Software Engineering Project:

Final Report

Group members:

Antoine Merlet

Gülnur Ungan

Mladen Rakic

Marcio Aloisio Bezerra Cavalcanti Rockenbach

Professors: Dr. Yohan Fougerolle, Dr. Cansen Jiang, Dr. David Strubel

January 7, 2018

Contents

I. INTRODUCTION / CONTEXT

This project is the main assignment of the Software Engineering course, which belongs to the first semester of the University of Burgundy Masters programs MAIA, ViBOT and MsCV. The course is taught and mentored mainly by professor Yohan Fougerolle, accompanied by Mr David Strubel and Mr Cansen Jiang. Our group is made out of four members, namely Gulnur Ungan, Antoine Merlet, Marcio Rockenbach and Mladen Rakic. All of our group members are from the MAIA Program, each of us with different backgrounds. That versatility and the individuality of all the team members were the key ingredients in making this project a reality.

The proposal of the project was to work on improvements of the previous year projects regarding the development of a software that scans a person using an external sensor (Kinect v2) and captures and displays a 3D representation of the scanned model. Although the previous year's students achieved relatively acceptable results, there are still many implementation flaws and much room for improvement. The codes and the reports were difficult to read and understand, the implementation was lacking some common programming practice and the installation process was quite challenging. Throughout this report, we will discuss the goals and main targets for improvement decided by the group, ways of achieving them and main challenges faced, the results we accomplished and propositions on what can be done in the future development.

II. CHOICES / TARGET

In terms of defining the main goal we want to achieve, we analyzed the target through various aspects. We believe that these points are very important when it comes to general improvement of the project and to better understanding of it for the next generation of students. Namely, these aspects can be formulated as follows:

- We decided to make the user interface in a way that is more readable and understandable (i.e. more user-friendly). The way to accomplish this is to create GUI based on forms instead of creating the blank GUI and then implementing separate elements manually. We argue that this way makes it easier to manage, modify and improve individual components of the GUI in comparison to the manual implementation.

- Organization of the code is another major point we wanted developed, meaning that the code, which is split into several modules, is more suitable for corrections and overall understanding. Our implementation should also be, for the most part, non-Qt dependant and also non-PCL dependant, which is a significant improvement, since it allows someone to easily switch to implementation using OpenCV, for example.

- Interfacing is another thing we want to advance, in a sense that the user interface is supposed to be very simple and minimalistic in design, yet very powerful, offering a wide range of options and methods for the user to choose from. Those methods, being very advanced, still should not add to the complexity of the interface usage. User manual is a particularity which ought to make things easily comprehensible.

- One of the most important milestones we wanted to reach is a good documentation of the project. This can be obtained through several instances - well commented code; logger class which helps a lot with most of the debugging processes and also adds to the overall user-friendliness; functions that are separately and precisely documented; overall good coding habits, such as reasonable naming of the functions, classes and variables and consistency throughout all the files; the manuals and the diagrams to make the project easily and relatively quickly readable and understandable; etc.

- We want to offer a wide palette of various filtering and registering options to choose from, rather than to stick to ICP only. We also aim to allow a reasonably effortless and straightforward refinement of the parameters, a crucial thing for the following generation that will possibly build upon our project.

- Considering the amount of the workload and the functionalities we focused on, it remains to be said that our main target was not to improve the meshing part of the project. Regarding that, we argue that having provided the good interfacing, methods for filtering and registration, debugging tools and overall organization and documentation of the code, it is significantly less demanding to build upon the current state of the project and provide a good meshing algorithm than it would be in any other scenario.

All the bullet points stated above serve to justify one of the most important decisions we made, which is to actually start the whole project from scratch. We firmly believe that, taking the current state of the previous year's projects, it is a wise choice to accept and keep some of the things we think are fairly well developed, and incorporate them intact or modified into our project, but all the time sticking to our goals and the way of organizing it.

III. PROJECT MANAGEMENT

To implement the difficult goal of carrying out this project, we decided to split it in different tasks. The idea was to assign the tasks in such a way that they are as much independent from each other as possible, making the overall management and individual development smooth. The rough assignment of the objectives is the following:

- Antoine: overall project design
- Marcio: input and output of the data
- Gülnur: filtering and registration techniques
- Mladen: GUI

Even though the assignments appear very individual and distinct from each other, we had to have good organization and communication skills in order to link all the dots properly and put the whole project into practice.

We created a repository for our group in that can be found in GitHub (https://github.com/AntoineMerlet/3DScan), in which we kept all the source codes and other important files. We had weekly group meetings to discuss everyone's progress and to address difficulties encountered, as well as to further develop the future assignments. Those meetings were always documented in our repository. Also, everyone had a personal weekly report, where each of us made a description on what we had worked on and the future tasks. More detailed description of each individual task will be provided in the upcoming sections.

