[Fall 2024] ROB-GY 6203 Robot Perception Homework 2

Akanksha Murali (am14013)

Submission Deadline (No late submission): NYC Time 11:00 AM, November 20, 2024
Submission URL (must use your NYU account): https://forms.gle/ayzWMC4BMPxfUmhZA

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".

5. Clearly state and explain the methods you used to solve each problem in your report. If applicable, reference the code you wrote and explain how it was used to generate your results. Make sure the code is well-organized and corresponds to the methods discussed in the report.

Contents

Task 1. RANSAC Plane Fitting (3pt) **2**

Task 2. ICP (3pt) **3**
a) (2pt) . . . 3
b) (1pt) . . . 4

Task 3. F-matrix and Relative Pose (3pt) **5**
a) (1pt) . . . 5
b) (1pt) . . . 5
c) (1pt) . . . 5

Task 4. Object Tracking (3pt) **7**

Task 5. Skiptrace (3pt) **8**

Task 1. RANSAC Plane Fitting (3pt)

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.

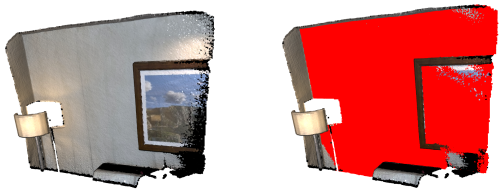


Figure 1: **Left:** Original Data, **Right:** Data with the best fit plane marked in red.

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:
Code - Task1.py

The threshold value of 0.03409 was selected for RANSAC with sample probability of p = 0.95.

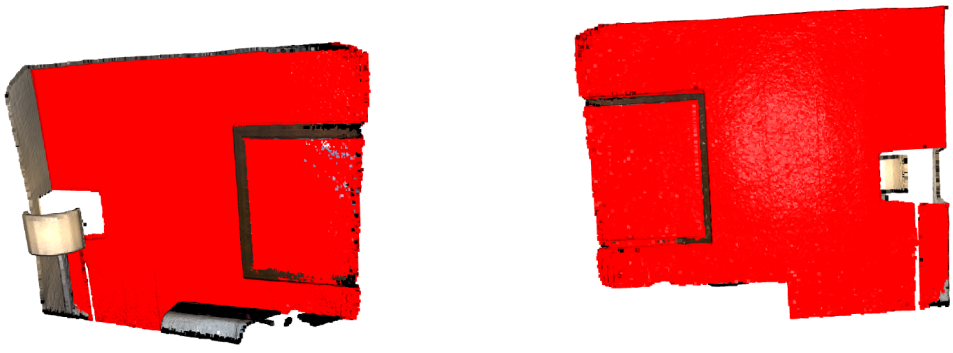


Figure 2: **Left:** Data with best fit plane- Front, **Right:** Data with the best fit plane marked - Back.

Task 2. ICP (3pt)

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:
Code - Task2.py

Figure 3 shows the result for the ICP implementation done. The cost for the problem converged to 7.4287215 after 63 iterations

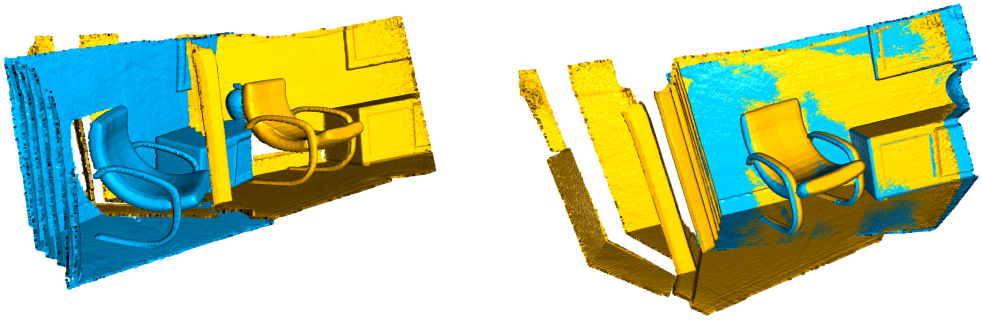


Figure 3: **Left:** Original Data, **Right:** ICP corrected data.

b) (1pt)

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:**
This section ran the same code with the Kitti pcd dataset. Figure 4 shows the results. As we can see from the visualization, the before and after doesn't change much. The overlap is not as good as in section a. This is because the cost in this section converged to 103.219059 after 43 iterations. Since the cost represents the cartesian distance between the point in source and target, we can say that 103 units is not a good distance to converge at for perfect overlap.

