CIS PA 4

MENGZE XU, LIUJIANG YAN

Summary
This report is for programming assignment 4 iterative closest points algorithm implementation, containing following parts,

* Mathematical Approach and Algorithm
* Programming Structure
* Validation and Results
* Discussion
* Summary of Unknown Data

The folder contains following subfolders and Matlab files,

* IterativeClosestPoint - Functions for finding closest points in mesh
  - find_closest_point_in_triangle.m
  - linear_search_brute_force.m - find the closest point in mesh by brute force
  - registration.m: 3D point set to 3D point set registration
  - registrationBySVD.m: 3D point set to 3D point set registration by SVD method
  - BoundingSphere&Box
    * radius_center_of_sphere.m
    * linear_search_bounding_spheres.m
    * bound_of_box.m
    * linear_search_bounding_boxes.m
  - Tree
    * OcTree
    * KdTree
    * TreeSearch
* Parse - Matlab functions for parsing data files
  - parseBody.m
  - parseMesh.m
  - parseSample.m
* PA234-StudentData - Data for PA4
* PA4Driver.m - Driver script for PA4 and leads to pa4output files
* PA4Validation.m - Matlab script for validation and error analysis
* PA4OutputData - Output data for different data set
* PA4OutputFig - Output figures for different search methods

Mathematical Approach and Algorithm

(a) octree search

The previous version (PA3) of octree search method is combined by brute force linear search and spatial aligning by octree. Though the introduction of octree greatly reduces the times of boundary check. However, the search time is mostly dependent on how many times of the careful triangle check.

Besides on the efficiency discussed in previous programming assignment, we know that bounding spheres outperform other search methods on reducing the careful triangle check times. Therefore in this assignment, we intend to improve the current octree search method by introducing the bounding sphere when the octree search comes to leaf node.

Algorithm 1 octree search improved by bounding sphere

```
1: if $s_x > octree.upper_x + bound$ or $s_x < octree.lower.x - bound$ then
2:     return
3: end if
4: if $s_y > octree.upper_y + bound$ or $s_y < octree.lower.y - bound$ then
5:     return
6: end if
7: if $s_z > octree.upper_z + bound$ or $s_z < octree.lower.z - bound$ then
8:     return
9: end if
10: if has subtree then
11:     for $i = 1$ to number of subtrees do
12:         subtree ← octree.childi
13:         subtee.octree_search()
14:         update bound, closest point
15:     end for
16: else
17:     triangle subset ← triangle set(octree.index)
18:     for $i = 1$ to number of triangles do
19:         if $||q_i - a|| - r_i \leq$ bound then
20:             $c_i \leftarrow$ findTheClosestPoint($s_i$, $triangle_i$)
21:             $d_i = ||s_i - c_i||$
22:             if $d_i \leq$ bound then
23:                 bound ← $d_i$
24:                 closest point ← $c_i$
25:             end if
26:         end if
27:         update bound, closest point
28:     end for
29: end if
```

(b) kd tree search
Besides octree, we also implemented kd tree to see the difference of efficiency. Different from octree, kd tree in 3D space has two children for each node, which contains almost the same points in each child, and leads to a more balanced spatial division. For convenience, we firstly divide the space by x coordinate (by taking the mean of all the points' x coordinate in the space), followed by y coordinate and then z coordinate and back to x.
The class definition and properties for kd tree object.

Algorithm 2 Class Definition of kdtree

```
1: classdef Octree
2: Properties
   split face, upper bound, lower bound, centers, index, child
3: Methods
   kdtree();
   enlarge_bound(this);
```

The construct method for kd tree.

