[Fall 2022] ROB-GY 6203 Robot Perception Homework 2

Rushi Bhavesh Shah

Submission Deadline (No late submission): NYC Time 11:00 AM, November 16, 2022
Submission URL (must use your NYU account): https://forms.gle/mAd3x65xi1rLwK919

1. Please submit the **.pdf** generated by this LaTex file. This .pdf file will be the main document for us to grade your homework. If you wrote any code, please zip all the **code** together and **submit a single .zip file**. Name the code scripts clearly or/and make explicit reference in your written answers. Do NOT submit very large data files along with your code!

2. Please **start early**. Some of the problems in this homework can be **time-consuming**, in terms of the time to solve the problem conceptually and the time to actually compute the results. *It's guaranteed that you will NOT be able to compute all the results if you start on the date of deadline.*

3. Please typeset your report in LaTex/Overleaf. Learn how to use LaTex/Overleaf before HW deadline, it is easy because we have created this template for you! **Do NOT submit a hand-written report!** If you do, it will be rejected from grading.

4. Do not forget to update the variables "yourName" and "yourNetID".

Contents

Task 1. RANSAC Plane Fitting (4pt)

In this task, you are supposed to fit a plane in a 3D point cloud. You have to write a custom function to implement the RAndom SAmple Consensus (RANSAC) algorithm to achieve this goal.

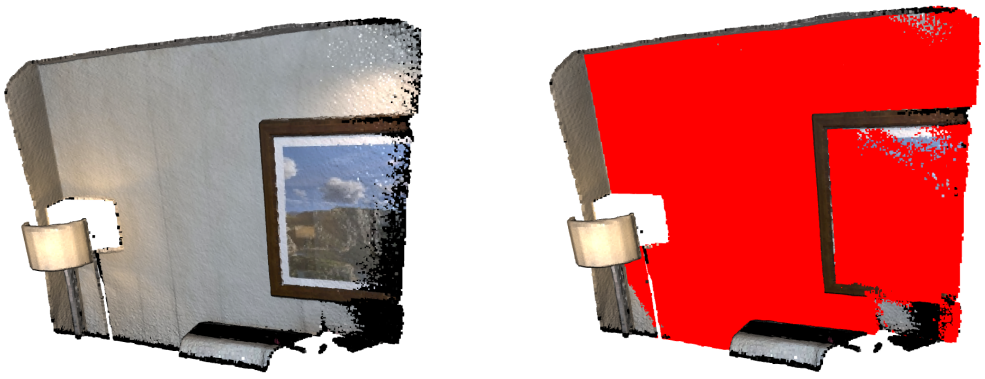

Figure 1: **Left:** Original Data, **Right:** Data with the best fit plane marked in red.

Use the following code snippet to load and visualize the demo point cloud provided by Open3D.

```
import open3d as o3d

# read demo point cloud provided by Open3D
pcd_point_cloud = o3d.data.PCDPointCloud()
pcd = o3d.io.read_point_cloud(pcd_point_cloud.path)

# function to visualize the point cloud
o3d.visualization.draw_geometries([pcd],
                                  zoom=1,
                                  front=[0.4257, -0.2125, -0.8795],
                                  lookat=[2.6172, 2.0475, 1.532],
                                  up=[-0.0694, -0.9768, 0.2024])
```

Note: If you use RANSAC API in existing libraries instead of your own implementation, you will only get 60% of the total score.

Answers:

RANSAC stands for RANdom Sampling and Consensus. It is a robust model fitting algorithm that is frequently compared to the Linear Regression algorithm. This particular task demands to fit a plane in a 3D point cloud for the dataset provided in the Open3D library.

Initially, a function is defined to generate and return the coefficients for a plane equation (i.e $ax + by + cz + d = 0$), given three randomly generated points. This is done by getting the intersecting vectors passing through all three points and cross multiplying them to get the vector perpendicular to both the vectors. So the three components of this vector are respectively 'a', 'b' and 'c'. Further, to get 'd', I get the dot product of this vector with either of the randomly generated points.

Further, to get the inliers and the no. of inliers, a function named 'inliers' is defined whichh return a list called *inlier_points*. It looks for the points from the dataset that lie within the threshold distance (0.02 in this case) and adds to the list. Further a function called *ransac* is defined that takes no. of iterations as input and contains two nested for loops. Outer loop generates random three points and the inner one defines the plane and finds the inliers. If the no. of inliers is highest till that point, then it stores them to a list and in the end return final inliers which are plotted alongside the original data.

