel()==1]

print(len(points1))

K = np.eye(3) # Take the camera intrisic matrix matrix K to be identity.
E = K.T @ F @ K
```

68

[124]:
```
def drawlines(img1,img2,lines,pts1,pts2):
    ''' img1 - image on which we draw the epilines for the points in img2
        lines - corresponding epilines '''
    r,c = img1.shape
    img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
    img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)
    for r,pt1,pt2 in zip(lines,pts1,pts2):
        color = tuple(np.random.randint(0,255,3).tolist())
        x0,y0 = map(int, [0, -r[2]/r[1] ])
        x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2

# Find epilines corresponding to points in right image (second image) and
# drawing its lines on left image
lines1 = cv2.computeCorrespondEpilines(points2.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
img5,img6 = drawlines(img1,img2,lines1,points1,points2)
# Find epilines corresponding to points in left image (first image) and
# drawing its lines on right image
lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3,img4 = drawlines(img2,img1,lines2,points2,points1)
plt.figure(figsize=(40,20))
```

```
plt.subplot(211),plt.imshow(img5)
plt.subplot(212),plt.imshow(img3)
plt.show()
```



[125]:
```python
import numpy as np

# this function should be helpful. You probably want to function below instead␣
↪of np.linalg.svd
def mySVD(E): # compute SVD E = U S V and enforcing det(U)=det(V)=1
    U,S,V = np.linalg.svd(E)
    detU=np.linalg.det(U)
    detV=np.linalg.det(V)
    U=U/detU
    V=V/detV
    S=S*detU*detV
    return U,S,V

def compute_one_R_and_t_from_E(E):
    # Input:
        # E: essential matrix
    # Output:
        # R: rotation matrix (3x3)
        # t: translation (3x1)
    W = np.array([[0,-1,0],
                  [1,0,0],
                  [0,0,1]])

    #from slides
    U, S, VT = mySVD(E)
    tx = np.transpose(VT) @ np.transpose(W) @ np.diag(S) @ VT
    t = np.transpose(np.array([((tx[1,2]-tx[2,1])/2),((tx[2,0]-tx[0,2])/
↪2),((tx[0,1]-tx[1,0])/2)]))
    R = U @ np.transpose(W) @ VT

    return R,t
```

[126]:
```python
R,t = compute_one_R_and_t_from_E(E)

tx= np.array([[0,t[2],-t[1]],
              [-t[2],0,t[0]],
              [t[1],-t[0],0]])
R@tx-E
```

[126]:
```
array([[-2.52325830e-01,  4.25325717e-01, -1.07580466e-03],
       [-3.76059159e-02,  6.33858196e-02,  2.17051297e-03],
       [ 1.26915358e-03,  1.14345838e-04, -1.50000051e+00]])
```

[127]:
```python
R,t = compute_one_R_and_t_from_E(E)
hx1=np.hstack((points1,np.ones((len(points1),1))))
hx2=np.hstack((points2,np.ones((len(points2),1))))

rint=np.random.randint(hx1.shape[0])
hx2[rint].T @ E @hx1[rint] # this should be almost zero
tx= np.array([[0,t[2],-t[1]],[-t[2],0,t[0]],[t[1],-t[0],0]])
rint=np.random.randint(hx1.shape[0])

hx2[rint].T @ R@tx @hx1[rint] # this should be almost zero
```

[127]:
```
-83264.21678039154
```

[128]:
```python
a1 = np.zeros((points1.shape[0],3))
b1 = np.hstack((points1,np.ones((points1.shape[0],1))))
a2 = np.tile(t,(points2.shape[0],1))
b2 = np.hstack((points2,np.ones((points2.shape[0],1))))
b2 = (R.T @ b2.T).T

Xs=computeIntersection(a1,b1,a2,b2)
Xps=(R@(Xs-t).T).T
Xs,Xps # note that third third columns should be positive
```

[128]:
```
(array([[-4.22696758e-01, -2.61971844e-01,  3.60352655e-04],
        [-4.22696758e-01, -2.61971844e-01,  3.60352655e-04],
        [-4.26747348e-01, -2.59106460e-01, -1.23602665e-04],
        [-4.26943405e-01, -2.58949910e-01, -1.40691786e-04],
        [-4.26562427e-01, -2.59339817e-01, -1.06091834e-04],
        [-4.35089466e-01, -2.27441480e-01,  2.83802446e-03],
        [-4.36101947e-01, -2.10327569e-01,  5.85130101e-03],
        [-4.36285441e-01, -2.09887041e-01,  5.89247886e-03],
        [-4.36285441e-01, -2.09887041e-01,  5.89247886e-03],
        [-4.08489163e-01, -2.69091075e-01,  2.88851661e-03],
        [-4.36379656e-01, -2.25888983e-01,  2.89109636e-03],
        [-4.10722964e-01, -2.68627301e-01,  2.43652904e-03],
        [-4.34988526e-01, -2.40544149e-01,  7.16617417e-04],
        [-4.34928693e-01, -2.41116444e-01,  6.42725652e-04],
        [-4.08346013e-01, -2.69665331e-01,  2.92116746e-03],
        [-4.31636502e-01, -2.51735061e-01, -1.99659703e-04],
        [-4.10820009e-01, -2.69201561e-01,  2.41505078e-03],
        [-4.14054277e-01, -2.68245282e-01,  1.77938657e-03],
        [-4.35623061e-01, -2.38633155e-01,  8.99375440e-04],
        [-4.34563843e-01, -2.43542244e-01,  3.58960591e-04],
        [-4.33690964e-01, -2.46569742e-01,  9.53411264e-05],
        [-4.38608342e-01, -2.14693295e-01,  4.40096857e-03],
        [-4.38544893e-01, -2.18648732e-01,  3.70073269e-03],
        [-4.39425863e-01, -2.07475730e-01,  5.44233553e-03],
```

