CIS PA 3

MENGZE XU, LIUJIANG YAN

Summary
This report is for programming assignment 3 matching of the iterative closest point algorithm, 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,

* CP - Essential Matlab 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
  - BoundingSphere&Box
    * radius_center_of_sphere.m
    * linear_search_bounding_spheres.m
    * bound_of_box.m
    * linear_search_bounding_boxes.m
  - Octree
    * octree_object.m
    * compute_subtree_bound.m
    * compute_subtree_centers.m
    * octree_search.m
    * find_update_closest_point.m
* Parse - Matlab functions for parsing data files
  - parseBody.m
  - parseMesh.m
  - parseSample.m
* PA234-StudentData - Data for PA3
* PA3driver.m - Driver script for PA3 and leads to pa2output files
* Validation - Matlab script for validation and error analysis
  - PA3Validation.m
  - PA3DoubleCheckICP.m
  - PA3DoubleCheckRegistration.m
* PA3output - Output data

Mathematical Approach and Algorithm

(a) find the closest point in triangle

Given a point with position s_i, we follow the process below to find the closest point in triangle with vertexes p q r.

First of all we slove the following linear system to find the parameter λ and μ.

$$s_i - p = \lambda(q - p) + \mu(r - p)$$

This will lead to four cases,

(1) $\lambda > 0$, $\mu > 0$ and $\lambda + \mu < 1$, which means the closest point is inside the triangle, so that,

$$c = p + \lambda(q - p) + \mu(r - p)$$

(2) $\lambda < 0$, which means the closest point lies in the line p and r,

$$\lambda' = \frac{(c - r) \cdot (p - r)}{(p - r) \cdot (p - r)}$$
$$\lambda^* = max(0, min(\lambda', 1))$$
$$c^* = r + \lambda^*(p - r)$$

(3) $\mu < 0$, which means the closest point lies in the line p and q,

$$\lambda' = \frac{(c - p) \cdot (q - p)}{(q - p) \cdot (q - p)}$$
$$\lambda^* = max(0, min(\lambda', 1))$$
$$c^* = p + \lambda^*(q - p)$$

(4) $\lambda + \mu > 1$, which means the closest point lies in the line r and q,

$$\lambda' = \frac{(c - q) \cdot (r - q)}{(r - q) \cdot (r - q)}$$
$$\lambda^* = max(0, min(\lambda', 1))$$
$$c^* = q + \lambda^*(r - q)$$

(b) linear search through all the triangles

The basic idea of linear search by brute force to find the closest point in mesh (a set of triangles) with given a point s is performed as,

Algorithm 1 linear search by brute force

```
minimum distance ← ∞
for i = 1 to number of triangles do
    c_i ← findTheClosestPoint(s_i, triangle_i)
    d_i = ||s_i − c_i||
    if d_i ≤ minimum distance then
        minimum distance ← d_i
        closest point ← c_i
    end if
end for
```

(c) linear search improved by bounding box

The idea of bounding box is to represent the triangles by a box bounding around the triangles,

which could reduce some careful checks required in purely brute force search.
We assign the minimum and maximum coordinates of triangles' vertexes as the lower as $[L_x, L_y, L_z]$ and upper bound as $[U_x, U_y, U_z]$. Then we follow the algorithm below,

Algorithm 2 linear search by bounding boxes

```
bound ← ∞
for i = 1 to number of triangles do
    if L_x − bound ≤ s_x ≤ U_x + bound and
    L_y − bound ≤ s_x ≤ U_y + bound and
    L_z − bound ≤ s_z ≤ U_z + bound then
        c_i ← findTheClosestPoint(s_i, triangle_i)
        d_i = ||s_i − c_i||
        if d_i ≤ bound then
            bound ← d_i
            closest point ← c_i
        end if
    end if
end for
```