A. Antoine

B. Marcio

1. *Logger*

One of my main goals was to create a logger class, responsible for creating and managing a log, which registers all the main important events that occur when a software is running. Through that tool, it is much easier to debug the code. That being said, it represents a significant improvement in comparison with previous year's projects. We can insert log messages in the chosen key points in the source code and then analyse its behaviour through the log.

In our software, I implemented the log in a singleton design, which means that it restricts the instantiation of a class to a single object, created in the main window. This pattern is useful in our case because we want the same logger to be called by all the windows and classes in our program.

The logger then operates through a pointer, which represents the only instance of the class, as stated before. First, for every session of the program, a .txt file is created in the build directory. The name of the file is as follows: Log + CurrentDate + CurrentTime + .txt. After the initiation of the logger, we also connect the pointer to the main window through a QTextEdit using Qt signals / slots system. This widget is constantly updated to display all the messages sent to the logger (figure 1).

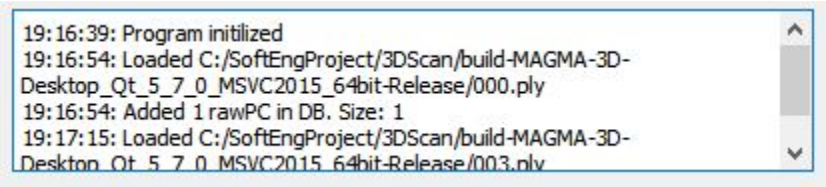

FIG. 1: Logger widget in the main window

I also implemented the logger in a way that is easy to be used everywhere in the code. This was done by defining a macro. By simply typing LOG(– string message here –); in any part

of the code, we can send a message to be included in the log file. Although this is not the most secure way to implement it(since it is not type safe), it adds a lot to easiness of usage. This class definitely can and should be improved by next year students, but it already is a functioning tool for project debugging.

2. I/O Tools

My second goal was to handle most of the input and output section of our project. For that, I had to work with the Kinect sensor and to implement the necessary code to obtain and save data from the Kinect and to display the point clouds in a visualizer.

The Kinect for Windows version 2.0 was released in the end of 2013 / beginning of 2014. The sensor is composed of an RGB camera (3 channels) that stores data in a 1280 x 960 resolution, an infrared emitter and an infrared depth sensor, which allows it to capture a depth image with resolution of 512 x 424. It also features a microphone and an accelerometer. It has a depth range between 0.4 - 4.5 m and has a field of view 70^{o} horizontally and 60^{o} vertically.

Point cloud library (PCL) has some modules for data input that are called grabbers, but it does not have one specific for the Kinect. Although it is not implemented in the PCL itself, previous year projects made use of a grabber for the Kinect that is available on the internet (http://unanancyowen.com/en/pcl-kinectv2-with-grabber/) and that is implemented in only one header / cpp file. We decided to keep this method of data acquisition by using this class.

After implementing and testing the Kinect acquisition using the kinect2grabber, it was necessary to connect it to our project. I worked with Mladen in the development of the GUI for that purpose, by creating a PCL Visualizer to display the live scan. For the acquisition of the point clouds, I implemented a callback function that interacts with the keyboard. By pressing the key "s", one point cloud is stored in the system. The visualizer of the live scan also features a bounding box, whose coordinates can be determined through sliders in the scan window.

I also worked on the loading and display of the point clouds in the main window (figure 2). When the user selects the files to be loaded, a QListView widget is updated to display the

corresponding filenames. By interacting with this widget (by checking or unchecking the items), the display widget (a QVTK widget) is updated to show the selected point clouds.

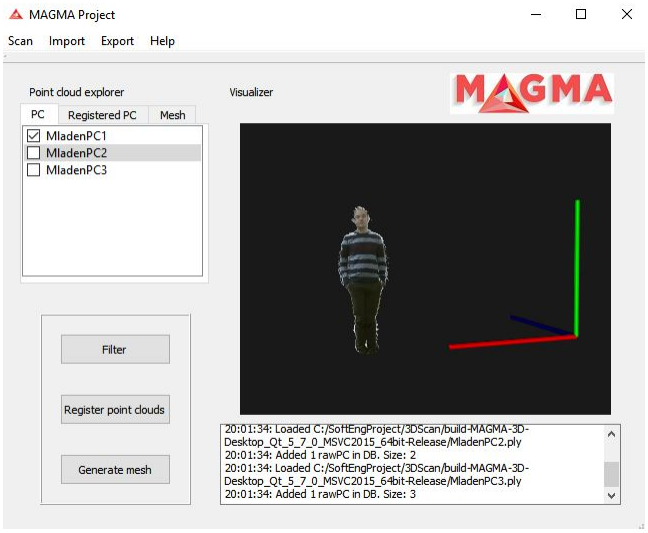

FIG. 2: Display of point clouds

Since the input and output are closely related to interaction with the user, it was a good opportunity for me to also learn a lot about GUI design in Qt and how to transmit information between different modules of the software.

