Mobile Robotics Assignment 2

Team ID 24

Roll No. 2019702002 Nivedita Rufus

Roll No. 2019702008 Sravya Vardhani S

Q1) Epipolar lines and epipoles

Approach:

Two sets of locations of corresponding points of few features and fundamental matrix are given and our task is to find out the epipolar lines and the epipoles. We know that the first plane, its correspondence and the fundamental matrix are related by :

$$\begin{pmatrix} u & v & 1 \end{pmatrix} \mathbf{F} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = 0.$$

The equation of line could be expanded to as :

$$(F_{11}u_1+F_{12}v_1+F_{13})\,u+(F_{21}u_1+F_{22}v_1+F_{23})\,v+(F_{31}u_1+F_{32}v_1+F_{33}) = 0.$$

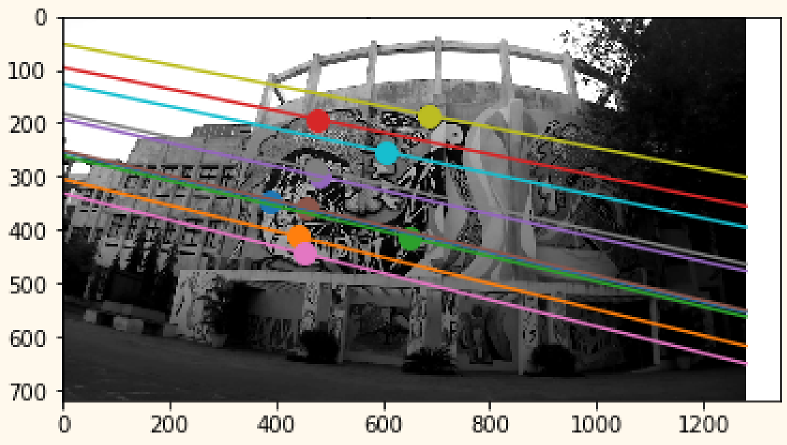
The code is based on this equation, first the coefficients of u and v are found out and then a line is drawn using them.

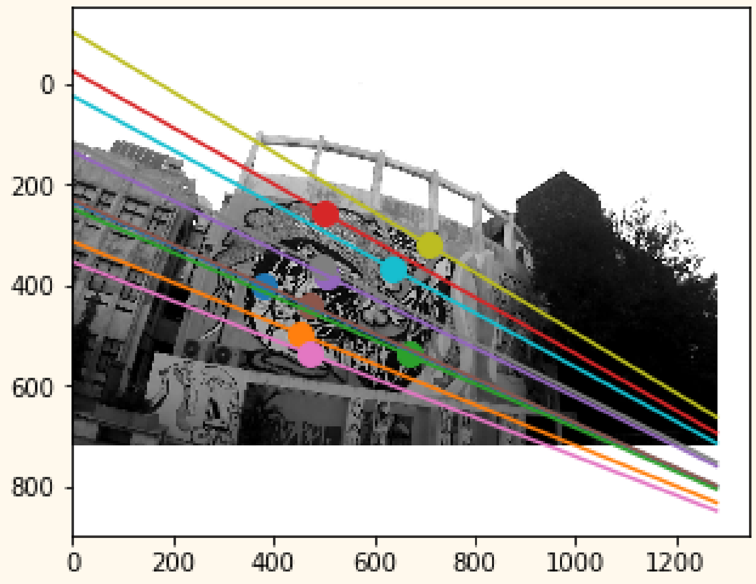
Code snippet :

```
lxd0 = np.matmul(Fmatrix,img1_points[0])
x0 = np.linspace(0,1280,num = 1280)
y0 = -lxd0[2]/lxd0[1] - x0*lxd0[0]/lxd0[1]
plt.imshow(img2,cmap = 'gray')
plt.plot(x0,y0)
plt.scatter(img2_points[0][0],img2_points[0][1],s=100)
```

```
#for the second image,
lx0 = np.matmul(img2_points[0].T,Fmatrix)
xd0 = np.linspace(0,1280,num = 1280)
yd0 = -lx0[2]/lx0[1] - x0*lx0[0]/lx0[1]
plt.imshow(img1,cmap = 'gray')
plt.plot(xd0,yd0)
plt.scatter(img1_points[0][0],img1_points[0][1],s=100)
```

Output :





Calculation of epipoles:

Approach:

