Fall 2021 Robot Perception HW2 Report

Aditya Wagh (amw9425)

Task 1

The focus of task 1 is to fit a 3d plane inside a point cloud data using RANSAC (Random Sample Consensus).

My algorithm to do so is as follows:

- Declare variables to store best plane parameters and best inliers.
- At each iteration, randomly select three points.
- Construct a plane using those three points. The parameters can be computed as follows. For a plane **ax+by+cx+d**
    - a = (y2 – y1) * (z3 - z1) – (z2 – z1) * (y3 – y1)
    - b = (z2 – z1) * (x3 - x1) – (x2 – x1) * (z3 – z1)
    - c = (x2 – x1) * (y3 - y1) – (y2 – y1) * (x3 – x1)
    - d = - (a * x1 + b * y1 + b* z1)
- Then we compute the distance of each point from the plane as follows
    - D = aX + bY + cZ / || plane normal||
    - Where || plane normal|| represents magnitude of plane normal.
- Then we calculate the best inliers by comparing their distance with a set threshold of 0.05 and store them in a list.
    - If at current iteration, if number of best inliers < number of current inliers
        - Then current inliers = best inliers, and current plant parameters are best plane parameters.

I got an inlier ratio of about 88-89 % with the plane that I was able to fit. I have provided a jupyter notebook to visualize the result. This makes sense since most of the data lies on a plane.

Plane Parameters :

[-0.01663011817554566, -0.29167456155922655, -0.22266615912497523, 0.1901735211847007]

Inlier ratio :

0.8902615236981724

Task 2

The focus of task 2 is to calculate the

- F Matrix using point correspondences
- Draw corresponding epipolar lines
- Find relative pose between images

I have used pyAprilTag.find to find out 40-point correspondences from the images, which are the 40-point correspondences in the image. Why 40-points? There are 40 april tags in those images. It outputs the positions of corners, centers, and the id and homography of each image. My point correspondences are centers of the April tags.

I stored the images into point correspondences into a dictionary since the order of points did not match, the I sorted the dictionary using key value pairs. Following which I computed the F matrix using cv2.findFundamentalMat() function in open cv2.

The value of F I got is –

```
[[-4.50756704e-06   -5.47177607e-05    3.26190789e-02]
[ 4.95879263e-05     -9.38841117e-06    7.19771172e-03]
[-3.38434320e-02     -3.46699368e-03   1.00000000e+00]]
```

For the 2nd part of the question, the epipolar lines will not be accurate since the image has radial distortion.

For the third part of the question, we cannot calculate the pose using F matrix since we do not have the camera calibration matrix. We need to compute the essential matrix using fundamental matrix and the camera calibration matrix. Following which, we can decompose the essential matrix to get the R and t vectors between the images.

We can however get K from the pyapriltag package as instructed by prof Chen. And then we can compute the R, t and essential matrix from F and K.

$$E = K' * F * K$$

K ' is transpose of K.

The values of R I got are

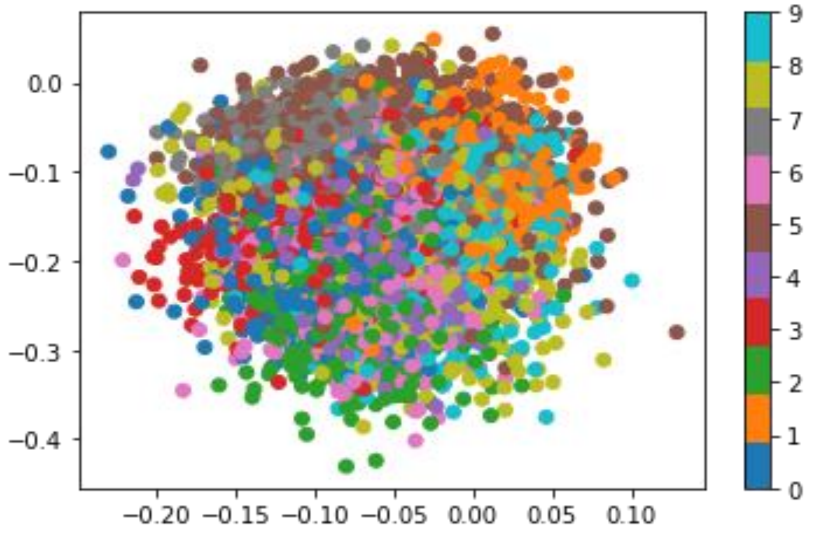
```
[ 0.99039797 -0.10391459            -0.091179]
[ 0.11315053          0.9882484           0.10277184]
[ 0.07942801  -0.11210198                 0.99051721]

[-0.49080152         -0.38874562         -0.77973759]
[-0.52842439         -0.57874008          0.62115021]
[-0.69273482          0.71689383          0.07862384]
```