(d) linear search improved by bounding sphere
Silently different from the bounding box, the bounding sphere method represent the triangles by radius and center.
Here we assume the a and b are the endpoints of the longest line.
Compute the midpoints f as

$$\vec{f} = \frac{\vec{a} + \vec{b}}{2}$$

Define,

$$\vec{u} = \vec{a} - \vec{f}$$
$$\vec{v} = \vec{c} - \vec{f}$$
$$\vec{d} = (\vec{u} \times \vec{v}) \times \vec{u}$$

Then the center of sphere q must lie along the line,

$$\vec{q} = \vec{f} + \lambda d$$

The distance from center to c must be smaller than from center to a, which leads to following inequality.

$$(\vec{f} + \lambda d - \vec{c})^2 \leq (\vec{f} + \lambda d - \vec{a})^2$$
$$(\lambda \vec{d} - \vec{v})^2 \leq (\lambda \vec{d} - \vec{u})^2$$

We could get some knowledge about λ as,

$$\lambda \geq \frac{\vec{v}^2 - \vec{u}^2}{2\vec{d} \cdot (\vec{v} - \vec{u})} \triangleq \gamma$$

If $\gamma < 0$ then $\lambda = 0$, else $\lambda = \gamma$. And the radius is given by $r = ||\vec{q} - \vec{a}||$.
Once we have the radius and center for each triangle, we could follow the linear search algorithm shown below to find the closest point in mesh.

Algorithm 3 linear search by bounding spheres

bound $\leftarrow \infty$
for $i = 1$ to number of triangles **do**
  if $||q_i - a|| - r_i \leq$ bound **then**
    $c_i \leftarrow$ findTheClosestPoint(s_i, $triangle_i$)
    $d_i = ||s_i - c_i||$
    if $d_i \leq$ bound **then**
      bound $\leftarrow d_i$
      closest point $\leftarrow c_i$
    end if
  end if
end for

(e) octree search
Another method, instead of linear search, is to construct hierarchical data structure to speed. Here we present how to construct an octree including all the triangles.
The octree class has following properties and methods

Algorithm 4 Class Definition of Octree

classdef *Octree*
Properties
split point, upper, lower, centers, index, child, depth
Methods
Octree();
enlarge_bound(this);

A brief introduction of the construct method is given below. We assign the centers of our triangles to each tree node.

Algorithm 5 octree construct method

if this.centers < 7 **then**
  return;
end if
for $i = 1$ to 8 **do**
  subtree bound $\leftarrow$ compute_subtree_bound(bound,i)
  subtree centers $\leftarrow$ compute_subtree_centers(centers, subtree bound)
  if number of subtree centers > 0 **then**
    subtree $\leftarrow$ Octree();
    this.childi $\leftarrow$ subtree;
  end if
end for

After constructing the octree, recalling that in the search process we need to deal with the bound of each tree node, we need to go through another process to enlarge bound so that each triangle in the node are fully included.

Algorithm 6 octree enlaring bound

```
triangle subset ← triangle set(octree.index)
bound of box ← bound_of_box(triangle subset)
update the upper and lower boundary
if has subtree then
    for i = 1 to number of subtrees do
        subtree ← octree.childi
        subtree.enlarge_bound()
    end for
end if
```

Once we have the octree, given any point s_i, we could perform a depth first search based method to find the closest point in mesh.

Algorithm 7 octree search

```
if s_x > octree.upper_x + bound or s_x < octree.lower.x − bound then
    return
end if
if s_y > octree.upper_y + bound or s_y < octree.lower.y − bound then
    return
end if
if s_z > octree.upper_z + bound or s_z < octree.lower.z − bound then
    return
end if
if has subtree then
    for i = 1 to number of subtrees do
        subtree ← octree.childi
        subtee.octree_search()
        update bound, closest point
    end for
else
    triangle subset ← triangle set(octree.index)
    for i = 1 to number of triangles do
        c_i ← findTheClosestPoint(s_i, triangle_i)
        d_i = ||s_i − c_i||
        if d_i ≤ bound then
            bound ← d_i
            closest point ← c_i
        end if
        update bound, closest point
    end for
end if
```

We can also package the update bound and closest point while iterating each triangle in subtree in a function.