We know that the epipolar points satisfy the equation:

$$F * e_L = 0$$

To solve this system of equations we use SVD and find the eigenvector corresponding to the smallest eigenvalue of F and normalise it to get the epipolar points.

$$[u,d] = \text{eigs}(F' * F)$$

Code snippet:

```
u, s, vh = np.linalg.svd(np.matmul(Fmatrix,Fmatrix.T))
print(u)
eigenval = u[:,2]
print(eigenval)
pt = eigenval / eigenval[2]
print(pt)
```

Output :

```
e1  = (2.15916627e+03,  1.18926809e+03)
e2  = (-5.13190979e+03,  -9.48854662e+02 )
```

Q2) Feature-based Visual Odometry

Approach:

The task is to implement a basic monocular visual odometry algorithm for a sequence of images from the KITTI dataset and its ground truth poses.

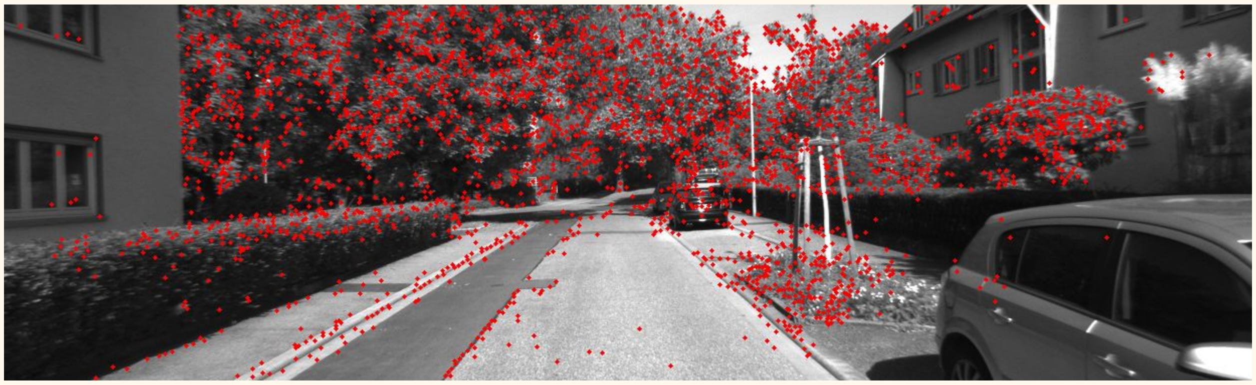
Steps :

1. The first step is to find the coordinates of key features of the images. This is done using the SIFT (Scale-invariant Feature transform)algorithm.

Code snippet:

```
def extract_sift_keypoints(im):
    sift = cv2.xfeatures2d.SIFT_create()
    key_points = sift.detect(im)
    pts = np.array([x.pt for x in key_points],dtype=np.float32)
    return pts
```

Output :

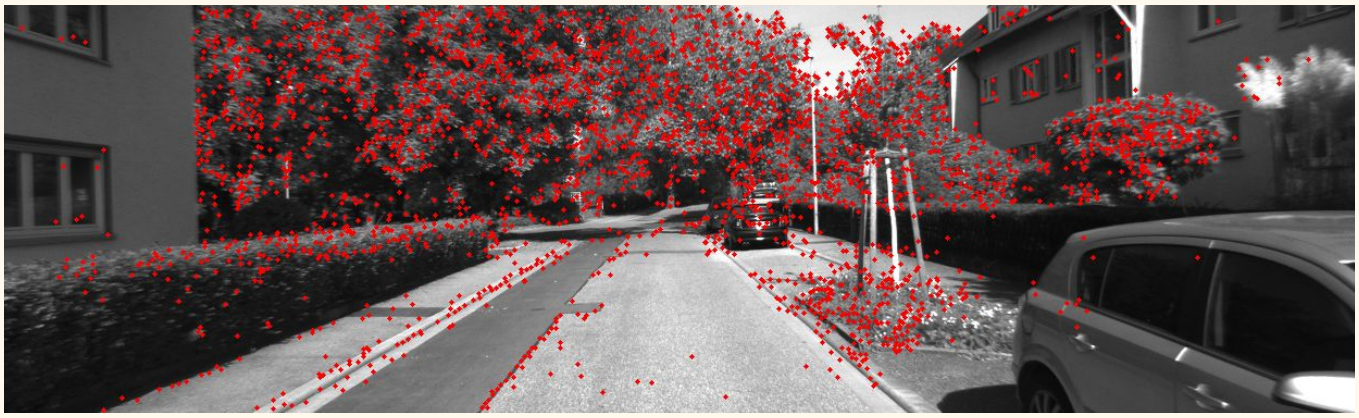


2. To track the key-points in the consecutive image, features of the previous image is matched with corresponding features of the current image. This is done by computing the 'Optical Flow'.