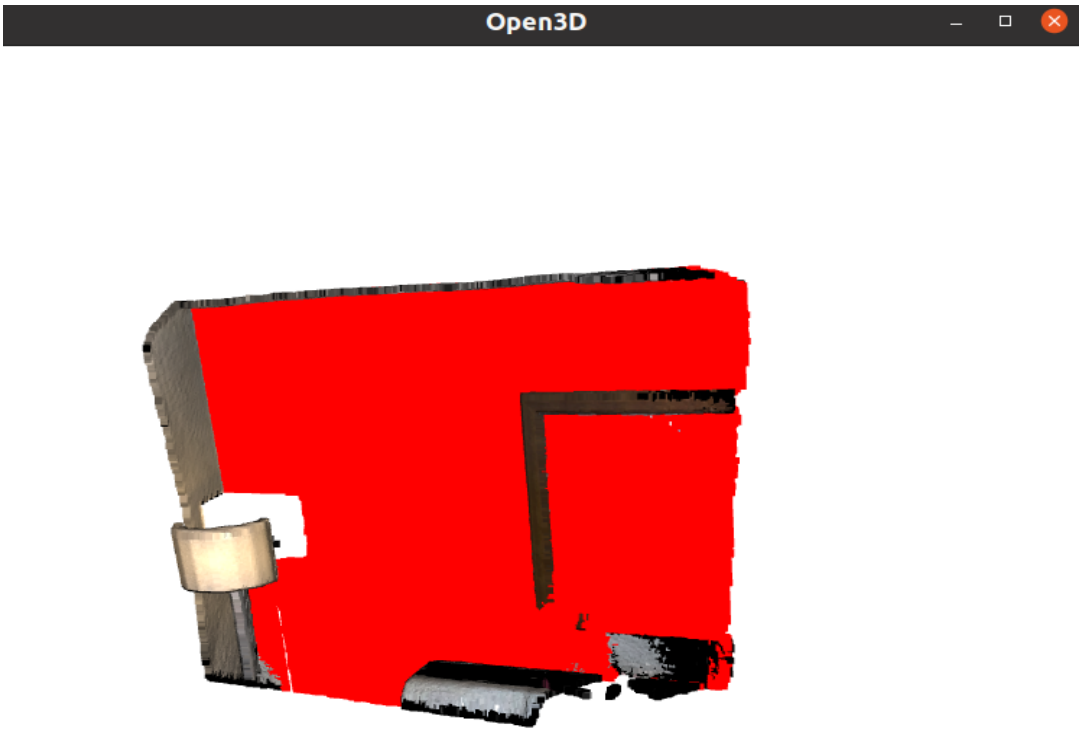

Figure 2: RANSAC Result (after 1000 iterations)

Task 2. ICP (4pt)

In this task, you are required to align two point clouds (source and target) using the Iterative Closest Point (ICP) algorithm discussed in class. The task consists of two parts.

a) (2pt)

In part 1, you have to load the demo point clouds provided by Open3D and align them using ICP. **Caution:** These point clouds are different from the point cloud used in the previous question. You are expected to **write a custom function to implement the ICP algorithm**. Use the following code snippet to load the demo point clouds and to visualize the registration results. You will need to pass the final 4X4 homogeneous transformation (pose) matrix obtained after the ICP refinement. Explain in detail, the process you followed to perform the ICP refinement.

```
import open3d as o3d
import copy

demo_icp_pcds = o3d.data.DemoICPPointClouds()
source = o3d.io.read_point_cloud(demo_icp_pcds.paths[0])
target = o3d.io.read_point_cloud(demo_icp_pcds.paths[1])

# Write your code here

def draw_registration_result(source, target, transformation):
    """
    param: source - source point cloud
    param: target - target point cloud
    param: transformation - 4 X 4 homogeneous transformation matrix
    """
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([1, 0.706, 0])
    target_temp.paint_uniform_color([0, 0.651, 0.929])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp],
                                      zoom=0.4459,
                                      front=[0.9288, -0.2951, -0.2242],
                                      lookat=[1.6784, 2.0612, 1.4451],
                                      up=[-0.3402, -0.9189, -0.1996])
```

Answers:
The ICP algorithm aims at finding the transformation between a point cloud and some reference surface (or another point cloud), by minimizing the square errors between the corresponding entities. The 'iterative' of ICP comes from the fact that the correspondences are reconsidered as the solution comes closer to the error local minimum. For corresponding data sets and performing ICP, my algorithm also employs an SVD-based least square best fit method.

