# Programming Assignments 1 & 2 601.455 and 601/655 Fall 2021

**Please also indicate which section(s) you are in (one of each is OK)**

## Score Sheet

| | |
|---|---|
| Name 1 | Shuran Zhang |
| Email | srzhang@jhu.edu |
| Other contact information (optional) | |
| Name 2 | Qiyuan Wu |
| Email | qwu37@jhu.edu |
| Other contact information (optional) | |
| Signature (required) | I (we) have followed the rules in completing this assignment<br>Shuran Zhang<br>Qiyuan Wu |

| **Grade Factor** | | |
|---|---|---|
| Program (40) | | |
| Design and overall program structure | 20 | |
| Reusability and modularity | 10 | |
| Clarity of documentation and programming | 10 | |
| Results (20) | | |
| Correctness and completeness | 20 | |
| Report (40) | | |
| Description of formulation and algorithmic approach | 15 | |
| Overview of program | 10 | |
| Discussion of validation approach | 5 | |
| Discussion of results | 10 | |
| TOTAL | 100 | |

The mathematical approach

2021年11月6日 星期六 上午11:46

Step 1-3

$U_{s(i)} = \text{ScaletoBox}(C_i)$. to be specific $U_{s(i)} = \frac{C_i - \min\{C_i\}}{\max\{C_i\} - \min\{C_i\}}$

$B_{N,K}(v) = \binom{N}{K}(1-v)^{N-K} v^K$, where $\binom{N}{K} = \frac{N!}{K!(N-K)!}$

$B_{S,K}(v) = \binom{S}{K}(1-v)^{S-K} v^K$

For each point of $U_s$, we can get $F_{ijk}$ from different values of ijk.

$F_{ijk}(u_x, u_y, u_z) = B_{S,i}(u_x)\, B_{S,j}(u_y)\, B_{S,K}(u_z)$

$$\begin{bmatrix} \vdots & & \vdots \\ F_{000}(U_{s(i)}) & \cdots & F_{SSS}(U_{s(i)}) \\ \vdots & & \vdots \end{bmatrix} \begin{bmatrix} C^x_{000} & C^y_{000} & C^z_{000} \\ \vdots & \vdots & \vdots \\ C^x_{SSS} & C^y_{SSS} & C^z_{SSS} \end{bmatrix} = \begin{bmatrix} \vdots \\ C^T_{expected,s(i)} \\ \vdots \end{bmatrix}$$

(As we've already known the expected and computed $F_{ijk}$, we can compute the coefficient matrix by solving the least square problem.)

Use the above to calculate $\begin{bmatrix} C^x_{000} & C^y_{000} & C^z_{000} \\ \vdots & \vdots & \vdots \\ C^x_{SSS} & C^y_{SSS} & C^z_{SSS} \end{bmatrix}$

Then use this paramater matrix to construct Distortion Correction function

Then correct all $G_i$s

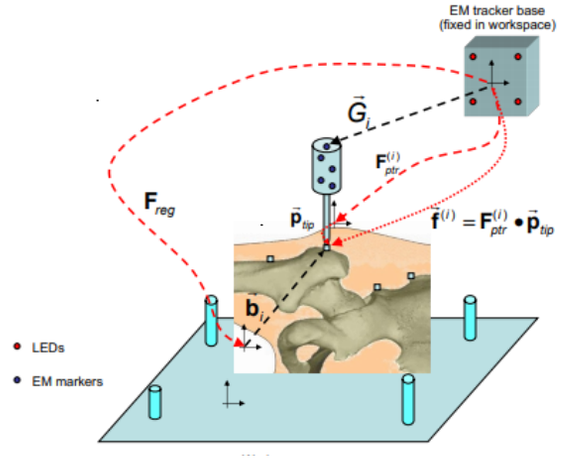
Step 4. use pivot calibration to get $\vec{P}_{tip}$:

$R_{ptr}[k]\,\vec{P}_{tip} - \vec{P}_{dimple} = -\vec{P}_{ptr}[k]$

use point cloud registration to get $F_{ptr}$ at fiducials

$F_{ptr} \cdot \vec{g}_j = \vec{G}_j$

calculate $f^{(i)} = F_{ptr}^{(i)} \cdot \vec{P}_{tip}$.



