Solution1

Theory:

Epipolar line is the line joining the epi-poles to the image point projected by the 3d point x in the other camera. It is formed by the intersection of image plane and the epi-polar plane. Thus, every point in the image plane1 has corresponding epi-polar line in the 2nd image plane. All epi-polar lines pass through the epi-pole of that image plane. The point lying in image1 has its corresponding point in image2 always lying on the epi-polar line in image2.

Epipole: It is the projection of the projection centre of 2nd camera into the 1st camera

- **Finding the epipolar lines**

- The equation below defines a mapping between points and epipolar lines:

$$\mathbf{x'^T F x = 0}$$

Case 1: right epipolar line u_r

the right epipolar line is represented by $u_r = F\mathbf{x}$

$\mathbf{x'}$ lies on u_r, that is, $\mathbf{x'}^T u_r = 0$ or $\mathbf{x'^T F x = 0}$

Case 2: left epipolar line u_l

$\mathbf{x'^T F x = 0}$ is equivalent to $\mathbf{x'^T F x = 0}$

the left epipolar line is represented by $u_l = F^T \mathbf{x}$

$\mathbf{x'}$ lies on u_l, that is, $\mathbf{x'}^T u_l = 0$ or $\mathbf{x'^T F x = 0}$

• Locating the epipoles from F (or E)

Case 1: locate $\bar{e}_l$

$\bar{e}_l$ lies on all epipolar lines in the left image, thus, it satisfies the equation:

$$\bar{e}_l^T u_l = 0 \quad \text{or} \quad u_l^T \bar{e}_l = 0 \quad \text{or} \quad \mathbf{x'^T F \bar{e}_l = 0}$$

which leads to the following homogeneous system:

$$\mathbf{F\bar{e}_l = 0}$$

We can obtain $\bar{e}_l$ by solving the above homogeneous system (the solution $\bar{e}_l$ is proportional to the last column of V of the SVD of F).

Case 2: locate $\bar{e}_r$

$\bar{e}_r$ lies on all epipolar lines in the right image, thus, it satisfies the equation:

$$\bar{e}_r^T u_r = 0 \quad \text{or} \quad \bar{e}_r^T F \mathbf{x} = 0 \quad \text{or} \quad \mathbf{x'^T F^T \bar{e}_r = 0}$$

which leads to the following homogeneous system:

$$\mathbf{F^T \bar{e}_r = 0}$$

The solution is proportional to the last column of V of the SVD of F^T (i.e., same as the last column of U of the SVD of F).

$$F^T = VDU^T$$

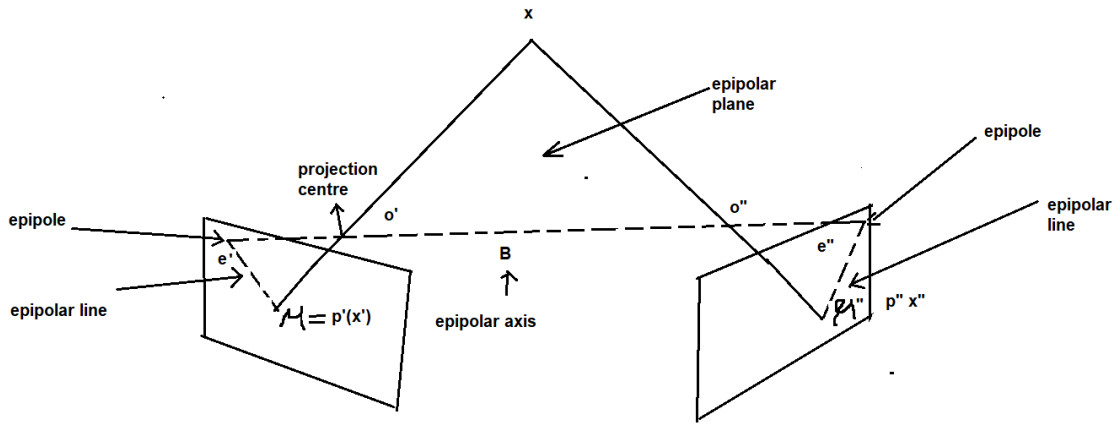

Fig-1 Epipolar Geometry

Algorithm:

