Programming Assignment 3

Zhiyuan Ding zding20@jh.edu, Deming(Remus) Li dli90@jh.edu

November 17, 2021

1 Math approaches

1.1 Frame transformation in step 1

We first briefly introduce the frame transformation to obtain d_k.

First, based on the frame relationship in each sample frame k, a point rigid registration is needed to obtain the frame transformation$F_{A,k}$,$F_{B,k}$ between the rigid bodiesA, B with respect to the tracker for each sample frame.

Then, the position of tip A relative to the the rigid bodies$B$ is computed by the following relationships:

$$d_k = F_{B,k}^{-1} F_{A,k} \vec{A_{tip}}$$

1.2 Find closest point for a triangle and a point

Here we illustrate the computation process to find the nearest point for point a and triangle with vertices $[p, q, r]$. There are four different regions as in Figure 1. We first solve the least square for the relationship:

$$a - p \approx \lambda(q - p) + \mu(r - p)$$

and depend on λ and μ we get these four following conditions:

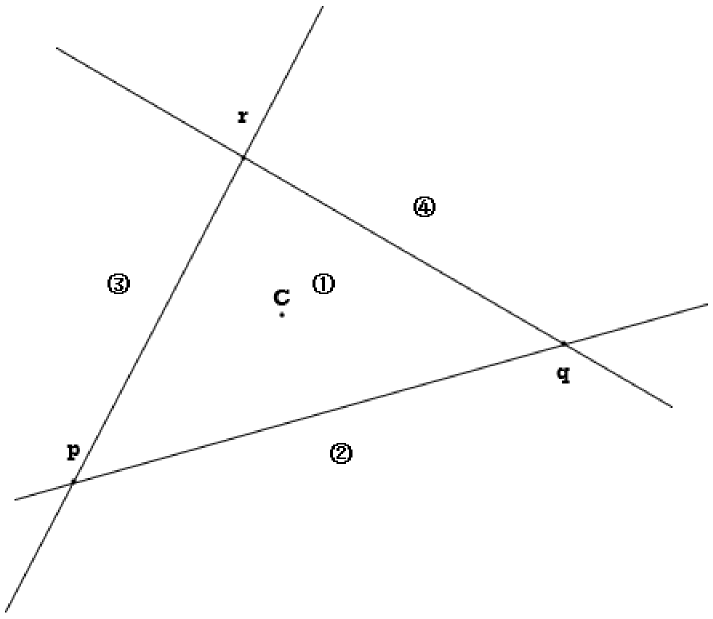


Figure 1: Four different regions in find closest point algorithm

- $\lambda + \mu \leq 1, \lambda > 0, \mu > 0$. Region 1 the projection of c lies in the triangle pqr. The nearest point is just point c as
$$c = p + \lambda(q - p) + \mu(r - p)$$
- $\mu \leq 0$. The projection of a is c which lies below l_{pq} in Region 2. λ is computed as:
$$\lambda = \frac{(c-p)(q-p)}{(q-p)(q-p)}$$
Then the closest point $c^* = p + \lambda_{seg}(q - p)$, where $\lambda_{seg} = max(0, min(\lambda, 1))$.
- $\lambda \leq 0$. The projection of a is c which lies in the left side of l_{rp} in Region 3. λ is computed as:
$$\lambda = \frac{(c-r)(p-r)}{(p-r)(p-r)}$$
Then the closest point $c^* = r + \lambda_{seg}(p - r)$, where $\lambda_{seg} = max(0, min(\lambda, 1))$.
- $\lambda + \mu \geq 1$. The projection of a is c which lies in the right side of l_{rq} in Region 3. λ is computed as:
$$\lambda = \frac{(c-q)(r-q)}{(r-q)(r-q)}$$
Then the closest point $c^* = q + \lambda_{seg}(r - q)$, where $\lambda_{seg} = max(0, min(\lambda, 1))$.

1.3 Simple search with bounding box

Simple search with a bounding box generally filters out some points based on the distance between the cuboid containing the triangle and the object point.

Specifically, for a given bound, we need to check xyz-coordinates separately for:

$$L_x - bound \leq c_{k,x} \leq U_x + boundL_y - bound \leq c_{k,y} \leq U_y + boundL_z - bound \leq c_{k,z} \leq U_z + bound$$

, then with triangle holds the above requirement, we compute the nearest point in the triangle and use the computed point to update bound. This method is generally a linear searching method with time complexity $O(N)$

1.4 Simple search with bounding sphere

Here we illustrate our method in constructing sphere bound for searching the nearest point. This is a two-step method which can be summarized as follows:

- Compute the centroid and radius for the sphere containing each triangle.