Figure 4: **Left:** Original Data, **Right:** ICP corrected data.

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

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) (1pt)

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.

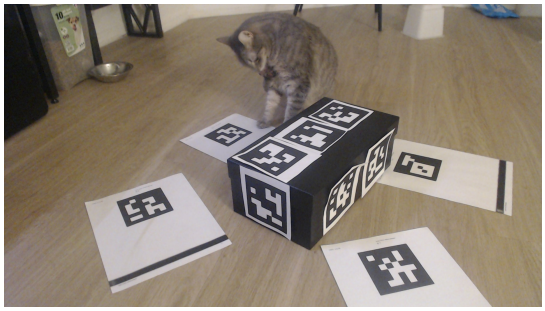


Figure 5: left.jpg

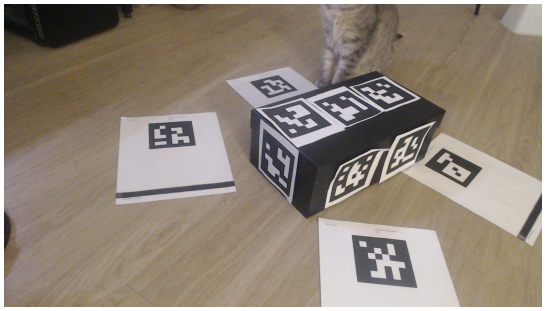


Figure 6: right.jpg

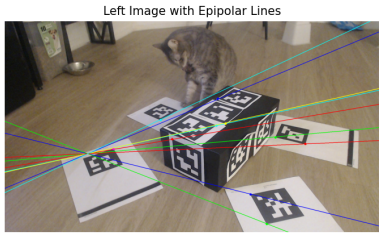
Answers:
Code - Task3.py

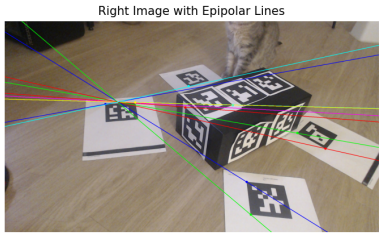
```
Fundamental Matrix:
[[-3.99398149e-07 -1.68560620e-06  1.11423428e-03]
 [ 1.02037760e-06 -1.55671456e-07  4.52505107e-03]
 [-1.15104117e-03 -3.51940405e-03  9.99982285e-01]]
```

b) (1pt)

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

Answers:





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, the Fundamental matrix alone is insufficient to calculate the R and T between the camera poses.

2. The camera's intrinsic parameters is also needed to estimate the R and T between the camera poses.To estimate R and T, we perform a SVD on KT · F ·K

```
Rotation Matrix R:
[[ 0.94673012 -0.24524912  0.20869823]
 [ 0.26109834  0.96392735 -0.05168866]
 [-0.18849333  0.10342597  0.97661319]]

Translation Vector t:
[[-0.84642545]
 [-0.06606159]
 [ 0.52839363]]
```

Task 4. Object Tracking (3pt)

Given a short video sequence, persistently track the entities in said sequence across time until it goes out of frame, or the sequence terminates. You can find the aforementioned sequence in the *data/Task4 folder of this Overleaf project*. You are free to use any tracking algorithm or pre-trained model for the task. With your implementation, answer the following two questions:

1. Can you explain in detail, the process or algorithm you used to perform the tracking?

2. Additionally, can you provide a few example frames from your resulting sequence with the proper visualization to demonstrate the efficacy of your implementation?