C. Gülnur

D. Mladen

One of the main reasons that made me tackle this particular part of the project was to actually get familiar with the concepts of creating the user interface and to get a better understanding of the way components are organized and connected. Also, the ability to design things in an appealing, yet simplistic and minimalistic way, came in handy when it came to this. Having studied the previous year's designs of the interface, I found numerous flaws in both the aesthetics and the implementation. Therefore, the proposed GUI, even though it inherited some minor concepts from those, is for the most part built from scratch and developed in a way to suit the particular needs

of our proposal.

The principal concept around which the whole design process revolved is to make the interface as much user-friendly as possible, while keeping it powerful, in a sense that it is able to provide the user with the wide variety of methods and parameters choices. The way I implemented this was to create several independent windows, each serving a particular purpose, while maintaining the user not overwhelmed with the amount of content.

Another salient aspect was to implement the overall connectivity between the GUI and other classes in a way that makes them as much independent from each other as possible. This is a powerful way to make the code easily modifiable and it will be further elaborated in this section. Although one might argue that some of it might not be the most elegant or the most common programming practice, we have to bear in mind that this way puts the project in a state that is very understandable and readable, which is very convenient for the next generation of students.

The user interface is consisted of a number of key components. These can be listed as follows:

- Main window, which gives user the possibility to interact with the major functionalities, such and importing and exporting of the point clouds, filtering and registering the data, generating the mesh, as well as the access to the user manual and the about section. Another particularity which makes the experience more user-friendly is the logger, implemented by Marcio, which helps to keep the track of the ongoing and completed processes, and serves as a great tool for the debugging.

- Scan window, used to perform live scans directly from the Kinect sensor. Other than that, it enables easy manipulation of the bounding box on the live scan which should bound the sensor's field of view. It also allows the easy capturing of the raw point clouds and has a variety of options for further development, such as the choice of performing horizontal or vertical acquisition. This, however, will be further developed in the Future tasks section.

- Filter window, which is a dialog box for an easy selection of the filters and the corresponding parameters.

- Register window, which is yet another dialog box that offers a variety of the registration methods and choosing of their respective parameters.
- About section, a pop-up window with the basic information regarding the project, team members, and mentors.

To further elaborate the overall design and implementation, and to justify some of the choices I made, I will make a brief insight on some key concepts. Main window (figure 3), however simple it may seem, offers an access to all the parts of the project very easily. It allows the user to load or save the point clouds and to display them on the QVTK Widget. It is possible to choose from a list of the loaded point clouds and to display multiple point clouds at once. The list of the point clouds with the checkboxes makes the whole process very intuitive and reasonable.

Clicking on a new scan option opens up the scan window (figure 4), while at the same time hiding the main window, which is a redundant component when it comes to the new acquisition. Scan window manages the manipulation of the bounding box and the capturing of raw point clouds from live feed from the sensor. Stopping the scan and closing the scan window make the main window appear again, with all the finished processes stated in the logger.

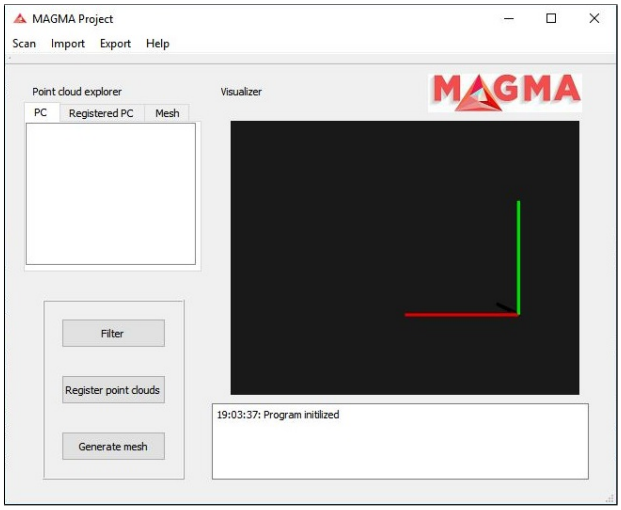

FIG. 3: Main Window

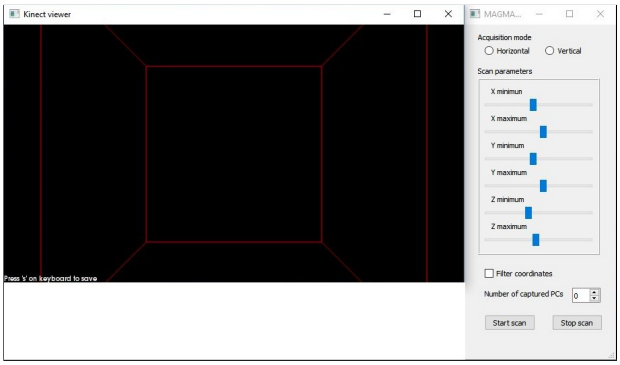

FIG. 4: Scan Window

As previously stated, some of the choices I made may not seem reasonable or elegant, and that is regarding the filtering and the registration windows (figures 5 and 6, respectively). Namely,

both of those windows contain numerous methods for data processing and the corresponding parameters which can be chosen. All the storage of the data is handled by the structures that contain them. For instance, parameters for median filter are stored into the following structure:

```
struct median {
bool checked = false;
int windowsize;
int maxmovement;
  } medianfilt;
```