To begin, the best fit transform is determined by first calculating the centroid of the point clouds for a pair of corresponding points, and then all points are projected to the center by calculating the difference between their coordinates and the centroid coordinates. To find the Rotation matrix, Singular Value Decomposition (SVD) is applied to the matrix formed by combining the above data. Further, rotation and translation matrices can be found using the $\boldsymbol{U}$ and $\boldsymbol{V\_transpose}$ matrices obtained after performing SVD. The equations used are $R = V\_transpose^T * U^T$ and $t = -R * A\_centroid + B\_centroid$. Further, ICP refinement is done using Scikit-learn's Nearest Neighbour to find the nearest (Euclidean) neighbor in target data for each data point from the source data. Once the best-fit transform is found using the ICP method for mapping points from the current source to the target, a transformation is calculated. Once the error is below a predetermined threshold, the final transformation is calculated and passed on as transformation. The current source is updated by verifying error repeatedly over time. The final transformation

matrix found is :

$$\begin{pmatrix} 0.99999092 & -0.00188073 & -0.003824 & -0.00243311 \\ 0.00198401 & 0.99962844 & 0.02718542 & -0.09554388 \\ 0.00377145 & -0.02719276 & 0.99962309 & 0.05347158 \\ 0. & 0. & 0. & 1. \end{pmatrix}$$

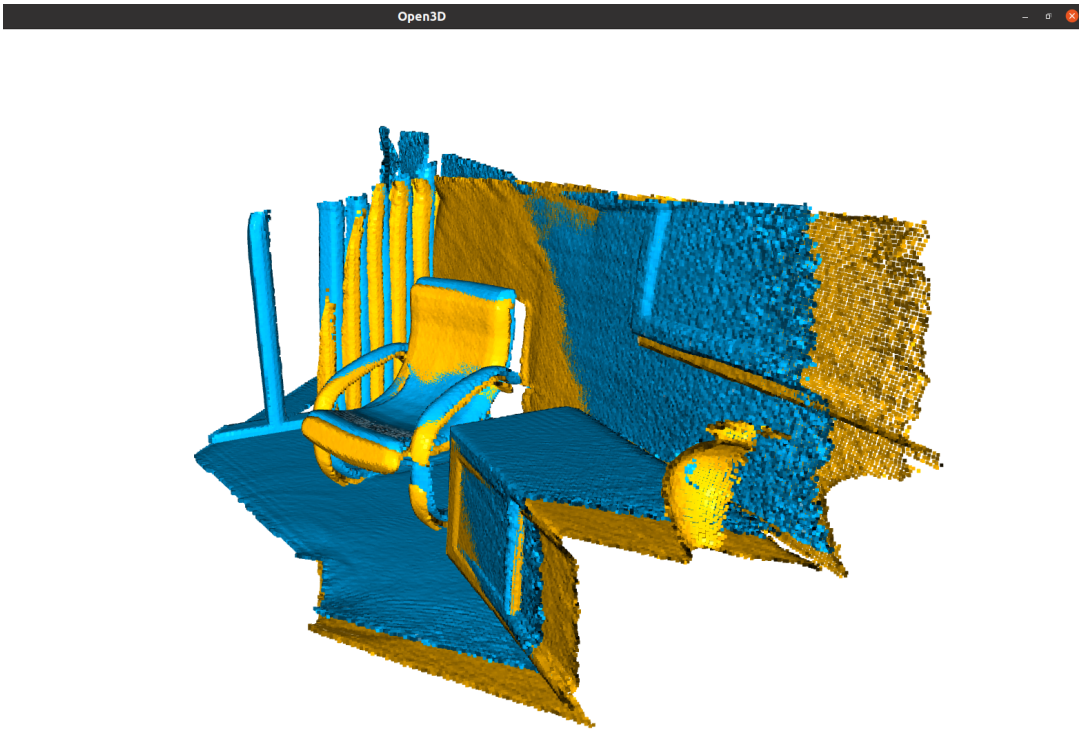

Figure 3: ICP

b) (2pt)

You have been given two point clouds from the KITTI dataset, one of the benchmark datasets used in the self-driving domain. *The point clouds are located in the data/Task2 folder under the overleaf project:* https://www.overleaf.com/read/fzhnnqsqrnbz. Repeat part 1 using these two point clouds. Compare the results from part 1 with the results from part 2. Are the point clouds in part 2 aligning properly? If no, explain why. Provide the visualizations for both parts in your answer.