The value of t I got is

```
[[-0.55919728]
[ 0.46481517]
[ 0.68647305]]
```

Task 3

For Q3, I have used an autoencoder to reduce the dimensions of the images to 16. The following is my plot for latent features when the latent space dimensions are 16. The colors are the 10 classes of Fashion MNIST.



The architecture of my Model is as follows

A – Encoder

```
class Encoder(nn.Module):
def __init__(self, latent_dims):
super(Encoder, self).__init__()
```

```
self.linear1 = nn.Linear(784, 512)
self.linear2 = nn.Linear(512, latent_dims)


def forward(self, x):
x = torch.flatten(x, start_dim=1)
x = F.relu(self.linear1(x))
return self.linear2(x)
```

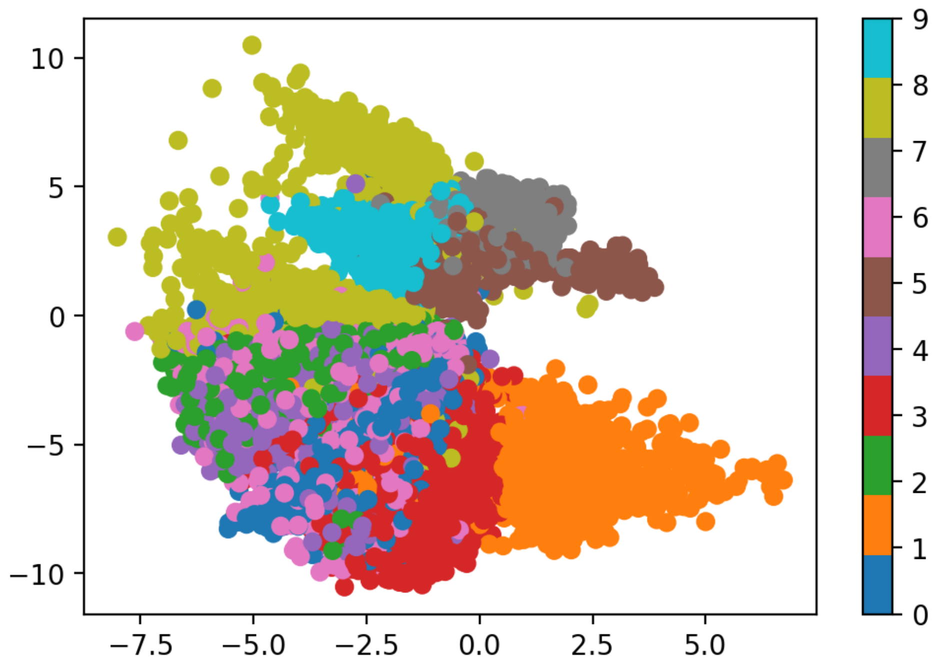
B – Decoder

```
class Decoder(nn.Module):
def __init__(self, latent_dims):
super(Decoder, self).__init__()
self.linear1 = nn.Linear(latent_dims, 512)
self.linear2 = nn.Linear(512, 784)


def forward(self, z):
z = F.relu(self.linear1(z))
z = torch.sigmoid(self.linear2(z))
return z.reshape((-1, 1, 28, 28))
```

Main Model

```
class Autoencoder(nn.Module):
def __init__(self, latent_dims):
super(Autoencoder, self).__init__()
self.encoder = Encoder(latent_dims)
self.decoder = Decoder(latent_dims)


def forward(self, x):
z = self.encoder(x)
return self.decoder(z)
```

I am using a mean square loss to calculate the error in predictions. I have used Adam as the optimizer with default parameters. The batch size that I am using is 128. So for training images, there would be around 489 latent feature maps.



I have also directly plotted the latent space in 2 dimensions. The following is a plot for that feature.