1. Store F
2. Store points image1 in a numpy 2D array
3. Get the epipolar lines if point in first image by multiplying F^T every time with points of second image.
4. Store it as array of lines for image1.
5. Get epipolar lines of points in second image by multiplying F with each point in first image Fx^T
6. Store it as array of lines to be plotted on second after converting it to image
7. Obtain SVD on F last column of v is proportional to null space and that is the Epipole for the first image
8. Obtain SVD on F^T last column of v is proportional to null space and that is the Epipole for the of second image

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
F= np.array([[-1.29750186e-06,  8.07894025e-07,  1.84071967e-03],
             [3.54098411e-06,  1.05620725e-06, -8.90168709e-03],
             [-3.29878312e-03,  5.14822628e-03,  1.00000000e+00]])
img1 = np.array([[381,402,1],
 [452,497,1],
 [671,538,1],
 [501,254,1],
 [506,381,1],
 [474,440,1],
 [471,537,1],
 [498,364,1],
 [706,319,1],
 [635,367,1]])
print(F.shape)
f= F.transpose()
p=0
img2= np.array([[390,346,1],
 [439,412,1],
 [651,417,1],
 [477,194,1],
```

```
 [482,300,1],
 [456,359,1],
 [454,444,1],
 [475,287,1],
 [686,185,1],
 [606,253,1]])



E1= np.zeros((10,3))
for i in range(10):
    E1[i]= np.matmul(f,img2[i])

E2= np.zeros((10,3))
for i in range(10):
#     E2[i]= np.matmul(img1[i],F)
    E2[i]= np.matmul(F,img1[i])

Ep1 = np.zeros((10,2))
for i in range(10):
    Ep1[i] = [(E1[i][0]/E1[i][1]),(E1[i][2]/E1[i][1])]
    Ep1[i][1] =Ep1[i][1]+p

Ep1 *= -1
Ep2 = np.zeros((10,2))
for i in range(10):
    Ep2[i] = [(E2[i][0]/E2[i][1]),(E2[i][2]/E2[i][1])]
    Ep2[i][1] =Ep2[i][1]+p
Ep2 *= -1


y1=np.zeros((10,1200))
x = np.zeros((1200,1))
for i in range(1200):
    x= np.linspace(0,1200,1200)
img=mpimg.imread('img1.jpg')
for i in range(10):
    y1[i] = Ep1[i][0]*x + Ep1[i][1]
    plt.plot(img1[i][0],img1[i][1],'go')
    plt.plot(x, y1[i], '-g')
plt.imshow(img)
plt.show()
```

```
y2=np.zeros((10,1200))
x = np.zeros((1200,1))
for i in range(1200):
    x= np.linspace(0,1200,1200)
img=mpimg.imread('img2.jpg')
for i in range(10):
    y2[i] = Ep2[i][0]*x + Ep2[i][1]
    plt.plot(x, y2[i], '-g')
    plt.plot(img2[i][0],img2[i][1],'go')
plt.imshow(img)
plt.show()
```

```
#Epipole Calculation

#Epipole of first image is found by taking SVD of F

(U, W, V) = np.linalg.svd(F, full_matrices=True)

V=V.transpose()

ep1=V[:,-1]

print("epipole for first image is",ep1)

#Epipole of second image is found by taking SVD of transpose of F

(P, Q, R) = np.linalg.svd(f, full_matrices=True)

R=R.transpose()


ep2=R[:,-1]