```
       [-4.38220575e-01, -2.22540460e-01,  3.08390108e-03],
       [-4.39159081e-01, -2.11880892e-01,  4.74569808e-03],
       [-4.38760130e-01, -2.17037033e-01,  3.93308351e-03],
       [-4.38456980e-01, -2.20911879e-01,  3.31541313e-03],
       [-4.36291424e-01, -2.37405127e-01,  9.66981041e-04],
       [-4.39810434e-01, -2.09448786e-01,  4.96481838e-03],
       [-4.34348861e-01, -2.45017288e-01,  1.97736655e-04],
       [-4.34867531e-01, -2.43477180e-01,  3.21303756e-04],
       [-4.38998448e-01, -2.21311364e-01,  3.09549932e-03],
       [-4.40906655e-01, -2.04324536e-01,  5.42554398e-03],
       [-4.35613676e-01, -2.41240954e-01,  5.09056169e-04],
       [-4.40389769e-01, -2.11567416e-01,  4.38931409e-03],
       [-4.40413675e-01, -2.12035155e-01,  4.29873988e-03],
       [-4.40413675e-01, -2.12035155e-01,  4.29873988e-03],
       [-4.40594571e-01, -2.10517731e-01,  4.49730619e-03],
       [-4.40562368e-01, -2.11180202e-01,  4.39416511e-03],
       [-4.39010483e-01, -2.24651114e-01,  2.49424954e-03],
       [-4.39010483e-01, -2.24651114e-01,  2.49424954e-03],
       [-4.40546222e-01, -2.11767241e-01,  4.29763752e-03],
       [-4.37780510e-01, -2.32409490e-01,  1.46084874e-03],
       [-4.35763283e-01, -2.41054159e-01,  5.08427970e-04],
       [-4.40914616e-01, -2.10694644e-01,  4.34460075e-03],
       [-4.36372981e-01, -2.38878071e-01,  7.18027991e-04],
       [-4.40889011e-01, -2.11423242e-01,  4.22828207e-03],
       [-4.36325298e-01, -2.39144877e-01,  6.86346715e-04],
       [-4.36485215e-01, -2.38635898e-01,  7.32079451e-04],
       [-4.39673048e-01, -2.22483073e-01,  2.68316201e-03],
       [-4.38787225e-01, -2.28103498e-01,  1.94404947e-03],
       [-4.35835019e-01, -2.41030122e-01,  4.98136831e-04],
       [-4.36551264e-01, -2.38492655e-01,  7.40557399e-04],
       [-4.41182771e-01, -2.10814413e-01,  4.21783078e-03],
       [-4.41339949e-01, -2.09076113e-01,  4.45130609e-03],
       [-4.35943409e-01, -2.40724607e-01,  5.22332767e-04],
       [-4.38895159e-01, -2.27820263e-01,  1.96327354e-03],
       [-4.37688506e-01, -2.34161750e-01,  1.18499012e-03],
       [-4.36858942e-01, -2.37616759e-01,  8.11645768e-04],
       [-4.37939888e-01, -2.33151346e-01,  1.29270700e-03],
       [-4.37759033e-01, -2.33930570e-01,  1.20651138e-03],
       [-4.37484310e-01, -2.35321491e-01,  1.04021677e-03],
       [-4.38050874e-01, -2.32869372e-01,  1.31215280e-03],
       [-4.41724495e-01, -2.08795763e-01,  4.33432589e-03],
       [-4.36454065e-01, -2.39284277e-01,  6.36624449e-04],
       [-4.38126477e-01, -2.32803810e-01,  1.30294367e-03],
       [-4.37870903e-01, -2.33996537e-01,  1.16580097e-03]]),
array([[ 0.00031223,  0.0028931 , -0.00963267],
       [ 0.00031223,  0.0028931 , -0.00963267],
       [ 0.00013707,  0.00081557, -0.00510441],
```