(d) matching sample points to mesh
We follow the process listed below to implement to mathcing part of iterative closest points.

(1) get the body definition files, recorded as a and b
(2) for each data frame k, get the body's coordinates as A_k and B_k
(3) the position d_k of the pointer tip with respect to rigid body B can be derived by composition rule,
$$d_k = F_{B,k}^{-1} F_{A,k} a_{tip}$$
where,
$$A_k = F_{A,k} a \quad B_k = F_{B,k} b$$
(4) give an initial guess for registration transformation as F_{reg} such that,
$$s_k = F_{reg} d_k$$
Then, we perform the finding closest point in mesh algorithm derived above to find c_i that are closest to s_i.

Programming Structure
(a) Phase Functions
Here we show how we phase the data file and get what we want.
Take the parse function for body as example, with filepath as input, we return the number of markers in body, the position of markers and the position of tip.

LISTING 1. parseBody.m

```
function [N_markers, P_markers, P_tip] = parseBody(filepath)
    FID = fopen(filepath);
    file = fgetl(FID);
    % get the first line
    header = textscan(file,'%f');

    % extract the numerical data
    N_markers = header{1,1};

    size = [3, Inf];
    Matrix = transpose(fscanf(FID, '%f\t%f\t%f\n', size));

    % markers' coordinates
    P_markers = Matrix(1:N_markers,:);

    % tip's coordinate
    P_tip = Matrix(end, :);

    fclose(FID);
end
```

(b) Programming Structure
The main script for programming assignment 3 is **PA3driver.m**.
1. The first part of it is to add some essential paths and clear the workspace.

```
%% initialization
```

```
clear;
clc;
format compact;

%% add the path
addpath('PA234 - Student Data/');
addpath('Parse/');
addpath(genpath('ICP/'));
```

2. Then we read body files and mesh files.

```
%% read Body A B and mesh files
body_A_filepath = 'PA234 - Student Data/Problem3-BodyA.txt';
[num_a, a, a_tip] = parseBody(body_A_filepath);

body_B_filepath = 'PA234 - Student Data/Problem3-BodyB.txt';
[num_b, b, b_tip] = parseBody(body_B_filepath);

mesh_filepath = 'PA234 - Student Data/Problem3MeshFile.sur';
triangle_set = parseMesh(mesh_filepath);
```

3. In order to avoid redundant computation in the ICP process, we get the triangles' sphere parameters, box parameter in advance. Also we construct the octree and enlarge the bound for following search process.

```
%% get the bounding sphere, boxes, octree for triangle set
% radius and center for each bounding triangle
[radius, center_of_triangle] = radius_center_of_sphere(triangle_set);
% lower and upper bound for each bounding box
[triangle_box_lower, triangle_box_upper] = bound_of_box(triangle_set);
% lower and upper bound for the whole triangle set
lower_bound = min(triangle_box_lower, [], 1);
upper_bound = max(triangle_box_upper, [], 1);

index_of_triangles = (1:size(center_of_triangle, 1))';
% build tree
octree = octree_object...
    (upper_bound, lower_bound, center_of_triangle, index_of_triangles);
% enlarge the bound
octree.enlarge_bound(triangle_set);
```

4. We then iterate each data file (debug or unknown), read the samples' information.

```
for char = ['A':'H','J']
    % read the samples' information
    if ismember(char, 'A':'F')
        sample_filepath = strcat('PA234 - Student Data/PA3-', char,...
            '-Debug-SampleReadingsTest.txt');
    else
        sample_filepath = strcat('PA234 - Student Data/PA3-', char,...
```

```
            '-Unknown-SampleReadingsTest.txt');
    end
    [num_samples, A_set, B_set] = parseSample(sample_filepath, num_a, num_b);
    output = zeros(num_samples, 7);

    % The ICP process performs here.

    csvwrite(strcat('PA3output/solved-PA3-',char,'-output.txt'), output);
end
```

5. For each dataframe, we perform the subroutine of ICP process derived above to get the closest point in mesh.

```
for i=1:num_samples
    A = A_set(:,:,i);
    B = B_set(:,:,i);
    FA = registration(A,a);
    FB = registration(B,b);
    d = FB\FA*[a_tip';1];

    % initial guess
    Freg = eye(4);
    s = Freg*d;
    s = s(1:3);
    d = d(1:3);

    % find the closest point in mesh by desired method
    c = linear_search_brute_force(s, triangle_set);
    c = linear_search_bounding_spheres...
        (s, triangle_set, center_of_triangle, radius);
    c = linear_search_bounding_boxes...
        (s, triangle_set, triangle_box_lower, triangle_box_upper);
    % some initial guess for octree search
    bound = inf;
    closest_point = [1000, 1000, 1000];
    point_index = 0;
    triangle_visited = 0;
    c = octree_search(s, octree, triangle_set, ...
        bound, closest_point, point_index, triangle_visited);

    error = norm(s-c);
    output(i,:) = [d',c',error];
end
```

Validation and Results

(a) Validation of the bounding sphere, box, octree search methods

We validate our methods by comparing the result returned by brute force linear search. If the closest point returned by octree, sphere and box are individually the same as what the brute force method return, then those methods are correct.

LISTING 2. subroutine of validating the search methods

```
% find the closest point in mesh by desired method
c_brute_force = linear_search_brute_force(s, triangle_set);
c_sphere = linear_search_bounding_spheres...
    (s, triangle_set, center_of_triangle, radius);
c_box = linear_search_bounding_boxes...
    (s, triangle_set, triangle_box_lower, triangle_box_upper);
% some initial guess for octree search
bound = inf;
closest_point = [1000, 1000, 1000];
point_index = 0;
triangle_visited = 0;
c_octree = octree_search(s, octree, triangle_set, ...
    bound, closest_point, point_index, triangle_visited);
if (norm(c-c_sphere)>0) ||...
        (norm(c-c_box)>0) || (norm(c-c_octree)>0)
    disp('not correct')
end
```

Through all the PA3 data, the results returned by bounding spheres and boxes are strictly the same as the result by brute force linear search. As for the octree search, there is only one mismatching which shows an error between two returned closest point with value of $2e-16$, which can be considered as round off error.

(b) Validation of the ICP process

With the output result data and given debug output data, we are able to perform some error analysis.

Here, first of all, we present the structure of our validation Matlab script (as PA3validation.m).

LISTING 3. PA3validation.m

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

%% add the path
addpath('PA234 - Student Data/');
addpath('PA3output/');
addpath('Parse/');

%% some validation criteria
% the difference of s
diff_s = zeros(15,3,size('A':'F',2));
% the maximum difference of each pair of files
max_s = [];
% the sum of difference of each pair of files
ssd_s = [];
% the difference of c
diff_c = zeros(15,3,size('A':'F',2));
```

```
max_c = [];
ssd_c = [];
% the difference of closest distance from point to mesh
distance_set = zeros(15,size('A':'F',2));