Algorithm 3 kdtree construct method

```
 1: if this.centers $< 7$ then
 2:    return;
 3: end if
 4: subtree bound $\leftarrow$ compute_subtree_bound(bound,split face)
 5: for $i = 1$ to 2 do
 6:    subtree centers $\leftarrow$ compute_subtree_centers(centers, subtree bound)
 7:    if number of subtree centers $> 0$ then
 8:       subtree $\leftarrow$ kdtree();
 9:       this.childi $\leftarrow$ subtree;
10:    end if
11: end for
```

Inspired by the construction process of kd tree, we could come back to octree and improve. Originally, the division for each tree level is to take the midpoint of the space, which may lead to unbalanced structure. Instead, like what kd tree has done, we could choose the mean value of x, y, z coordinates of the points in space. We will discuss the efficiency gains for each method in discussion part following.
(c) iterative closest points algorithm
The iterative closest points is based on PA3's one time closest point algorithm, by introducing registration updating process. Also another key point is to choose a proper stopping condition.
The stopping condition are combined by: accuracy, maximum iterations and convergence. The accuracy are measured by the distance between rigid transformed local sample points and the corresponding closest points. The maximum iterations is set default as 100 times. The convergence is measured by the delta term of the update process, if the delta term is almost identity matrix then we consider it convergence.

Algorithm 4 iterative closest point

```
1: Freg ← $I_{4\times4}$ (for initial guess)
2: while error $>$ 0.1 and norm(ΔFreg - $I_{4\times4}$)>0.01 or iteration<max iteration do
3:     $s$ ← rigid transformation($d$, Freg)
4:     $c$ ← find closest points in mesh($s$)
5:     error← $|s - c|$
6:     ΔFreg←registration($s$,$c$)
7:     Freg← ΔFreg·Freg
8: end while
```

Programming Structure
(a) Phase Functions
We implements parseBody.m for reading markers' information, parseMesh.m for mesh information and parseSample.m for markers' local coordinates.
(b) Programming Structure
We briefly sum up the process we performed to implement the iterative closest points in our driver script.

Algorithm 5 programming assignment 4 driver

```
1: initializing the workspace
2: add some file path
3: read markers and mesh information
4: build the bounding sphere, bounding box and octree
5: for each data set do
6:     read local markers coordinates, registration, compute sample points $d$
7:     while ICP process do
8:         perform ICP process to get the closest points $c$ for $d$
9:     end while
10: end for
```

Besides, we also implemented a real time plot for error term while executing the PA4Driver script. The plot will clearly show the iterative process of ICP and how the ICP process meets its stopping condition, like below. Also we display some essential computational information during the execution of driver script, including elapsed time, registration residual error, iteration experienced, error term, etc.

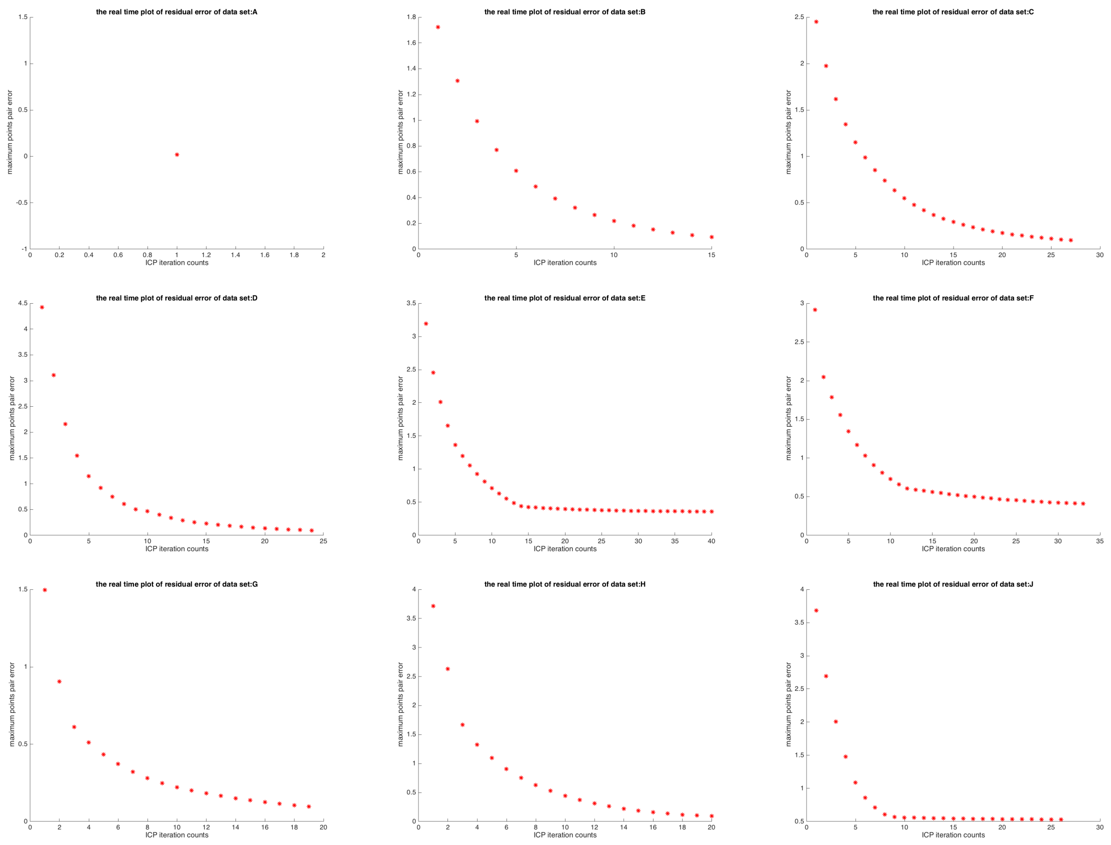

Figure 1. iterative process for each data set

Validation and Results

We previously validated our search methods in PA3. In this part, we intend to validate the whole ICP process, by comparing our computed results of the sample points' local coordinates, closest points and the distance between them, to the debug result files. More specifically, we compute the norm of each points pair in the same data set.
Here we list how the validation script execute,

Listing 1. PA5Validation.m

```
%% initialization
clear; clc;
format compact;

%% add the path
addpath('PA234 - Student Data/');
addpath('PA4OutputData/');
addpath('Parse/');

%% validate the algorithm
s_diff_norm = zeros(6,1);
c_diff_norm = zeros(6,1);
distance_diff_norm = zeros(6,1);
for char = 'A':'F'
    % read the given debug result file
    validation_set_path = strcat('PA4-', char, '-Debug-Output.txt');
    FID = fopen(validation_set_path);
    file = fgetl(FID);
    datasize = [7, Inf];
    validation_set = transpose(fscanf(FID, '%f\t%f\t%f\n', datasize));
    fclose(FID);

    % read the computed resul file
    computed_set_path = strcat('solved-PA4-',char,'-output.txt');
    computed_set = csvread(computed_set_path);

    % get the difference of three terms
    s_diff = sum((validation_set(:,1:3) - computed_set(:,1:3)).^2, 2).^1/2;
    s_diff_norm(abs(char)-64) = sum(s_diff.^2, 1).^1/2;
    c_diff = sum((validation_set(:,4:6) - computed_set(:,4:6)).^2, 2).^1/2;
    c_diff_norm(abs(char)-64) = sum(c_diff.^2, 1).^1/2;
    distance_diff = validation_set(:,7) - computed_set(:,7);
    distance_diff_norm(abs(char)-64) = sum(distance_diff.^2, 1).^1/2;

    % display the results
    disp(strcat('data set:', char));
    disp(strcat('the difference of sample points coordinates:', ...
        num2str(s_diff_norm(abs(char)-64))));
    disp(strcat('the difference of closest points coordinates:', ...
        num2str(c_diff_norm(abs(char)-64))));
    disp(strcat('the difference of closest distance:', ...
        num2str(distance_diff_norm(abs(char)-64))));
    disp('——————————————————————————————————————————————————');
end
```

The difference of each term are showed as bar plot below,

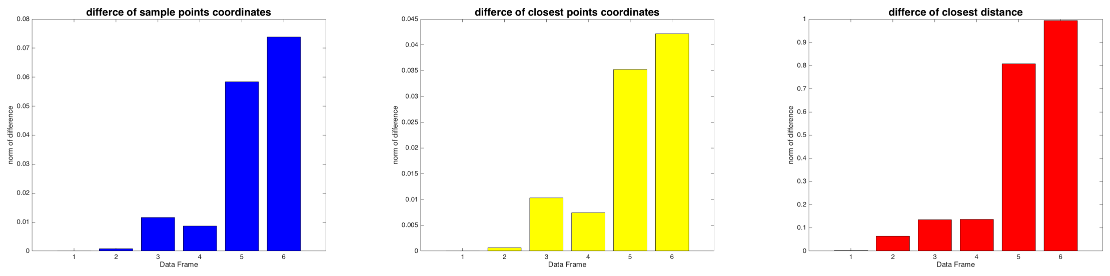

Figure 2. norm of difference for each data set

From the plot we can see, the fifth and sixth data set have relatively more significant error than the former four data set. The error may have two possible sources.
The first one is the ICP process. However, since the process is general then if the process is somewhere imperfect, the error should occur in each data frame but not only the last two data set.
The second one is the registration process for computing samples points' coordinates, which is data-specific. Since there might be some noise among the given sample points, which may lead to inaccurate registration, and will present such inaccuracy as the residual error. Recalling our registration process, we could define the residual error as,

$$d_k = F_{B,k}^{-1} F_{A,k} a_{tip}$$

where,

$$A_k = F_{A,k} a \quad B_k = F_{B,k} b$$

then the residual error is shown as below, where P stands for A or B.

$$res = \sum (F_{P,k} p - P)$$

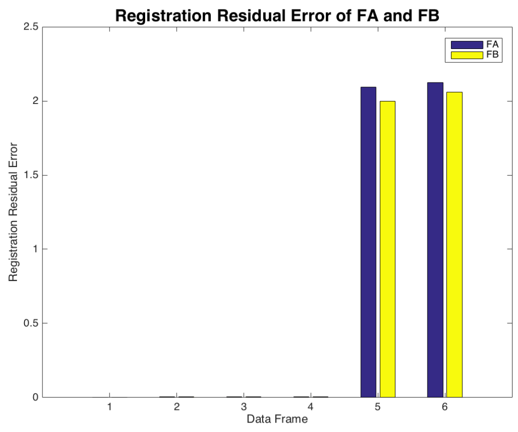

Figure 3. Summing Residual Error of FA and FB for each data set

The summing residual error of registration has the almost same pattern as the norm of difference above.
The residual error may be caused by the registration process. Besides the regular registration method by solving least square problem, we also implement another registration method by using Singular Value Decomposition (SVD).

Algorithm 6 registration by singular value decomposition

```
1: find the centroid of A, $C_A \leftarrow 1/N \sum P_A$
2: find the centroid of a, $C_a \leftarrow 1/N \sum P_a$
3: construct the covariance matrix, $H \leftarrow \sum (P_A^i - C_A)^T (P_a^i - C_a)$
4: SVD, $[U, S, V] = svd(H)$
5: rotation and translation, $R = VU^T$ and $t = -RC_A + C_a$
```

Use the same definition of residual error as above, we could measure the process by comparing the residual error derived by these two methods. We plot the residual error by different methods in each data set.

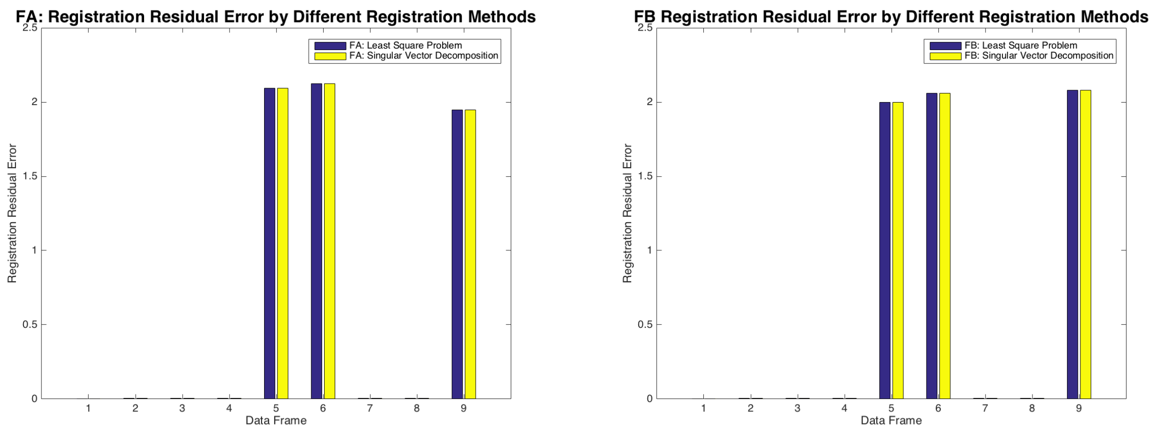

FIGURE 4. Registration Residual Error Comparison

We find that, the methods give almost the same residual error level, then we may concludes that the error should be method-independent, which could lead to a safe conclusion that, the significant difference among the computed and debug result is related to the residual error of transformation.

Discussion

Here we would like to discuss about the efficiency of those search methods. Beginning from brute force linear search, bounding boxes and spheres are still linear search, but improved the efficiency by substituting some careful triangle checks to boundary check. Then the t data structure octree is introduced to reduce the boundary check by introducing spatial division.
Also, if we look inside the ICP process, finding closest points in mesh for each sample point actually could be done in parallel, which could take advantage of multi-core CPU.
Here we recorded different methods' time consuming for every data set.

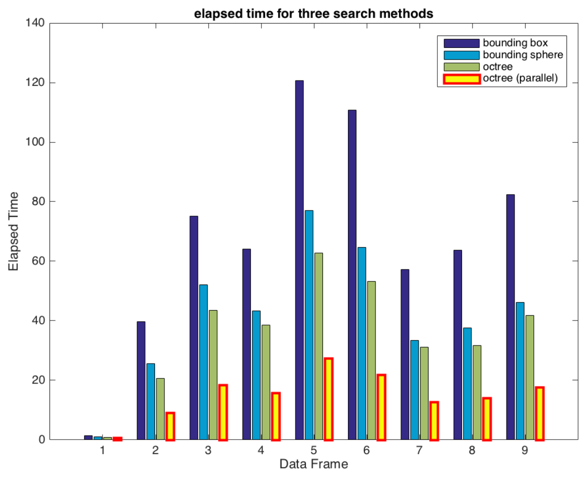

FIGURE 5. Time Consuming for Each Search Method

We could see that, depth first search of hierarchy data structure strictly dominates other two linear search methods. Besides, introducing parallel programming speeds up twice (for your information, my cpu is quad-core).
Also, we compared the efficiency of original octree (split space by geometrical mid point), kd tree, and balanced octree (split space by mean point) for each data set to see the difference of each method. Note that we make every other issue the same.

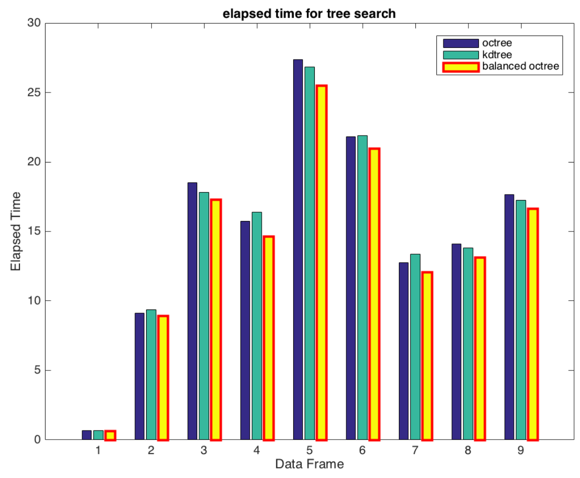

FIGURE 6. Time Consuming for Each Tree Method

A bit analysis for the result above is: though kdtree is a balanced structure while the original octree is not, which may save some node check (since balanced structure usually has fewer levels). However octree has eight children, 4 times as the kdtree. So that the efficiency of these two data structure are almost the same. A balanced structure of octree may reduce the depth of the tree or divide the points more evenly, and leads to more efficient result, as the plot shown above.

Summary of Unknown Data

(1) All the result files are in path **\PA4OutputData** with file name as **\solved-PA4-*index*-output**.
(2) Inside the data file, the result are listed line by line as, x y z coordinates of d_k, x y z coordinates of c_k, and magnitude of difference.
(3) The iterative process for different search methods are in path **\PA4LogFile** with file name as **\*method*_log**. The figures for the iterative process of different search methods are in **\PA4OutputFig**.