Member *checked* serves to store the information whether the corresponding checkbox on the filter window has been checked or not. Clicking on the Filter button will trigger a function that will check which boxes have been selected by the user, then switch the *checked* member of the corresponding structure to true and then and only then allow the storing of the parameters for that particular structure. That way, when the filtering parameters are chosen and the window has been closed, the appropriate function that performs the filtering will be able to accept the parameters only from the structures that have the member *checked* set to be true. This implementation makes the user interface fairly independent from the mathematical tools (filtering and registration functions) and makes it easy to potentially modify the parameters and the used methods themselves. I can argue that this way is well suited for the refinement and further improvement of the data processing. It certainly made the project management simple in a sense that big parts we all individually worked on, although very connected and dependent when considering the project as a whole, can still be partially independently developed.

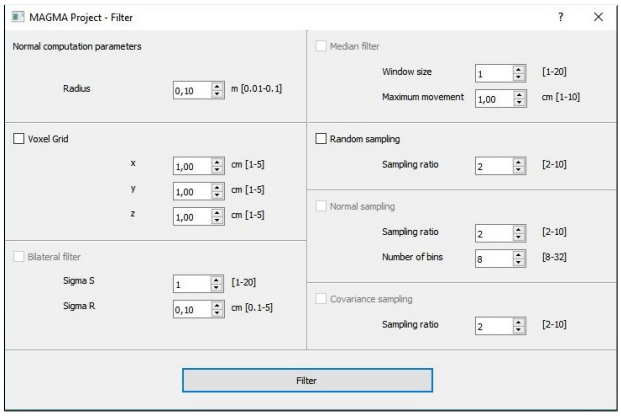

FIG. 5: Filter Window

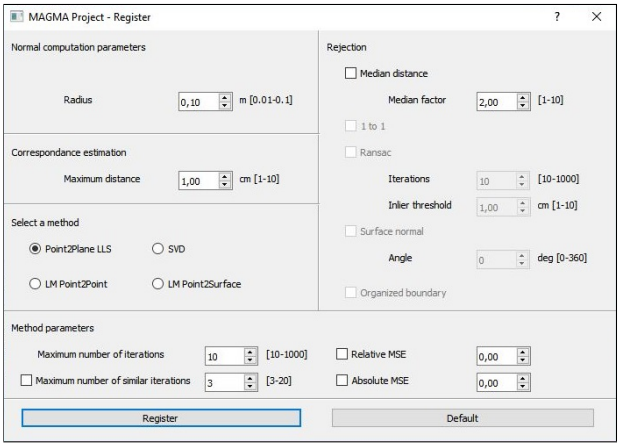

FIG. 6: Register Window

IV. CHALLENGES

One of the main challenges to start to work on this project is the simple installation and configuration of the source code. Since it requires different libraries which need to be correctly installed and configured both in the user's system and in the project files, it is a very time consuming task. It takes a lot of effort to download all of the required tools and to make them work. Because of that, we needed weeks just to be able to run the previous year projects and to start working on improvements.

Other important challenge is to learn the "Qt ways". Qt is a very powerful tool, but requires a lot of time and studying to learn and to be comfortable with its syntax and methods. For example, Qt has an option to implement GUI using forms and it also uses a signals / slots system to make it easier to communicate between different objects in the software. Those are great features, but the majority of our group was not familiar with that before the project started.

Finally, one of the most significant challenges is the level of complexity of the project. It is a huge leap between small console oriented projects using only basic C++ syntax and STL that we did in the labs and in class to a elaborate project involving user interface, usage of different and complex libraries and interaction with an external sensor.

V. CONCLUSION

VI. FUTURE WORK

VII. REFERENCES

1. http://unanancyowen.com/en/pcl-kinectv2-with-grabber/

2. https://github.com/UnaNancyOwen/KinectGrabber/tree/Kinect2Grabber/Sample

3. https://cppcodetips.wordpress.com/2014/01/02/a-simple-logger-class-in-c/

4. http://pointclouds.org/

5. https://www.vtk.org/

6. http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1

7. https://github.com/umaatgithub/3D-KORN

8. https://github.com/WajahatAkhtar/Project-S.E