print("epipole for second image  is",ep2)
```

b)

Epipole for 1st image is [8.75919893e-01 4.82456398e-01 4.05675052e-04]

Epipole for 2nd image is [-9.83333374e-01 -1.81811546e-01 1.91611592e-04]

Solution 2:

Theory:

Visual odometry (VO) is the process of recovering the ego-motion (in other words, the trajectory) of anagent using only the input of a camera or a system of cameras attached to the agent.In this problem, we've been provided 801 images which are taken from a camera mounted on the top of a moving vehicle at regular intervals. We're supposed to find the trajectory of the car by processing the images provided using the methods of Monocular Visual Odometry. As the absolute translation from the camera is not known in monocular case, we've been provided with ground truth to retrieve the same.

Algorithm:

1. Find corresponding features between frames *Ik; Ik*-1 :

- 1st step is to load all 801 images in the program in grey scale form and store them as an array having 801 rows.
- By the function p0 = cv2.goodFeaturesToTrack(i0,500,.1,50) we extracted similar features from a pair of images
- Then we calculated optical flow between the obtained correspondence by the function p1, status, _ = cv2.calcOpticalFlowPyrLK(i0,i1,p0, **None)**

2. Using these feature correspondences, estimate the essential matrix between the two images withina RANSAC scheme.

- For estimating the Essential matrix,1st we found the Fundamental matrix under RANSAC scheme.
- For estimating under RANSAC scheme, 1st we chose random nine points correspondence between an image pair and estimated fundamental matrix for these nine points using 8-point algorithm.

- **In 8-point algorithm**

$$\boldsymbol{x}_m'^{\top}\mathbf{F}\boldsymbol{x}_m = 0$$

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

ONE correspondence gives you ONE equation

$$\begin{aligned} x_m x'_m f_1 + x_m y'_m f_2 + x_m f_3 + \\ y_m x'_m f_4 + y_m y'_m f_5 + y_m f_6 + \\ x'_m f_7 + y'_m f_8 + f_9 = 0 \end{aligned}$$

Set up a homogeneous linear system with 9 unknowns

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_Mx'_M & x_My'_M & x_M & y_Mx'_M & y_My'_M & y_M & x'_M & y'_M & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = \mathbf{0}$$

We normalize the obtained matrix column wise and applied SVD to obtain $M = U\,S\,V^T$. For applying rank 2 constraint we made last element of S to be 0, and again multiplied it with U & V^T to get the F-matrix.

- Then we calculated F matrix for thousands sets of nine random correspondence points.
- For estimating the most error less F out of the thousands obtained, on each F weapplied the co-planarity constraint w.r.t all the correspondences obtained for an image pair. We chose F which was mostly following the co-planarity constraints with most of the points by choosing F having minimum value for x'Fx= 0.
- After obtaining the F matrix, we calculated Essential matrix by $K^{-T}FK^{-1}$; where K is camera calibration matrix.

3. Decompose this essential matrix to obtain the relative rotation *Rk* and translation *tk*, and formthe transformation *Tk*.
   - Adopting Essential matrix by decomposing the function cv2.recoverPose(E, points1, points2,R, t,focal,pp,mask=**None**)
   - Here we obtain t & R for one pair of frames
   - We repeat the steps 1,2 & 3 for 800 pairs of images & obtain individual R & t for each pair.
4. Scale the translation *tk* with the absolute or relative scale
   - As we're dealing with Monocular camera, the absolute translation is not known. So we calculate the absolute translation for each pair of image from the ground truth provided.
   - Then we multiply the absolute value with the T obtained for each pair of image frame to get absolute translation of camera w.r.t the image plane.

- Then by concatenating R & corrected t, we obtain T for each pair of image.

5. Concatenate the relative transformation by computing $Ck = Ck\text{-}1Tk$, where $Ck\text{-}1$ is the previouspose of the camera in the world frame.
   - Taking T_0 as a reference pose, we obtain
     $[C_0 , C_1 , C_2 , \ldots C_{n]} = [T_0 , T_0{*}T_1 , T_0{*}T_1{*}T_2 , \ldots T_{n-1}{*}T_n]$
   - We flattened C in the dimension of (801,12) from (801,3,4) and stored in the text file(In the same format as the ground truth was given
6. Then we plot the estimated trajectory along with the ground truth trajectory.

Code:

```
from os import listdir
from os.path import isfile, join
import numpy as np
import random
import matplotlib.pyplot as plt
import cv2
import matplotlib.image as mpimg
import math

#1)loading images

mypath='C:\\Users\\Asus\\Desktop\\images'
onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
```