```
[ 0.00013352,  0.00072494, -0.00486986],
[ 0.00014821,  0.00085812, -0.00539941],
[ 0.00176256,  0.01009204,  0.02639699],
[ 0.00359108,  0.01829117,  0.04164237],
[ 0.00362229,  0.01836281,  0.04211494],
[ 0.00362229,  0.01836281,  0.04211494],
[ 0.0015287 ,  0.01177867, -0.02299297],
[ 0.00186382,  0.00978307,  0.02838996],
[ 0.00133262,  0.01004357, -0.02145667],
[ 0.00063239,  0.0032682 ,  0.01506789],
[ 0.0005945 ,  0.00302012,  0.01454484],
[ 0.00158567,  0.0116163 , -0.02356018],
[ 0.00013526,  0.00035281,  0.00372922],
[ 0.00136677,  0.00966871, -0.02190161],
[ 0.00108009,  0.00730264, -0.01943208],
[ 0.00075123,  0.00371705,  0.01703572],
[ 0.00044879,  0.00206759,  0.01227116],
[ 0.00030307,  0.00124765,  0.00922098],
[ 0.002804  ,  0.01374509,  0.03916069],
[ 0.00239945,  0.01170432,  0.03572391],
[ 0.00339818,  0.01683653,  0.04578911],
[ 0.00203925,  0.00993004,  0.03220917],
[ 0.00300484,  0.01474301,  0.04186172],
[ 0.00253612,  0.01236679,  0.03722069],
[ 0.00217653,  0.01058287,  0.03373125],
[ 0.00081163,  0.00377625,  0.01843292],
[ 0.00312279,  0.01544523,  0.04428668],
[ 0.00037226,  0.00148423,  0.01089211],
[ 0.00044551,  0.00183624,  0.01248174],
[ 0.00205818,  0.00988826,  0.03366306],
[ 0.00333505,  0.01715969,  0.04925533],
[ 0.00055921,  0.00235453,  0.01478633],
[ 0.00278629,  0.01380034,  0.0427581 ],
[ 0.00273478,  0.01353113,  0.04236768],
[ 0.00273478,  0.01353113,  0.04236768],
[ 0.00284071,  0.01417038,  0.04376585],
[ 0.00278417,  0.01384904,  0.04317926],
[ 0.00171509,  0.00810782,  0.03079463],
[ 0.00171509,  0.00810782,  0.03079463],
[ 0.00273059,  0.01355299,  0.04266577],
[ 0.00111653,  0.00509644,  0.02349077],
[ 0.00056374,  0.00232111,  0.01502327],
[ 0.00274385,  0.01378625,  0.04377653],
[ 0.00068611,  0.00292851,  0.01720664],
[ 0.00268008,  0.01342406,  0.04313638],
[ 0.00066865,  0.00283011,  0.01695272],
[ 0.00069623,  0.00295695,  0.01747221],
```

```
        [ 0.00182436,  0.0086631 ,  0.032998  ],
        [ 0.00140082,  0.00647854,  0.02770948],
        [ 0.00056094,  0.00227061,  0.01508047],
        [ 0.00070234,  0.00297408,  0.01762913],
        [ 0.00266173,  0.01347891,  0.04380998],
        [ 0.00278262,  0.0142547 ,  0.04538618],
        [ 0.00057586,  0.00233571,  0.01539861],
        [ 0.00141244,  0.00653207,  0.02800821],
        [ 0.00096311,  0.00425186,  0.02193573],
        [ 0.00074642,  0.0031636 ,  0.01853965],
        [ 0.00102615,  0.00456243,  0.02293337],
        [ 0.00097611,  0.00431138,  0.02217062],
        [ 0.00088063,  0.00382046,  0.02083358],
        [ 0.00103847,  0.00461277,  0.02323257],
        [ 0.00269512,  0.01405113,  0.04582329],
        [ 0.00064637,  0.00264295,  0.01689828],
        [ 0.00103412,  0.00458005,  0.02332749],
        [ 0.00095509,  0.00417691,  0.02217079]]))
```

[129]:
```
cloud=o3d.geometry.PointCloud(o3d.utility.Vector3dVector(Xps))
o3d.visualization.draw_geometries([cloud])
```

[130]:
```
thing = cv2.imread('something.png')

thing = cv2.cvtColor(thing, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(40,40))
plt.subplot(121), plt.imshow(thing), plt.title("Something?"), plt.xticks([]),
 ↪plt.yticks([])
```

[130]:
```
(<AxesSubplot:title={'center':'Something?'}>,
 <matplotlib.image.AxesImage at 0x7fd7a0a3d7f0>,
 Text(0.5, 1.0, 'Something?'),
 ([], []),
 ([], []))
```

Something?