Note: By *tracking*, it means that you should be able to return the bounding box or centroid coordinate(s) of the entities in the video across the entire sequence.

Tip 1: For the second part of the question, you should provide an example via a few adjacent frames so that the tracking performance is obvious. You should also use consistent labelling or color coding for your visualization corresponding to each unique entity for consistency if possible.

Tip 2: You should yield something like the following for your implementation and submission.

Figure 7: Frame t_1

Figure 8: Frame t_2

Answers:
Code - Task4.py

1. To perform the tracking, Lucas-Kanade method has been applied to estimate the optic flow of points on the frame and have visualized it to represent the flow. The method vectorizes a specific point of interest over successive frames, and to visualize this, we store the point of interest over a set of frames and plot the trajectory over frames.

2.

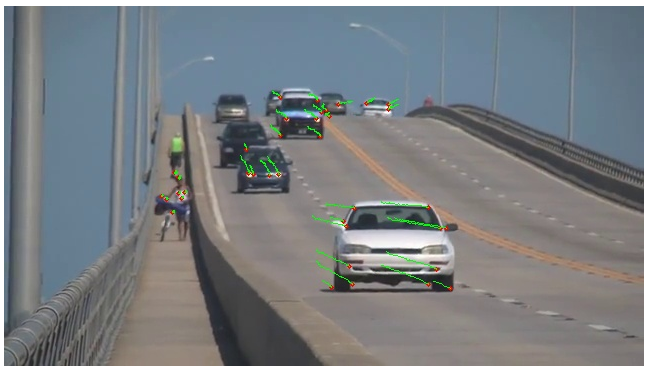


Figure 9: Frame t_1

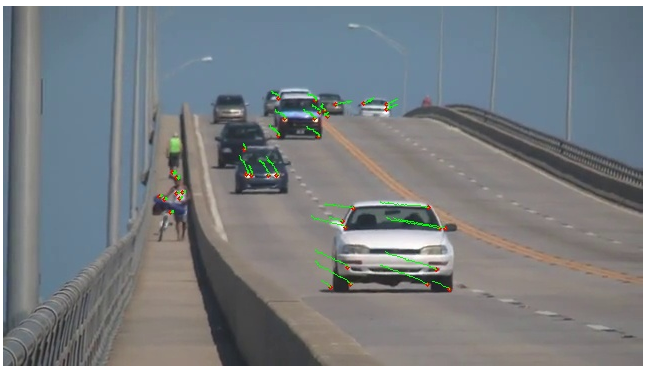


Figure 10: Frame t_2

Task 5. Skiptrace (3pt)

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:
Code - Task5.py

In this task, we used SIFT to identify keypoints and descriptors in both query images and surveillance images from the database. By comparing the descriptors of each query image with those in the database, we calculated a similarity score based on the number of good matches found. If the similarity exceeded a threshold of 0.9, the database image was considered a match. The matched images were then copied to the detection folder for further analysis. These matches indicate the possible locations of the associates, and reviewing these images will help Sherlock track them down and potentially defeat Professor Moriarty.

```
Query Image:  query_1.jpg
High similarity found:  dr5rsn0yet6m-dr5rsn0ympkm-cds-3670f40ef7987d5a-20161022-1109-395.jpg
Similarity Index:  0.9537105069801617
Stopped further comparisons for query image: query_1.jpg
Query Image:  query_2.jpg
High similarity found:  dr5rsn1tgm9f-dr5rsn1tggkn-cds-22ecab6d4a71bfcf-20160901-1157-7188.jpg
Similarity Index:  0.9624819624819625
Stopped further comparisons for query image: query_2.jpg
Query Image:  query_3.jpg
High similarity found:  dr5rsn1w5bcx-dr5rsn1tuppy-cds-22ecab6d4a71bfcf-20160901-1157-7368.jpg
Similarity Index:  0.9674721189591078
Stopped further comparisons for query image: query_3.jpg
```

Figure 11: Output

Figure 12: Query 1 Match

Figure 13: Query 2 Match

Figure 14: Query 3 Match