```
images = np.empty(len(onlyfiles), dtype=object)
g_images=np.empty(len(onlyfiles), dtype=object)
for n in range(len(onlyfiles)):
    images[n] = cv2.imread( join(mypath,onlyfiles[n]) )
    g_images[n] = cv2.cvtColor(images[n], cv2.COLOR_BGR2GRAY)
#Loading Ground Truth
#extracting t and calculating norm of t for each pic according to ground truth for absolute translation
t=np.empty((801,3))
for i inrange(801):
    t[i][0]=GT[i][3]
    t[i][1]=GT[i][7]
    t[i][2]=GT[i][11]
print(t.shape)
t_abs=np.empty((1,801))    # calculating norm of each t in ground truth
for i inrange(801):
    t_abs[0][i]=np.linalg.norm(t[i])
print(t_abs.shape)
#taking 2 images and calculating the optical flow
defOptical_Flow(i0,i1):
        p0 = cv2.goodFeaturesToTrack(i0,500,.1,50)
        p1, status, _ = cv2.calcOpticalFlowPyrLK(i0,i1,p0, None)
        status = status.reshape(status.shape[0])
        p0 = p0[status ==1]
        p1 = p1[status ==1]
        p0 = np.ndarray.reshape(p0,(len(p0),2))
        p1 = np.ndarray.reshape(p1,(len(p0),2))
return(p0,p1)
#Function to normalise a 2d array column-wise
from decimal import Decimal
defNor_Col(M):
        n=M.shape     #size tuple
        k=n[-1]       #column dimension
        r=n[0]        #row dimension
        N=np.empty(k)
        F = np.empty((r,k))
for i inrange(k):
            N[i]   = np.linalg.norm(M[:,i])
for j inrange(k):
for i inrange(r):
                a=np.asscalar(M[i][j])
                b=np.asscalar(N[j])
                c= Decimal(a)/Decimal(b)
```

```
            F[i][j]=c


return(F)

#Estimating F from obtained correspondences under RANSAC
defF_Estimation(p0,p1):
#Taking 9 random correspondence at a time and estimating F for that: doing
this for 500 times and have 500 proposed F for 2 images
        cor0=np.zeros((1000,9,2))
        cor1=np.zeros((1000,9,2))
        M=np.ones((9,9))
        F=np.zeros((1000,3,3))
for k inrange(1000):
for i inrange(9):
                        j=random.randrange(len(p0))
                        cor0[k,i]=p0[j]                         #sampling random points
of p0 & p1 9 at a time
                        cor1[k,i]=p1[j]
#cor1[0]=x',cor1[1]=y';cor0[0]=x,cor0[1]=y
#M[i]=[x'x xy' x yx' yy' y x' y' 1]

M[i]=[cor1[k][i][0]*cor0[k][i][0],cor1[k][i][1]*cor0[k][i][0],cor0[k][i][0],cor1[
k][i][0]*cor0[k][i][1],cor1[k][i][1]*cor0[k][i][1],cor0[k][i][1],cor1[k][i][0],cor1
[k][i][1],1]
                M=Nor_Col(M)
                (U, W, V) = np.linalg.svd(M, full_matrices=True)
                W[-1]=0#Rank 2 constraint
                A = np.matmul(np.matmul(U,W),V)
                F[k]=A.reshape((3,3))



#Applying RANSAC method i.e. Calculating x'Fx for each F and finding which
x'Fx is smallest by comparing norms
        p0_h=np.ones((len(p0),3))
        p0_h[:,:-1]=p0
        p1_h=np.ones((len(p0),3))
        p1_h[:,:-1]=p1
        p0_h=p0_h.transpose()
        points1=np.zeros((9,2))
        points2=np.zeros((9,2))
        val=np.zeros((1000,len(p0)))
```

```
        F_mat=np.zeros((3,3))
        minN=3000
for k inrange(1000):
for b inrange(len(p0)):                #x'Fx=val for kth set for the Kth F matrix
                val[k,b]= np.matmul(np.matmul(p1_h[b],F[k]),p0_h[:,b])
          N= np.linalg.norm(val[k])                    #norm of the all distances
from proposed F
if N<minN :
                minN=N
                F_mat=F[k]
                points1=cor0[k]
                points2=cor1[k]
print(minN)
return(F_mat,points1,points2)
defPose_Calculator(F,points1,points2):
#Calculate E,R,t and pose
        R =np.empty((3,3))
        t =np.empty((3,1))
        K =
np.array([[7.215377000000e+02,0.000000000000e+00,6.095593000000e+02],

