CS 548: Assignment 04

Programming Assignments (95%)

python/Assign04.py

In this assignment, you will implement the Point-to-Point ICP approach in Python. All functions assume numpy parameters.

You should include the following (among other things):

from General_Assign04 import *

Define these functions:

- **def get_matching_points(p_points, q_points):**
  - Build a search tree from q_points so you can search for nearest neighbors
    - I would recommend the cKDTree from scipy.spatial
  - Find the INDICES of the nearest q_point to each p_point
  - Return the indices
- **def get_centered_cloud(points):**
  - COPY the cloud first (you can use np.copy)
  - Compute the centroid
  - Center the cloud
  - Return the centered points, and the centroid
- **def compute_point_to_point_iteration(p_points, q_points):**
  - Get the matching points from q_points → q_matches
  - Get the centered clouds (from p_points and q_matches)
  - Get the cross-variance
  - Compute SVD
    - You can use np.linalg.svd(), BUT note that Vh is in fact the TRANSPOSE of V
  - Compute the rotation matrix R
  - If the determinant of R is less than zero, flip the sign of the last column of V and recalculate R
  - Get the translation vector Tr
  - Get updated_p_points by transforming p_points
  - Return the updated_p_points, q_matches, R, and Tr
- **def compute_RMSE(p_points, q_points):**
  - Calculate and return the root-mean-square-error (RMSE) between the two sets of points

- **def do_point_to_point_icp(p_points, q_points, max_iter, min_rmse):**
    - Initialize the current_rmse to some large value, iterations to 0, and total_transform to the identity matrix (np.eye(4))
    - While the current_rmse is greater than min_rmse:
        - Do one iteration of point-to-point ICP
        - Get the current transform (4x4) using **create_transform_4x4() (provided in General_A04.py)**
        - Multiply the current transform ON THE LEFT of the total_transform to get the new total_transform
        - Compute the updated RMSE
        - Update p_points
        - Increment the number of iterations
        - If the number of iterations is greater than or equal to max_iter, break
    - Return the updated p points, total_transform, the number of iterations, and the current (last) RMSE

While optional, you may find it helpful to have the following main function for visualization purposes:

```
def main():
    # Set key (see General_Assign04 for list of options)
    key = "BUNNY_ROT_SMALL_ALL_TR_BIG"

    # Get example
    example = filename_pairs[key]

    # Load data
    p_cloud, q_cloud, R, Tr = prepare_pair(example)

    # Get actual points
    p_points = np.asarray(p_cloud.points)
    q_points = np.asarray(q_cloud.points)

    # Copy p_cloud
    orig_p_cloud = copy.deepcopy(p_cloud)

    # Set current error to max error to start
    MAX_ERROR = 40000000
    current_error = MAX_ERROR
    current_iteration = 0

    # Set minimum error
    min_error = 1e-6
```

```
    # Animation function
    def animate(vis):
        nonlocal p_points
        nonlocal current_error
        nonlocal current_iteration

        if current_error > min_error:
            # Do Point-to-Point
            updated_p_points, q_matches, R, Tr =
compute_point_to_point_iteration(p_points, q_points)

            # Increment iteration
            current_iteration += 1

            # Get RMSE
            current_error = compute_RMSE(updated_p_points, q_matches)
            print("Iteration %03d, RMSE: %f" % (current_iteration,
current_error))

            # Set new points
            p_points = updated_p_points

            # Update cloud visualization
            p_cloud.points = o3d.utility.Vector3dVector(updated_p_points)
            vis.update_geometry(p_cloud)

        return False

    def reset_cloud(vis):
        nonlocal p_points
        nonlocal current_error
        nonlocal current_iteration

        current_error = MAX_ERROR
        current_iteration = 0
        p_points = np.asarray(orig_p_cloud.points)
        p_cloud.points = orig_p_cloud.points
        vis.update_geometry(p_cloud)

    def close_window(vis):
        vis.close()
        return False

    # Create visualization
```

```
    vis = o3d.visualization.VisualizerWithKeyCallback()
    vis.create_window("ICP", width=800, height=600)
    vis.add_geometry(p_cloud)
    vis.add_geometry(q_cloud)

    vis.register_key_callback(256, close_window)
    vis.register_key_callback(ord(" "), reset_cloud)

    while True:
        animate(vis)
        if not vis.poll_events():
            break
        vis.update_renderer()

    vis.destroy_window()

if __name__ == "__main__":
    main()
```

Testing Screenshot (5%)

I have provided several files for testing:

- data/assign04
  - input/ - contains input cloud files
  - ground/ - contains the ground truth files (of which there are many)
- python/
  - Test_Assign04.py – the test program for the Python code
  - General_Assign04.py – globals and helper functions

Run the testing program through the testing section of Visual Code.

You MUST run the tests and send a screenshot of the test results! Even if your program(s) do not pass all the tests, you MUST send this screenshot!

Python Tests

You may have to do “Command Palette” → “Python: Configure Tests” → pytest → python (directory)

You should then be able to run the Python tests in your testing window in Visual Code.

ALTERNATIVELY: open a terminal and enter: **pytest python/Test_Assign04.py**

…then take a screenshot of the terminal output.

Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 95% - Programming assignments

I reserve the right to take points off for not meeting the specifications in this assignment description. In general, these are things that will be penalized:

- **Code that is not syntactically correct (up to 60 points off!)**
- Sloppy or poor coding style
- Bad coding design principles
- Code that crashes, does not run, or takes a VERY long time to complete
- Using code from ANY source other than the course materials
- Collaboration on code of ANY kind; this is an INDIVIDUAL PROJECT
- Sharing code with other people in this class or using code from this or any other related class
- Output that is incorrect
- Algorithms/implementations that are incorrect
- Submitting improper files or failing to submit ALL required files