  First the longest side of the triangle$\vec{a}, \vec{b}, \vec{c}$ is founded, denoted $\vec{ab}$. Then the midpoint of $\vec{ab}$ $\vec{q}$ is computed to test if we hold the following inequality:
$$|\vec{c} - \vec{q}| \leq |\vec{a} - \vec{q}|$$
  which ensures the definition of radius, and $\vec{q}$ is centroid. Else we compute$\vec{f} = (\vec{a} + \vec{b})/2$ $\vec{u} = \vec{a} - \vec{f}$,$\vec{v} = \vec{c} - \vec{f}$ and $\vec{d} = \vec{u} \times \vec{v} \times \vec{u}$. A coefficient γ is computed as $\gamma = \frac{\vec{u}^2 - \vec{v}^2}{2\vec{d}(\vec{v} - \vec{u})}$. And
$$\vec{q} = \vec{f} + \lambda\vec{d}$$
  where $\lambda = \gamma$ if $\gamma > 0$ else $\lambda = 0$.
- Then we can compute the radius of each sphere as ρ. Based on the centroid $\vec{q}$ and radius ρ, a more directed judgement of distance between the triangle surface and object point can be made by comparing current bound and
$$||\vec{q} - \vec{c_k}|| - \rho$$
  , which is faster than the simple bounding search method as one condition is judged here.This method is generally a linear searching method with time complexity $O(N)$

1.5 KD-Tree based search

To further speed up the searching efficiency, we construct a KD-Tree structure by utilizing the centroid position found in previous sphere manner.

A brief introduction to KD-Tree is here. For our case, the xyz-coordinates of each point are used as features. Here is an iteration to construct the KN-Tree:

- Compute the variance of each feature among current sample set to obtain the feature with largest variance.
- Based on the selected feature, we find the median in this feature dimension and use it as the root. Then we split the sample set by putting samples with larger values in this feature to the right children and smaller ones in the left children.
- We then continuously repeat the above two steps until the size of each splitted sample set is one, i.e. no further children nodes.

With the constructed KD-Tree structure, the time complexity of searching is between $O(log_2N)$ and $O(N)$.

2 Algorithms steps

Here we introduce some of the main functions' algorithm realization steps as follows:

2.1 FindClosestPoint

FindClosestPoint function is used to find the nearest point within the given triangle region. All of the coordinate in this program is 3D.

Same as what we proposed in math approach, we first compute the coefficient λ, μ based on least square method. Then based on the range of λ and μ, we use four different criterion to work out the final center point.

When the coefficient lies in the intersection from two of four regions, the nearest point should be the vertex in the intersection boundary of them. So this can be searched by one of these two criterion.

2.2 simple search method

The bounding box and bounding sphere searching methods both work in a linear search manner, i.e. search on all of the meshes one after each other.As in 1 is the algorithm step for simple search method.

Algorithm 1 Designed framework simple search

```
Input: original $bound = 10^6$, the object point$a_k$, triangles
1: Initialize nearest point$b_k$ and distance in a relative large value
2: for $searching - triangle = 1$ to $N_{triangle}$ do
3:   if Bound Condition Satisfies then
4:     FindClosestPoint find Current-Nearest-Point$c_{k,searching-triangle}$ for searching-triangle with
       point
5:     if $|c_{k,searching-triangle} - a_k|$ ¡ $bound$ then
6:       $bound = |c_{k,searching-triangle} - a_k|$,$b_k = c_{k,searching-triangle}$
7:     end if
8:   end if
9: end for
Output: The found $b_k$
```

The bounding condition is used to filter out some points with large distance in advance to improve the efficiency of searching process. As we introduced in math approach part, for the bounding box manner, bounding condition is three judgements on xyz-coordinate and for the bounding sphere manner, the bounding condition is one judgement on the distance (between the object point a_k and the centroid of triangle) and the radius of the triangle.

2.3 KD-Tree construction and searching

The construction part is generally the same as in math approach.

For the searching part, with a KD-Tree structure, we can make the searching process from the root to the leaf nodes of the tree. For each node, we first check the distance between the node itself and searching point to determine if it can be the current best. Then for we construct a hyper-sphere whose centroid is the object point and radius is the current nearest distance. For each split action in tree, this can be viewed as a hyper-plane to segment the subtree into two parts, we then make a judgment to where there is intersection of the hyper-plane and hyper-sphere to decide if further searching is needed for this part. In this way, we can drop out the unrelated part onces without a linear search for the related point. If we can drop out half of the sample set once in each searching, then the time complexity of the searching strategy is $O(log_2 N)$. If each time we need a simple search for each leaf node in the tree, the time complexity is $O(N)$.

3 Overview of the program

The overview of total program is shown in Figure 2. We construct a 3-layer program where each layer is a python file for specific functions.

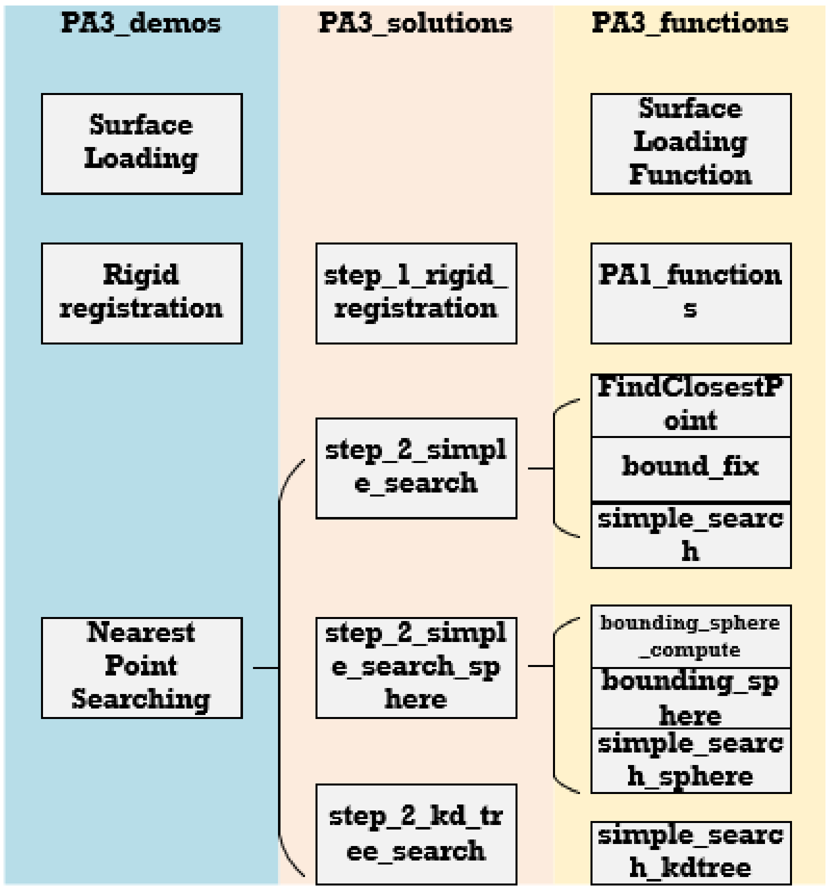


Figure 2: Overview of the program

In the PA-3demos.py, we construct the path to loading files including the surface mesh files and the positions of probe tips and the output path of the result. Then functions from PA3-solutions are used to finish the main task in PA3, which including first to work out the rigid registration for A_{tip} relative to B body and three different searching methods for searching the nearest points in surface.

PA3-functions includes the underlying functions to support the work of PA3-solutions:

Table 1: Error of our searching result

data set	error
a	0.244
b	0.528
c	0.468
d	0.623
e	0.376
f	0.050

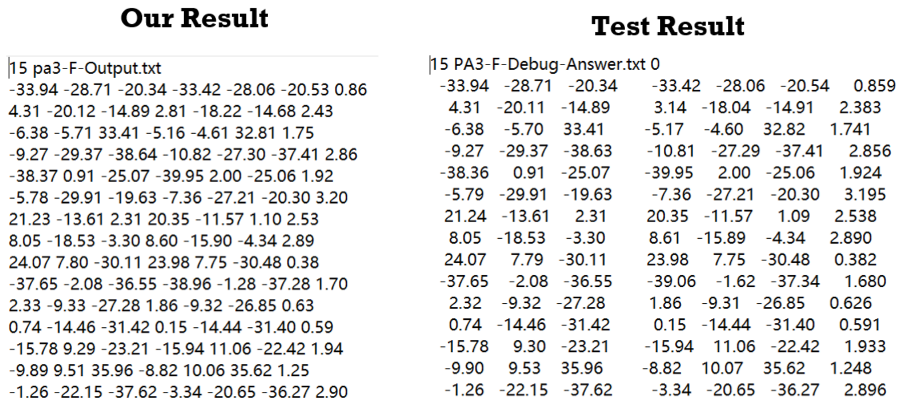
- Surface Loading Function is used to load the surface file and extracts the vertices coordinates and the triangles.
- We use some functions like frame rigid registration and frame transformation from our PA1-functions to realize the required frame transformation.
- We work out the used 3 different searching method based on a FindClosestPoint function, two different boundary construction methods for simple searches and KD-Tree construction. Then three different searching and update process functions are constructed.