Note: If you use an ICP API in existing libraries instead of your own implementation, you will only get 60% of the total score.

Answers:

In contrast to previous task, two pointclouds for the KITTI dataset are not aligning. This is because the ICP algorithm tries to fit the ground plane, which is why the two pointclouds' alignment is off. The two pointclouds will line up if the ground plane is taken out of the equation. It is also advised to remove the ground plane in the such cases because it has numerous inliers, which subject to noise, and can result in improper fitting of two the pointclouds.

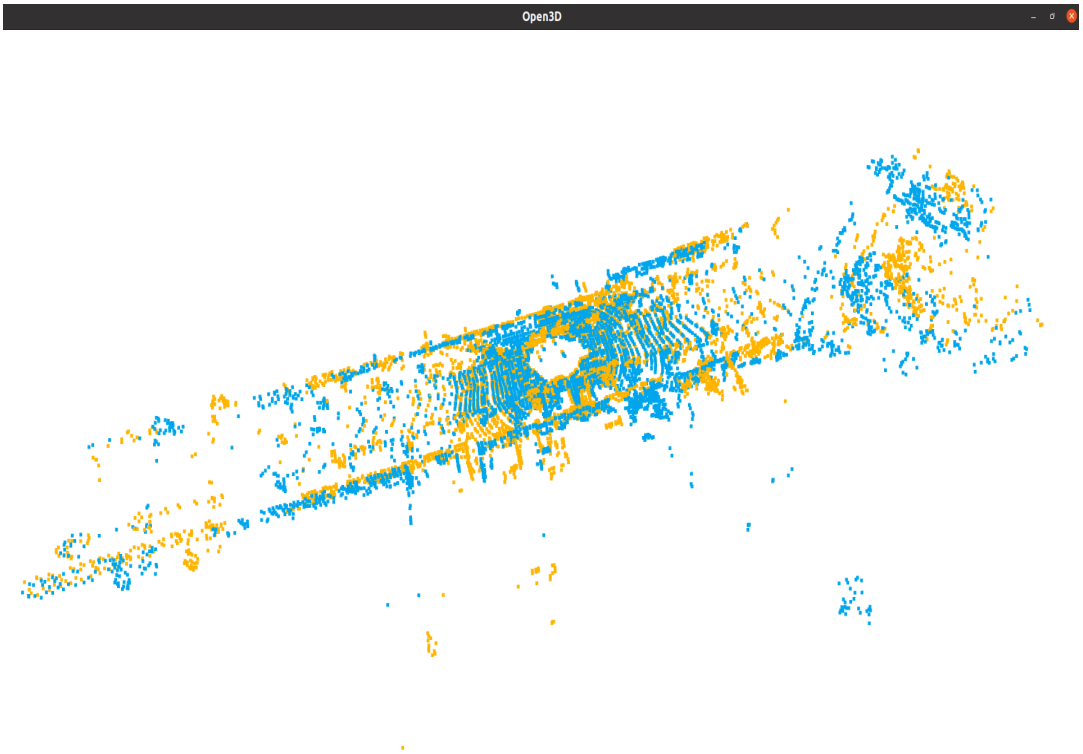

Figure 4: ICP on KITTI dataset

Task 3. F-matrix and Relative Pose (4pt)

All the raw pictures needed for this problem are provided in the `data/Task3` folder under the overleaf project: https://www.overleaf.com/read/fzhnnqsqrnbz. You may or may not need to use all of them in your problem solving process.

a) (2pt)

Estimate the fundamental matrix between `left.jpg` and `right.jpg`.

Tips: The Aruco tags are generated using Aruco's 6×6 dictionary. Although you don't have to use these tags.