Step 5. $F_{reg} \cdot \vec{b}_j = \vec{f}^{(j)}$

$F_{reg}$ ↓ fixed

cloud to cloud registration of b and f to get $F_{reg}$.

Step 6. Calculate $F_{ptr}^{(i)}$ of navigation points using point cloud registration of g and G.

Compute $\vec{V}$ of navigation points

$$F_{reg} \cdot \vec{V}_i = F_{ptr}^{(i)} \cdot \vec{P}_{tip}$$

$$\vec{V}_i = F_{reg}^{-1} F_{ptr}^{(i)} \cdot \vec{P}_{tip}$$

- An overview of the structure of the computer program, sufficient to enable someone with reasonable skill (the grader) to understand your approach and follow your code.

**Header: XX.h**

**--CatesianMathBasic.h**

```
-- Vector3d crossp(Vector3da, Vector3db)

-- double dotp(Vector3da, Vector3db)

-- Matrix3d invR(Matrix3dR)

-- Matrix3d Rot3(Vector3d w, double theta)
```

**--F.h**

**Class F:**

--Class members: Rotation matrix R and translation vector p.

--Class functions: F operates F; F operates point; Inverse of F.

**--3Dpoint_sets_registration.h**

--F Point3D_Sets_Registration(vector<Vector3d>&a, vector<Vector3d>&b).

**--pivot.h**

--Function: VectorXdpivot(std::vector<F>&Fdata).

**--ScaleToBox.h**

--Function: `MatrixXd ScaleToBox(int N, int N_frames, int N_x, vector<Vector3d>& q)`

**--CorrectDistortion.h**

--Function: `int binomialCoefficients(int n, int k);`

--Function: `MatrixXd Fijk(int N, double ux, double uy, double uz);`

--Function: `vector<Vector3d> CorrectDistortion(int N, int N_frames, int N_x, vector<Vector3d>& q, MatrixXd c_matrix);`

**--PCalibrationPA1.h**

--Function: `VectorXd PCalibrationPA1(const std::vector<Vector3d> &G, const int N_G, const int N_frames, std::vector<Vector3d> &g);  //read data set of G`

**Library:**

**--<stdlib.h>**

**--<iostream>**

**--<fstream>**

**--<Eigen\Dense>**

**Program: XX.cpp**

**-- CatesianMathBasic.cpp**

-- crossp: Compute cross product of two vevtors. Use the function in Eigen.

input: 2 vectors

output: cross product of these two vectors

--dotp: Compute dot product of two vevtors. Use the function in Eigen.

input: 2 vectors

output: inner product of these two vectors

--invR: Compute inverse matrix of the rotation matrix R, which is equal to the transpose of matrix R. Use the function in Eigen.

input: 1 rotation matrix

output: inverse of the input

--Rot3: Build the rotation matrix R using Rodrigues's formula given rotation axis and angle.

input: rotation axis, angle of rotating

output: corresponding rotation matrix

**--3Dpoint_sets_registration.cpp**

--Input: 3D point set a & 3D point set b

--Output: Frame from coordinate system a to b.

--Process:

- Compute the average value of 3D point set a and b.
- Compute new value of centralized points in a and b.
- Direct techniques due to K.Arun to solve for R to solve for R.
  1. Compute H.
  2. Compute the SVD of $H = USV^T$.
  3. Compute $R = VU^T$.
  4. Verify det(R) = 1.
- Output desired transformation frame F.

**--pivot.cpp**

--Input: Frame vector.

--Output: p as [$p_t$;$p_{pivot}$].