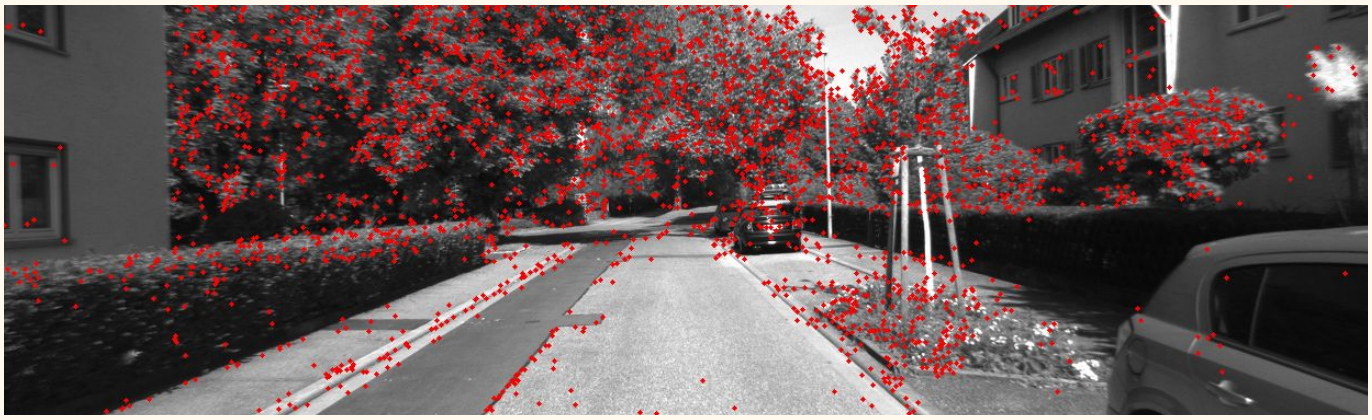
Code snippet:

```
def featureTracking(img_1, img_2, p1):
    p2, st, err = cv2.calcOpticalFlowPyrLK(img_1, img_2, p1, None)
    st = st.reshape(st.shape[0])
    p1 = p1[st==1]
    p2 = p2[st==1]
    return p1,p2
```

Output :
Previous image:



Current image:



3. From the obtain correspondences, we determine the fundamental matrix 'F' using the 8-point algorithm:

$$\mathbf{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = 0.$$

Where, **(x'_1, y'_1) , (x_2, y_2)** are correspondences from the current and the previous image respectively. NOTE: a minimum of 8 correspondences are required.

Code snippet :

```
def compute_fundamental(x1,x2):
    n = x1.shape[1]
    A = zeros((n,9))
    for i in range(n):
        A[i] = [x1[0,i]*x2[0,i], x1[0,i]*x2[1,i], x1[0,i]*x2[2,i],
                x1[1,i]*x2[0,i], x1[1,i]*x2[1,i], x1[1,i]*x2[2,i],
                x1[2,i]*x2[0,i], x1[2,i]*x2[1,i], x1[2,i]*x2[2,i] ]
    U,S,V = linalg.svd(A)
    F = V[-1].reshape(3,3)
    U,S,V = linalg.svd(F)
    S[2] = 0
    F = dot(U,dot(diag(S),V))
    return F/F[2,2]
```

But the 'F' obtained using only the 8-point algorithm is highly erroneous because the presence of outliers. To account we iteratively compute 'F' using the RANSAC scheme.

Code snippet :

```
def F_from_ransac(x1,x2,model,maxiter=200,match_theshold=1e-6):
    import ransac
    data = vstack((x1,x2))
    F,ransac_data =
ransac.ransac(data.T,model,8,maxiter,match_theshold,20,return_all=True)
    return F, ransac_data['inliers']
```

4. The next step is to obtain the essential matrix. This is calculated from the fundamental matrix using:

$$\mathbf{E = K^T(FK)}$$

Where, **K** is the camera Matrix and **F** the fundamental matrix.