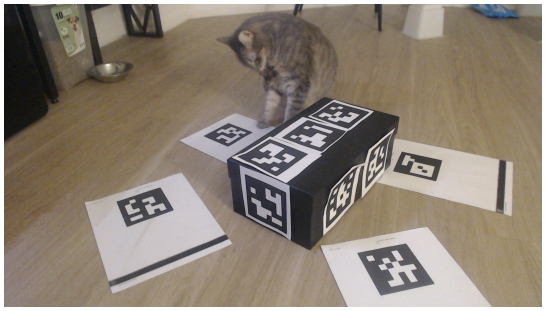

Figure 5: left.jpg

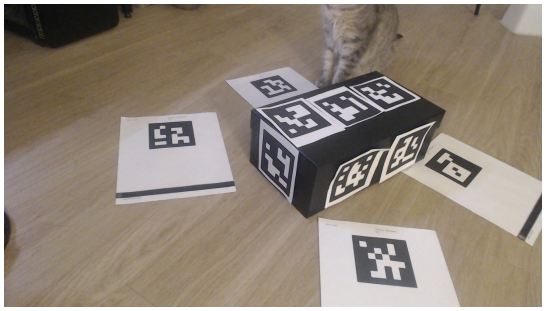

Figure 6: right.jpg

Answers:

First of all the camera is calibrated using the images provided in the dataset(even though it's not required for estimating the Fundamental Matrix). Further, SIFT feature extractor is used to determine the keypoints and the descriptors. Next, FLANN which is a library for performing fast approximate nearest neighbor searches in high dimensional spaces is used to set the paramaters and then perform a ratio test based on Lowe's algorithm to select suitable inliers. Further, 8-point algorithm is implemented to obtain a ***Fundamental*** matrix which is as follows :

$$\begin{pmatrix} 8.80667619e-072.28526765e-06-3.47322389e-03 \\ -2.93907073e-062.31552064e-064.52777565e-03 \\ 8.12868481e-04-6.95886539e-033.23672032e+00 \end{pmatrix}$$

b) (1pt)

Draw epipolar lines in both images. You don't need to explain the process. Just provide the visualization.

Answers:

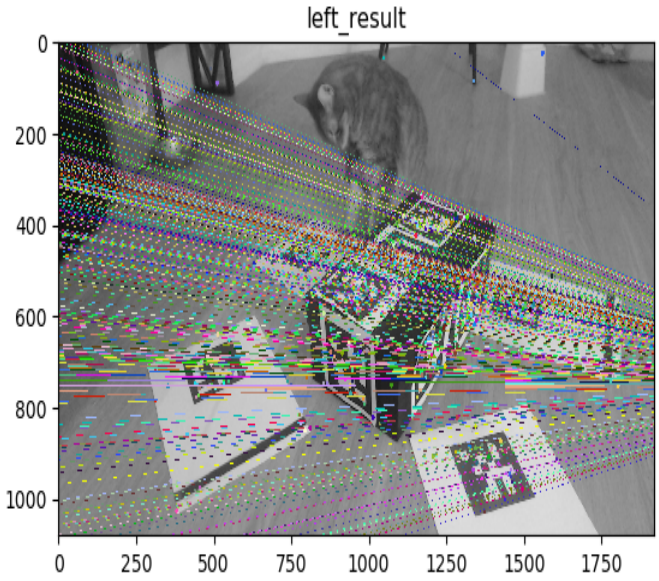

Figure 7: **Left:** Epipolar lines in Left Image

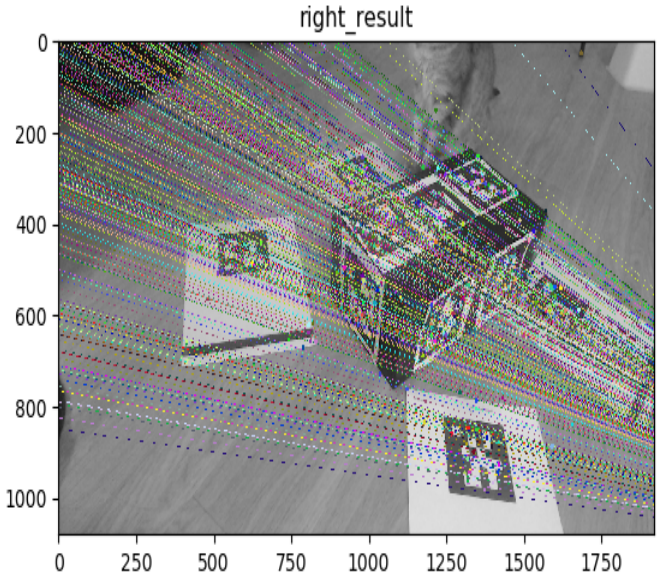

Figure 8: **Right:** Epipolar lines in right image

c) (1pt)