--Process:

- Dynamic size assignment to combined Jacobian matrix and combined RST.
- Constructing combined Jacobian matrix (3i*6).
- Constructing combined RST (3i*1).
- QR decomposition approach to solve the least square problem A*p=B, where A=$J_{fc}$, B=$p_c$.

**-- ScaleToBox.cpp**

--Input: Degree of Bernstein polynomials, the number of frames, the number of data in each frame, dataset needs to be scaled.

--Output: Scaled dataset.

--Process:

- Put all the data in the dataset into a matrix.
- Set a specific maximum and a minimum value.
- Scale the data in the data set using the equation:$us = \frac{qs-min}{max-min}$.

**-- CorrectDistortion.cpp**

--binomialCoefficients:

Input: Degree of Bernstein polynomials, integer number.

Output: binomialCoefficients:

--Fijk:

Input: Degree of Bernstein polynomials, ux, uy, uz.

Output: F value in the F_us matrix.

--CorrectDistortion:

Input: Degree of Bernstein polynomials, the number of frames, the number of data in each frame, dataset needs to be corrected, coefficient matrix.

Output: Corrected dataset.

Process:

- Scale the dataset to be corrected into the scale which is the same as the matrix used to compute the coefficient matrix.
- Construct a "tensor form" interpolation polynomial.
- Compute corrected dataset and output as a vector.

**-- PCalibrationPA1.cpp**

--Input: dataset vector, the number of data in each frame, the number of frames, points dataset.

--Output: pivot calibration dataset vector.

--Process:

- Compute average position of frames.
- Compute g.
- ComputeF_G[k]: Call 3Dpoint_sets_registration function.
- Compute p_G: Call pivot function.

**--Main program: PA1.cpp**

--Input: Data files.

--Goals: Finish Q4-Q6.

--Outputs: Data computed by the program in the output file.

--Process:

- Define problem set variables.
- Define read-in aiding temporaries.
- Finish Question 1 and get Cexpected.
- Finish Question 2 and get coefficient matrix for correction function.
- Finish Question 4:
  1. Read files.
  2. Compute a frame F_D: Call 3Dpoint_sets_registration function.
  3. Compute a frame F_A: Call 3Dpoint_sets_registration function.
  4. Compute a C_expected: Call Class functions in F.
- Finish Question 5:
  1. Read files.
  2. Compute average position of frames.
  3. Compute g.
  4. ComputeF_G[k]: Call 3Dpoint_sets_registration function.
  5. Compute p_G: Call pivot function.
- Finish Question 6:
  1. Compute average position of frames.
  2. Compute h.
  3. Compute F[k]: Call 3Dpoint_sets_registration function.
  4. Compute p_H: Call pivot function.
  5. Output required data in an output file.