%% validation process
i = 1;
for char = 'A':'F'
    if ismember(char, 'A':'F');
        validation_set_path = strcat('PA3-', char, '-Debug-Output.txt');
    else
        validation_set_path = strcat('PA3-', char, '-Unknown-Output.txt');
    end
    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);

    computed_set_path = strcat('solved-PA3-',char,'-output.txt');
    computed_set = csvread(computed_set_path);

    difference_s = validation_set(:,1:3) - computed_set(:,1:3);
    diff_s(:,:,i) = difference_s;
    max_s = [max_s; max(sum(difference_s.^2, 2).^1/2)];
    ssd_s = [ssd_s; sum(sum(difference_s.^2, 2).^1/2)];

    difference_c = validation_set(:,4:6) - computed_set(:,4:6);
    diff_c(:,:,i) = difference_c;
    max_c = [max_c; max(sum(difference_c.^2, 2).^1/2)];
    ssd_c = [ssd_c; sum(sum(difference_c.^2, 2).^1/2)];

    distance = validation_set(:,7) - computed_set(:,7);
    distance_set(:,i) = distance;

    i = i+1;
end
```

We record three different data, sample points' coordinates s, the corresponding closest point c and distance from s to c. The result shows that in some cases, such as file A B C and D, the data fits very well. However in file E and F we observe some significant error.

EM	$\|\|\Delta s\|\|$	$\|\|\Delta c\|\|$	$\|\|\Delta d\|\|$
A	0.0017	0.0013	0.0842
B	0.0014	0.0007	0.0169
C	0.0009	0.0004	0.0138
D	0.0011	0.0007	0.0039
E	10.8025	10.9732	0.3149
F	6.8843	5.6550	1.7986

The source of error may come from two places: the matching part of iterative closest point and the registration process. We need to separately identify which source of error leads to the mismatching of the results.

1. finding the closest point in mesh

Firstly, we double check the matching part of iterative closest point. In order to do that, we eliminate the registration error by just taking the samples' position from output debug files and comparing the closest point to our result. See **PA3DoubleCheckICP.m** for details. The difference of each pair of closest points' coordinates are less than 0.01 mm, which implies that the process here for finding the closest point in mesh is correct.

2. registration

Secondly, we double check the registration process, recalling that,

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

where,

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

We can define the residual for the registration process as, where P stands for A or B.

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

Also we compute the difference between the real samples' coordinates (from output file) and our derived samples' coordinates. See **PA3DoubleCheckRegistration.m** for details.

We observe that, there are two patterns

(1) when the residual for registration both A and B are small enough (like 10^{-5}), there is no significant error between the real samples' coordinates and our derived samples' coordinates.

(2) when the residual for registration both A and B are small enough (like 10^{-1}), there is significant error (up to several mms) between the real samples' coordinates and our derived samples' coordinates.

To sum up, through the discussion above, we could say that the error between our derived results and the given debug results are due to the registration process. More points pair $A,\ a$ and $B,\ b$ will be helpful to boost the accuracy.

A possible reason for the registration error may be the outliers while registering for F_A and F_B. To solve the registration problem, here I present RANSAC to throw out the outliers.

Algorithm 8 Random Sample Consensus

```
minimum residual ← ∞
for i = 1 to n do
    randomly pick 3 or 4 or 5 pair of points, registration
    if the residual is smaller than minimum residual then
        final F ← F
        minimum residual ← residual
    end if
end for
```

However, we performed the RANSAC for 3, 4 and 5 pairs of points, which gives the worse result (larger residual) than the origianl registration method. It turns out that the error

might more probably be caused by noise but not outliers.

Discussion

Comparing to linear search, bounding sphere and bounding box improve the efficiency and speed up due to less careful triangle check and substitute by simple boundary check.

Having a insight in the triangles checked for each iterative closest point process, we find that, bounding sphere checks only 10% of the triangles set, which leads to a 10 times speed than brute force search, and bounding box checks a little more than bounding sphere.

Though, since we have a large set of triangles, linear search by bounding sphere or box still needs thousands of bound check, which therefore could be improved and reduced by introducing hierarchical data structure, as octree.

Summary of Unknown Data

All the result files are in path **\PA3output** in file name as **\solved-PA3-*index*-output**. Inside the data file, the result are listed line by line as,xyz coordinates of d_k , xyz coordinates of c_k, and magnitude of difference.