Find the relative pose (R and t) between the two images, expressed in the left image's frame. Before you give the solution, answer the following two questions

1. Can you directly use the F-matrix you estimated in a) to acquire R and t without calculating any other quantity?

2. If yes, please describe the process. If no, what other quantity/matrix do you need to calculate to solve this problem?

Answers:
1. No, as in contrast to essential matrix, which deals with calibrated cameras, fundamental matrix deals with uncalibrated cameras and is defined to a specific scale. The epipolar geometry is the intrinsic projective geometry between two perspectives, hence it is not possible to rebuild a 3D scene using only the Fundamental Matrix's features. It is unaffected by scene structure and only depends on the internal settings and relative posture of the cameras.

2. Other quantity that was needed to be calculated was the Essential Matrix. The camera calibration matrix was initially calculated using OpenCV methods utilizing the provided Chessboard Images. After calculating the camera calibration matrix, the fundamental matrix and camera calibration matrices were utilized to solve an equation to determine the essential matrix. Additionally, the rotation and translation of the second image relative to the first were obtained by decomposing the fundamental matrix using SVD. Further, by using U and V_transpose from SVD, two rotation matrices and one translation matrices are obtained.

Equation to calculate Essential Matrix : K^T*F*K (where K is the camera matrix and F is the fundamental matrix)

Equation to calculate the rotation vector : U.($\omega.V^T$)($whereU.S.V^T = SVD(EsentialMatrix)$)

$$E = \begin{bmatrix} 1.66993767 & 4.34771825 & -1.72821367 \\ -5.59157764 & 4.41986564 & 4.1229085 \\ -0.06581085 & -4.60483438 & 0.45679842 \end{bmatrix}$$

$$R1 = \begin{bmatrix} -0.12860971 & 0.48342448 & 0.86588701 \\ -0.1887394 & -0.86911092 & 0.45719103 \\ 0.9735692 & -0.10462779 & 0.20301735 \end{bmatrix}$$

$$R2 = \begin{bmatrix} 0.91127052 & -0.39835951 & 0.10438264 \\ 0.41127166 & 0.89329726 & -0.18131635 \\ -0.02101563 & 0.20815787 & 0.97786944 \end{bmatrix}$$

$$t = \begin{bmatrix} 0.62474339 \\ 0.17763194 \\ 0.76035689 \end{bmatrix}$$

Task 4. Low Dimensional Projection (4pt)

Given the Fasion-MNIST dataset, **train** an unsupervised learning neural network that gives you a lower-dimensional representation of the images, after which you could easily use tSNE from `Scikit-Learn` to bring the dimension down to 2. **Visualize** the results of all 10000 images in one single visualization.

Answers:

Before proceeding to for the solution to this task, I imported ***keras*** and then loaded Fashion-MNIST dataset from keras. The training and testing data is loaded in separate containers along with their labels. Before we pass the data through the Neural Network the data needs to be pre-processed. So first both the training and the test set needs to be normalized. Further, we need to flatten the data i.e. forming a single column array by stacking all the rows below one another. Further, we'll be using an Autoncoder in *Tensorflow* wherein an input vector or a feature vector will be the input , which will be reduced or "encoded" to a particular dimension and then again "decoded" to reconstruct the original data.

The dimension in the layer in which the dimension of dataset is reduced, is declared as the latent_dim=16. In my solution, I just have one encoded_layer and one decoded_layer and one co-vector. We also use the "dense layer" since we are using Artificial Neural Networks and use "ReLU" as the activation function. And then we'll have a decoded layer, where we'll pass the encoded layer reconstruct the data back to original dimension to create a sequential model.

Further, we'll make a combined model called autoencoder() and also encoded_input() along with decoder(). Further we compile the attoencoder() using "adam" optimizer. Then we fit the data for 10 epochs. Further we'll predict the images using ". predict" function

Further all the 10000 encoded test images are passed through tSNE for visualization by bringing the dataset down to 2 dimensions.

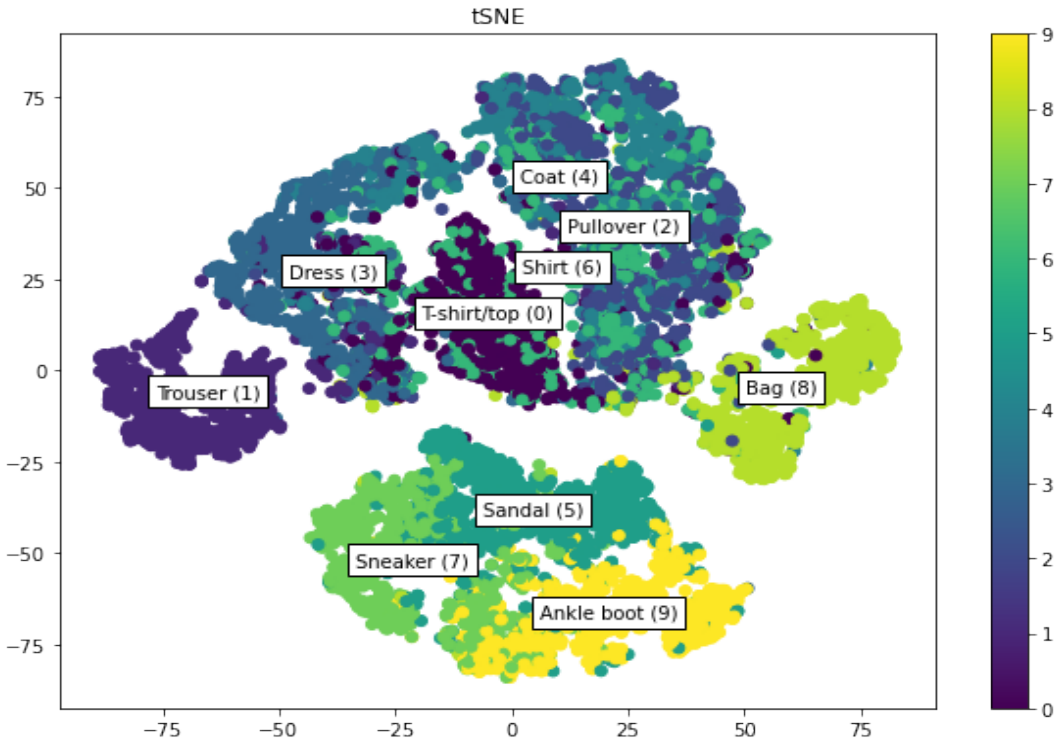

Figure 9: **D**ata visualization using tSNE

Task 5. Skiptrace (4pt)

Sherlock needs a team to defeat Professor Moriarty. Irene Adler recommended 3 reliable associates and provided 3 pictures of their last known whereabouts. Sherlock just needs to know their identities to be able to track them down.

Sherlock has a database of surveillance photos around NYC. He knows that these three associates definitely appear in these surveillance photos once.

Could you use these 3 query pictures provided by Irene Adler to figure out the names of pictures that contain our persons of interest? After you **obtain the picture names**, **show these pictures** to us in your report, and comment on the possibility of them defeating Professor Moriarty!

Tip 1: This question can be time consuming and memory intensive. To give you some perspective of what to expect, I tested it on my laptop with 16GB of RAM and a Ryzen 9 5900HS CPU (roughly equivalent to a Intel Core i7-11370H or i7-11375H) and it took about 50 mins to finish the whole thing, so please start early.

Tip 2: Refrain from using `np.vstack()/np.hstack()/np.concatenate()` too often. Numpy array is designed in a way so that frequently resizing them will be very time/memory consuming. Consider other options in Python should such a need to concatenate arrays arise.

Answers:

The following process was used to calculate the classification problem utilizing a set of query images. In order to compute keypoints and descriptors and to achieve to feature extraction, the ORB feature extractor was used. Using the feature extraction function, keypoint_data and descriptor_data were stored in an array. Vertically stacked SIFT features are further fed to K means clustering by defining K-means model to compute clusters. Further, histograms are created for all the images in the dataset as well as the query, and then stored in an array of visual words by indexing properly. Used the Nearest Neighbor library from scikit-learn to obtain the indexes of the trained dataset images to that of the query images. The closest index for each query image is then found by fitting the model. Following the discovery of the closest index, Matplotlib is used to display images related to each query image.