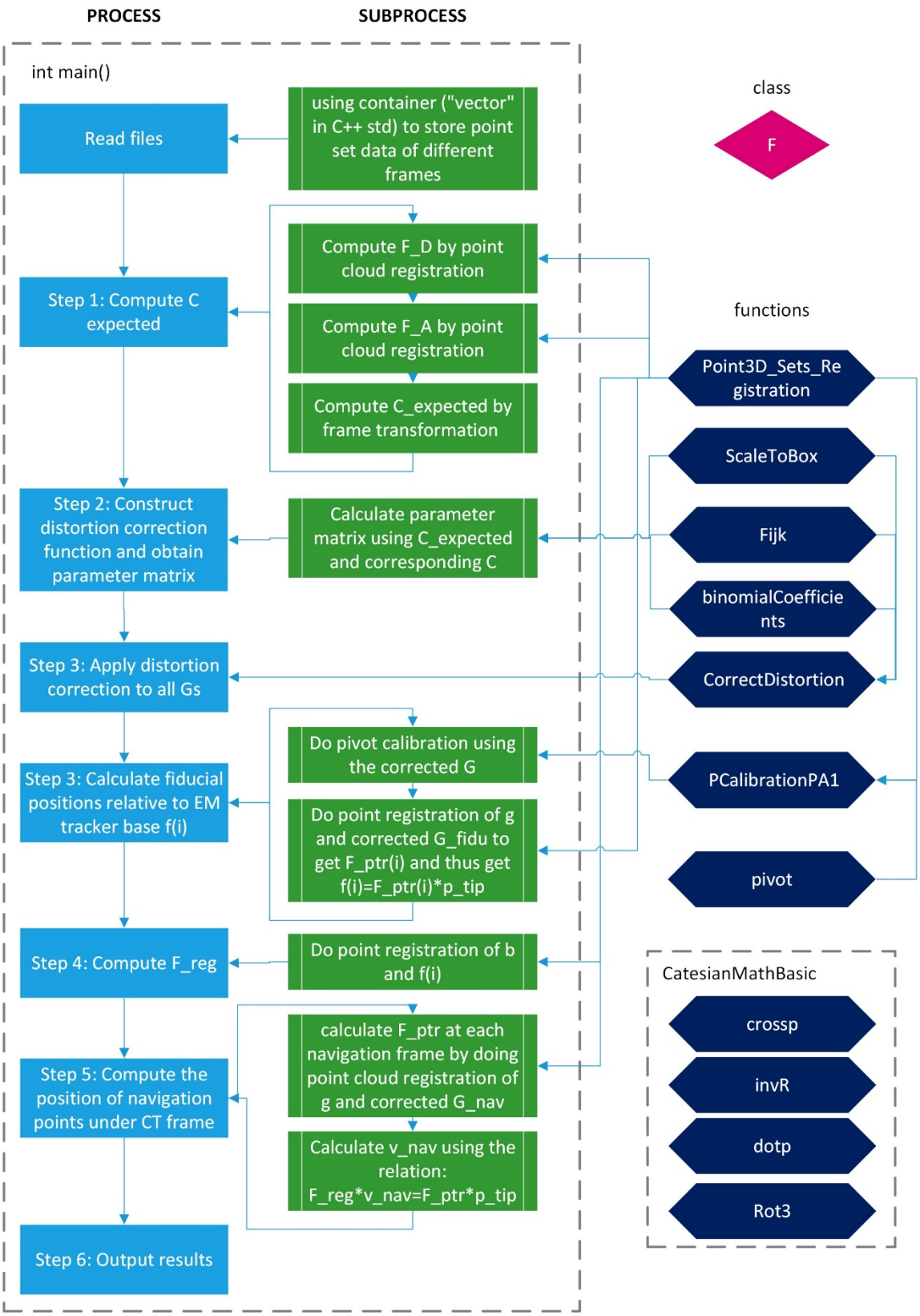



Figure 1 Flowchart of the main program

- The steps taken to verify that the program is working correctly. Typically, this would take the form of a discussion of the results using the debugging examples.

1. Generating special values to test if a module is designed correct.
   For example: For example: When testing the 3D point cloud to point cloud registration module, we used the first step of pivot calibration (calculating g from G at frame 0) to verify. Just do point cloud registration for g and G at frame 0 and get F_ptr, the translation term p_ptr of frame transformation F_ptr should be exactly the same as G_0 (the averaged Gs), and the rotation part should be identity matrix I. This is how we found our problem in point cloud registration.
2. We wrote some "cout"s at several important points of our programs, mainly to check if the intermediate result and size of containers are correct.
3. Input the debug data from a to f to get their computed output file separately.
4. Compare the output data we get from our code with that of the given data roughly by looking through these two files. If data in those two files is similar, move on to the next step, or check the program. Take debug file a as an example.
5. Run the test program to compare the output data we get from our code with that of the given data and compute the average error. The equation of the average error is: $Ave\_Error = \frac{abs(OUTPUT\_computed - output\_given)}{The\ number\ of\ values}$.
6. If the average error is lower than 1, our code is verified to work correctly. For example, table 1 shows the average error of all the given debug datasets.