[0.000000000000e+00,7.215377000000e+02,1.728540000000e+02],

[0.000000000000e+00,0.000000000000e+00,1.000000000000e+00]])
        focal=np.array([7.215377000000e+02])
        pp=(6.095593000000e+02,1.728540000000e+02)
        k=K.transpose()
        k1=np.linalg.inv(K)
        k2=np.linalg.inv(k)
        E=np.matmul(np.matmul(k2,F),k1)
        cv2.recoverPose(E, points1, points2,R, t,focal,pp,mask=None)
        T=np.zeros((4,4))
        S=np.append(R,t,axis=1)
        T=np.append(S,[0,0,0,1])
        T=np.reshape(T,(4,4))
return(T)
# main program body
F=np.empty((3,3))
P1=np.empty((9,2))
P2=np.empty((9,2))
T=np.empty((801,4,4))
for i inrange(800):
        i0=g_images[i]
```

```
        i1=g_images[i+1]
        k0,k1 = Optical_Flow(i0,i1)         #Calculating optical flow
        F,P1,P2 = F_Estimation(k0,k1)       #Calculating F
        T[i]=Pose_Calculator(F,P1,P2)       #Calculating Pose Matrix
for j inrange(4):
            T[i,j,3] *= t_abs[0][i]         #Providing Absolute translation
        T[i,3,3] =1
#obtaining the text file for 801 poses
M=np.empty((801,4,4))
C=np.empty((801,12))
G=np.empty((801,3,4))
M[0]=T[0]
G[0]=np.delete(M[0],3,axis=0)
C[0]=G[0].flatten()
for i inrange(1,801):
        M[i] = np.matmul(M[i-1],T[i])
        G[i] = np.delete(M[i],3,axis=0)
        C[i] = G[i].flatten()
np.savetxt('C:\\Users\\Asus\\Desktop\\Pose.txt',C)
```

Where does algo works well, where it fails–

1. Algo might work well in the condition if the pictures taken are at very close instance of time in which there are more correspondence points between a pair of images, hence better optical flow can be calculated and better estimation of poses transformation can be done.

But in case vehicle is moving at a very fast pace, then the consecutive Images won't have many pairs of correspondences, hence the chances of the error in pose estimation is increased as there is very less overlap between consecutive frames.

2. The error in the computation of each pose, gets multiplied due to pose concatenation. That's why the obtained trajectory is the deviated version of the ground truth trajectory and this can be corrected if we apply Bundle Adjustments. This algo may fail drastically without it.

3. The algo might work well in those situations where the static scene has the dominance over the moving objects. i.e. there are more static objects togive us more points of correspondences otherwise the algo might fail.
4. Algo will work well, if there is enough texture to identify the apparent motion. i.e. if large section of the image has homogeneity, then identifying correct correspondence pair becomes a challenge.
5. The algo will work well in the conditions having illumination level above certain threshold, otherwise it'd fail.
6. The algo might also fail because of non-optimal choice in the no. of iterations of RANSAC algo.

Bonus: Other ways to compute the absolute scale

1) Multi sensor-based methods:

LIDAR has been widely employed with cameras for VO and SLAM. The absolute scale can be recovered directly based on the depth sensor. Though, the motion estimation-based LiDAR point cloud can suffer from motion distortion effects as the range measurements are received at various times during continuous LiDAR motion. A combination of Camera and LiDAR to obtain motion estimation is also commonly employed to enhance the benefit and neglect the disadvantages of each other.

IMU sensor can produce 3-D acceleration of rotation and translation of moving platform. The platform's position can be obtained by a second order integral. We can observe direct integration of acceleration of measurement drifts quadratically in time because of measurement noise. Although, the drift of camera-based VO is comparatively small. That's why camera and intelligent combination of IMU can provide not only scale but also stable estimation. Stereo Camera also plays an important role as a sensor for VO and SLAM as the baseline length is fixed and always known.

2) Camera height-based methods:

To obtain absolute scale estimation, we can use camera height methods on the assumption of flat ground plane. A good ground point should be at the centre bottom area of an image. It helps in presenting good image gradient along the epipolar line to enhance the matching the precision and is expected to be close to the estimated ground.

3) Scene knowledge-based methods:

This method is used when ground is not present in the image. The scale is estimated by minimizing the distance between detected object's size and previous scene knowledge.

4) Deep learning-based methods:

There are approaches to learn camera pose and scale from image sequences implicitly in Deep learning. The stereo sequences are used for training process to provide both spatial and temporal photometric wrap error. This error provides the constraints for the scene depth and camera motion to be estimated in the real-world scale. When the network is tested, it can find single view depth and two view odometry from a monocular sequence.

Contribution: ***Theory by Sourav & code by Apoorva & Report by both.***