The figures that contain our persons of interests for queries 1, 2 and 3 respectively are :

[1157-7368.jpg 1109-395.jpg 1157-7188.jpg]

(Figures are on the next page)

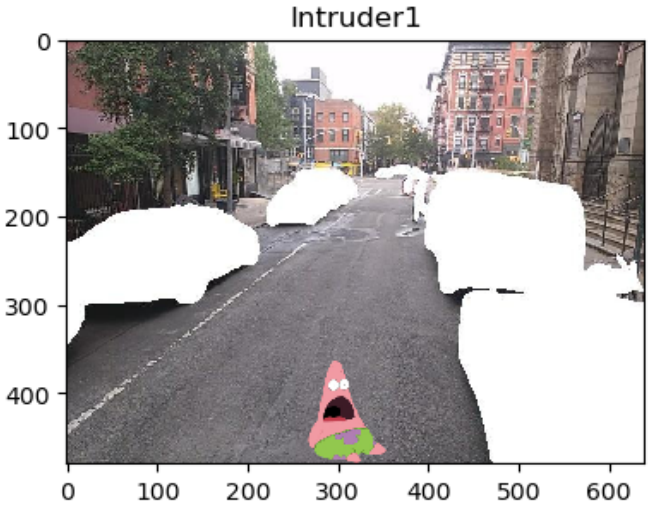

Figure 10: **d**r5rsn1w5bcx-dr5rsn1tuppy-cds-22ecab6d4a71bfcf-20160901-1157-7368.jpg

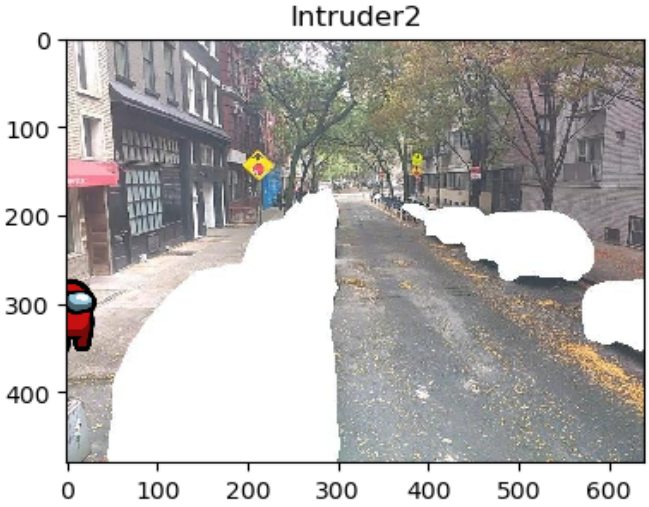

Figure 11: **d**r5rsn0yet6m-dr5rsn0ympkm-cds-3670f40ef7987d5a-20161022-1109-395.jpg

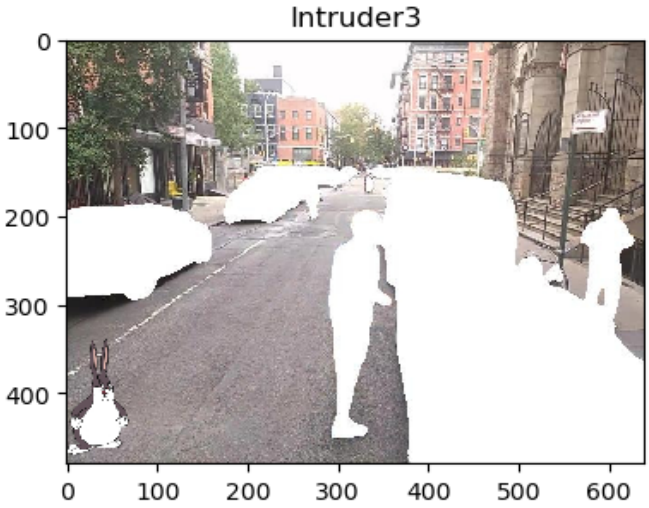

Figure 12: **d**r5rsn1tgm9f-dr5rsn1tggkn-cds-22ecab6d4a71bfcf-20160901-1157-7188.jpg