Table 1 Average Error of all the debug datasets

| Filename | pa2-debug-a | pa2-debug-b | pa2-debug-c | pa2-debug-d | pa2-debug-e | pa2-debug-f |
|---|---|---|---|---|---|---|
| Ave_Error | 0.01135 | 0.35625 | 0.0263 | 0.012225 | 0.491 | 0.37045 |

Table 2 Output analysis of debug dataset a

| Filename | | SZ&QW-pa2-debug-a-OUTPUT-2.txt | pa2-debug-a-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 148.327, 118.974, 43.1999 | 148.33, 118.97, 43.20 |
| | *2* | 76.2589, 116.056, 152.642 | 76.26, 116.06, 152.64 |
| | *3* | 36.9013, 148.371, 170.974 | 36.90, 148.38, 170.97 |
| | *4* | 75.8039, 75.615, 172.003 | 75.80, 75.62, 172.00 |

This set of debug data has no distortion, noise, and OT jiggle, and the result is exactly the same as the reference data (the exactly same at 2 digits after the decimal point if retained all numbers to 2 digits). This shows that the frame transformation process, point cloud to cloud registration process and pivot calibration process does not introduce significant numerical errors.

Table 3 Output analysis of debug dataset b

| Filename | | SZ&QW-pa2-debug-b-OUTPUT-2.txt | pa2-debug-b-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 89.3715, 83.9368, 170.621 | 89.36, 84.06, 170.89 |
| | *2* | 39.6497, 163.553, 157.442 | 39.49, 163.50, 157.59 |
| | *3* | 117.986, 156.328, 81.9247 | 117.80, 156.28, 81.82 |
| | *4* | 35.0265, 36.1994, 134.441 | 35.05, 36.12, 134.66 |

This dataset is with EM noise, and a slight mismatch with the reference data is observed at the digits after the decimal point. Seems like the noise brings irregular and unpredictable performance to the algorithms, but its impact is small.

Table 4 Output analysis of debug dataset c

| Filename | | SZ&QW-pa2-debug-c-OUTPUT-2.txt | pa2-debug-c-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 91.5743, 88.2207, 161.128 | 91.58, 88.22, 161.13 |
| | *2* | 45.8406, 130.707, 99.2817 | 45.84, 130.70, 99.31 |
| | *3* | 127.681, 96.2753, 171.757 | 127.69, 96.27, 171.75 |
| | *4* | 106.53, 64.0532, 39.3964 | 106.51, 64.04, 39.39 |

This dataset is with EM distortion, and a slight mismatch with the reference data is observed at the digits after the decimal point. One possible reason is that some of the parameters (‘c’s in the parameter matrix) may get very small or very large, so there might be some numerical errors when calculating the parameter matrix and using this parameter matrix to correct distortions.

Table 5 Output analysis of debug dataset d

| Filename | | SZ&QW-pa2-debug-d-OUTPUT-2.txt | pa2-debug-d-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 125.325, 148.454, 50.1994 | 125.32, 148.45, 50.20 |
| | *2* | 105.366, 79.7545, 97.6994 | 105.36, 79.75, 97.70 |
| | *3* | 162.481, 32.465, 47.7453 | 162.48, 32.46, 47.75 |
| | *4* | 47.1159, 138.057, 27.1186 | 47.11, 138.06, 27.11 |

This dataset is with OT jiggle, and quite a well match between our results and the reference data was observed. This shows the frame transformation algorithms and frame correspondence is correct and works well.