4 Verification

In this part, we did two tasks to verify our program: In the first part, we compare our output result with the given debug data set. Then we made a comparison of the running time for simple searching with bounding box, with bounding sphere and KD-Tree structure.

4.1 Result on debug dataset

We first made the comparison between our obtained result and the standard debug files. All of the result comes from the KD-Tree based method for comparison. A direct result is given in Figure 3 for the debug data set F(which is we randomly selected).



```
15 pa3-F-Output.txt
-33.94 -28.71 -20.34 -33.42 -28.06 -20.53 0.86
4.31 -20.12 -14.89 2.81 -18.22 -14.68 2.43
-6.38 -5.71 33.41 -5.16 -4.61 32.81 1.75
-9.27 -29.37 -38.64 -10.82 -27.30 -37.41 2.86
-38.37 0.91 -25.07 -39.95 2.00 -25.06 1.92
-5.78 -29.91 -19.63 -7.36 -27.21 -20.30 3.20
21.23 -13.61 2.31 20.35 -11.57 1.10 2.53
8.05 -18.53 -3.30 8.60 -15.90 -4.34 2.89
24.07 7.80 -30.11 23.98 7.75 -30.48 0.38
-37.65 -2.08 -36.55 -38.96 -1.28 -37.28 1.70
2.33 -9.33 -27.28 1.86 -9.32 -26.85 0.63
0.74 -14.46 -31.42 0.15 -14.44 -31.40 0.59
-15.78 9.29 -23.21 -15.94 11.06 -22.42 1.94
-9.89 9.51 35.96 -8.82 10.06 35.62 1.25
-1.26 -22.15 -37.62 -3.34 -20.65 -36.27 2.90
```

```
15 PA3-F-Debug-Answer.txt 0
 -33.94  -28.71  -20.34    -33.42  -28.06  -20.54   0.859
   4.31  -20.11  -14.89      3.14  -18.04  -14.91   2.383
  -6.38   -5.70   33.41     -5.17   -4.60   32.82   1.741
  -9.27  -29.37  -38.63    -10.81  -27.29  -37.41   2.856
 -38.36    0.91  -25.07    -39.95    2.00  -25.06   1.924
  -5.79  -29.91  -19.63     -7.36  -27.21  -20.30   3.195
  21.24  -13.61    2.31     20.35  -11.57    1.09   2.538
   8.05  -18.53   -3.30      8.61  -15.89   -4.34   2.890
  24.07    7.79  -30.11     23.98    7.75  -30.48   0.382
 -37.65   -2.08  -36.55    -39.06   -1.62  -37.34   1.680
   2.32   -9.32  -27.28      1.86   -9.31  -26.85   0.626
   0.74  -14.46  -31.42      0.15  -14.44  -31.40   0.591
 -15.78    9.30  -23.21    -15.94   11.06  -22.42   1.933
  -9.90    9.53   35.96     -8.82   10.07   35.62   1.248
  -1.26  -22.15  -37.62     -3.34  -20.65  -36.27   2.896
```

Figure 3: Our Result and given correct result

And we also made a statistic of the produced distance and the standard result. We compute the error based on the square of all the distance error in each frame. The result is shown in 1. As we can see, the general error is acceptable. And when we compare the result we further find that some of these errors result from the saving formation, where we same as $.2f$ and the standard result is in $.3f$. When we switch to save 3 floats after the dot, the error comes to be in 10^{-3} magnitude.

Table 2: Operation time of 3 methods

data set	time consuming(s)
bounding simple	3.51
bounding sphere	4.61
KD-Tree	1.73

Table 3: Output Tabular for data set

Name	Description
NAME-answer.txt	required output file containing the coordinate and nearest point in surface
NAME-closest-point-coordinate.npy	the obtained closest point coordinate
centroid.npy	the position of centroid of each sphere
radius.npy	radius of each sphere

4.2 Running time comparison of 3 methods

In this part, we made a comparison of running time for three different methods. All of these results are obtained from the same PC with same conditions. We operate three methods from dataset A to F and the operation time is shown in Table 2.

We find that KD-Tree based method run fastest while the bounding sphere need longest time. For the KD-Tree method, as the time complexity for search is restricted no larger than $O(N)$, so the cost of time is quite acceptable. But for the unexpected longer time for bounding sphere operation, we guess that this may come from the computation of centroid and radius process, which is not included in the simple bounding box method. So this will lead to longer operation time.

5 Result on unknown data

Result of the output is saved in the output folder and the meaning of each file is given in 3. We also save some middle result for PA4 and the user can check more details of the program.

6 Contribution

Zhiyuan Ding works on the program construction and part of the report writing. Remus Li helps in the debugging process and part of the report writing.