5. The next step is to decompose this essential matrix to obtain the relative rotation R_k and translation t_k , and form the transformation T_k. But the translation needs to be scaled. For this we obtain the absolute scale from the ground truth data provided.

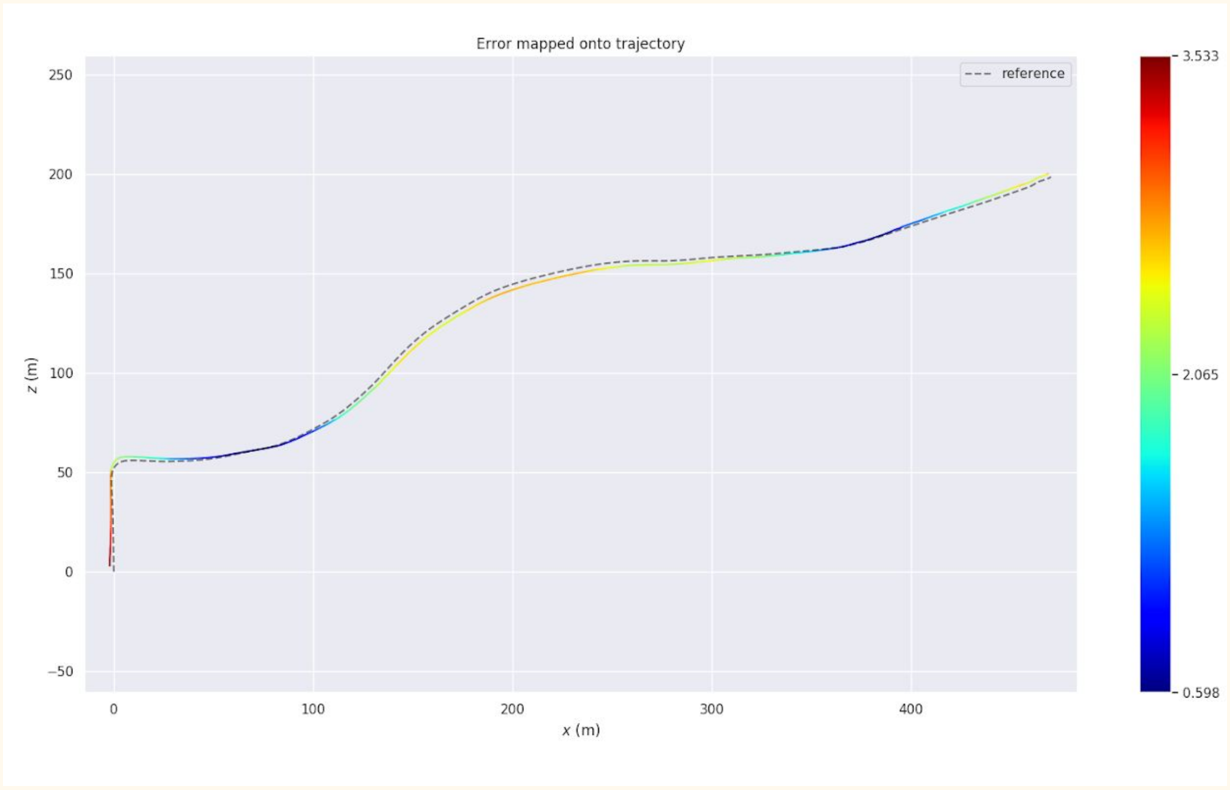
Code snippet :

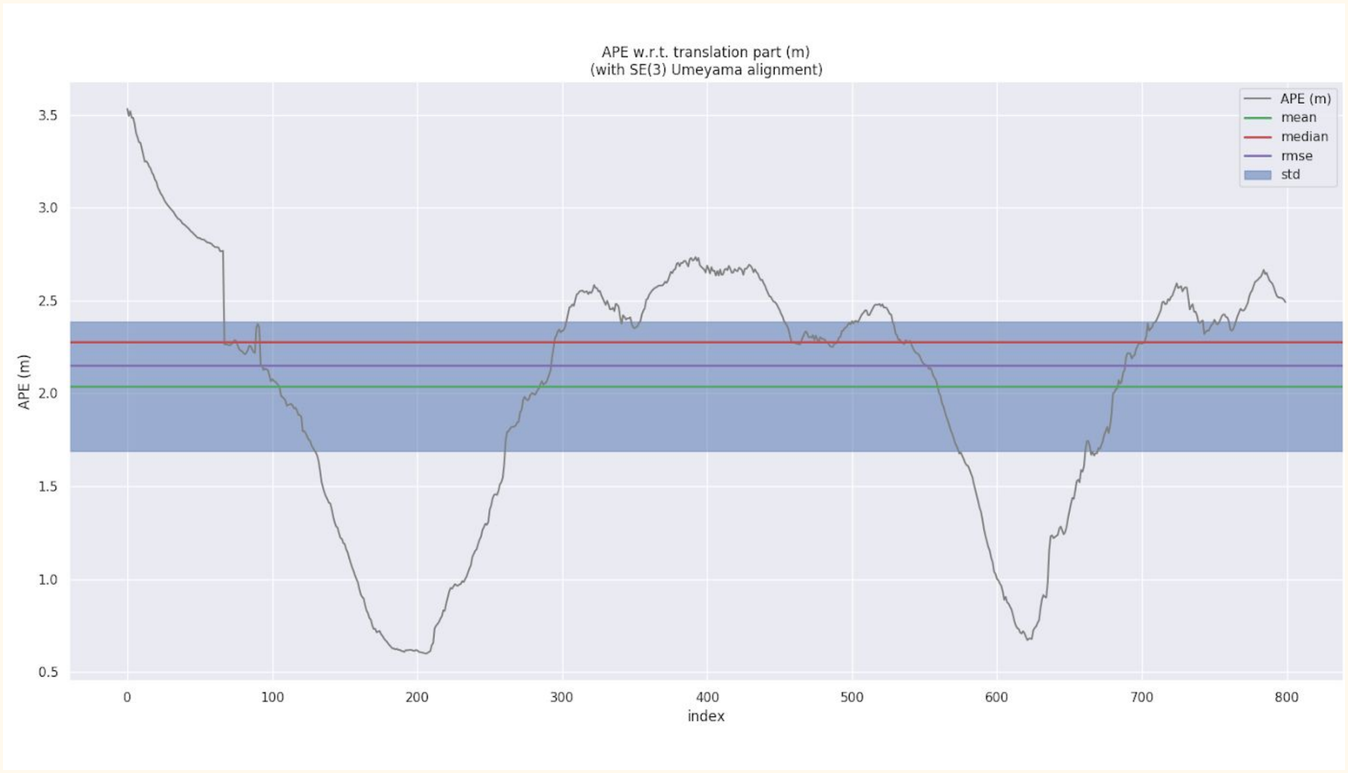
```
def Scale(f, frame_id):
      x_pre, y_pre, z_pre = f[frame_id-1][3], f[frame_id-1][7], 
f[frame_id-1][11]
      x, y, z = f[frame_id][3], f[frame_id][7], f[frame_id][11]
      scale = np.sqrt((x-x_pre)**2 + (y-y_pre)**2 + (z-z_pre)**2)
      return scale

if scale > 0.1:
        t_f = t_f + scale*R_f.dot(t)
        R_f = R.dot(R_f)
```

6. The final step is to concatenate all the relative transformation and store them. This procedure should be repeated for the next pair of images.

Output :





Comments on the obtained output:

- It can be observed that obtained trajectory is not very accurate but error can be observed at certain points; for example, at the start of the trajectory we observe maximum error.
- It can also be inferred from the plot that the error is never zero, which means there is a minimum error of 0.6 metres.
- The reason for having this erroneous output maybe because the ground truth data given to us maybe noisy. This will lead to wrong absolue scale calculation.
- The other reason for the occurrence of error maybe because the camera matrix may calibrated with some noise. This will result in errors while estimating the essential matrix.

BONUS Question:

Methods to calculate absolute scale:

- To know the scale factor, we can use additional encoders such as optical encoders incorporated with a specific motion model. Using this we will be able to estimate the difference in camera position between a pair of images which effectively becomes our scale factor.
- Another way is to have a depth sensor (eg: kinect) which can reliably calculate depth if there are objects indeed present at that depth.
- Incorporation of the stereo camera is also another way to estimate the depth from the disparity ratio. This can be used to obtain the scale.

Roles :

Nivedita Rufus :

Q1: estimation of epilines

Q2: feature tracking, computation of F using Ransac, scale estimation

Sravya Vardhani S:

Q1: estimation of Epipoles

Q2: feature and keypoint detection, final concatenation of transformations, plotting results

NOTE: We had discussed everything together before execution. The bonus question was discussed and done together as well.