Table 6 Output analysis of debug dataset e

| Filename | | SZ&QW-pa2-debug-e-OUTPUT-2.txt | pa2-debug-e-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 162.886, 76.06, 117.443 | 162.54, 75.83, 117.41 |
| | *2* | 86.1508, 79.7563, 84.8745 | 85.95, 79.76, 84.70 |

| | | | |
|---|---|---|---|
| | *3* | 46.5077, 169.317, 99.2204 | 46.12, 169.36, 99.29 |
| | *4* | 66.5063, 45.9172, 74.9008 | 66.75, 45.67, 74.91 |

Table 7 Output analysis of debug dataset f

| Filename | | SZ&QW-pa2-debug-f-OUTPUT-2.txt | pa2-debug-f-output2.txt |
|---|---|---|---|
| $N_{frames\_nav}$ | | 4 | 4 |
| $v_{nav}$ | *1* | 116.238, 32.9379, 139.569 | 116.28, 32.70, 139.46 |
| | *2* | 41.6053, 171.661, 54.2944 | 41.21, 171.90, 54.37 |
| | *3* | 40.9841, 60.9562, 58.878 | 40.91, 60.94, 58.90 |
| | *4* | 65.6395, 133.483, 90.0458 | 65.44, 133.55, 90.05 |

In summary, these datasets provide scenarios with EM distortion, EM noise and OT jiggle separately and combined, which enable us to analysis where the errors come from. One source of error is from out distortion correction process, in specific the error possibly to be the numerical error of very large and small floating-point numbers. But the magnitude of this error is acceptable, since the correction process already eliminated the relatively large-scale distortion. The second source of error possibly comes from the EM noise in the input data. Since the noise is purely irregular, distortion correction process has no effects on reducing the noise.

- A tabular summary of the results obtained for unknown data

| Filename | | SZ&QW-pa2-unknown-g-OUTPUT-2 | SZ&QW-pa2-unknown-h-OUTPUT-2 |
|---|---|---|---|
| Nframes_nav | | 4 | 4 |
| $v_{nav}$ | *1* | 111.836, 75.797, 148.56 | 116.28, 32.70, 139.46 |
| | *2* | 85.5754, 143.098, 75.2556 | 41.21, 171.90, 54.37 |
| | *3* | 76.5622, 47.6537, 92.4541 | 40.91, 60.94, 58.90 |
| | *4* | 65.8896, 80.7997, 26.3894 | 65.44, 133.55, 90.05 |
| Filename | | SZ&QW-pa2-unknown-i-OUTPUT-2 | SZ&QW-pa2-unknown-j-OUTPUT-2 |
| Nframes_nav | | 4 | 4 |
| $v_{nav}$ | *1* | 63.7137, 91.5561, 109.047 | 143.084, 113.925, 59.8739 |
| | *2* | 106.397, 165.703, 42.2703 | 96.9621, 34.0297, 114.633 |
| | *3* | 73.8169, 145.778, 122.516 | 133.897, 41.2057, 109.094 |
| | *4* | 50.0857, 87.7432, 88.8339 | 126.29, 46.9108, 117.798 |

- A short discussion for the results of running your program. This certainly includes the tabular summary above, but may also include a discussion of convergence if you adopt an iterative process or of difficulties if you suspect that your answer is wrong.

1. Discussion about problems we met at ScaleToBox:
   In the beginning, we set the minimum value and the maximum value which were going to be used in scaling the dataset as the minimum value and the maximum value of the dataset. But when we used the coefficient matrix to correct other datasets, we found that the results were far away from the correct one. And then we thought it may happen

because the dataset to be corrected has different scales with the dataset used to compute the coefficient matrix. Hence we set the minimum value and the maximum value to fixed values so that we can project different datasets into the same scale. And the results showed that this method is right.

2. Another mistake we've made is about the integral/floating number multiplication and dividing process when calculating average G. Actually when multiplying type(int) and type(double), we should put the type(int) variable in sequel to the variable of type(double), to make sure we get a type(double) result.

- A short statement of who did what.

1. Both: Analyze the scenario and figure out the Pseudo-code of each part of the assignment. Debug and modify the whole code. Discuss the result and analyze the reasons. Write the report.
2. Qiyuan Wu: Question 3-6 of the programming.
3. Shuran Zhang: Question 1-2 of the programming.

OnedriveLink of the Zip file: