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 8, 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

1. Introduction

Regarding the group members' skillset and background, we decided that I would be in charge of the team leading / software design. Thanks to my IT background, I have basic knowledge regarding software design, software engineering, UML, C++ and reverse engineering. In the early stage of the project, I was able to share my knowledge about Kinect-based 3D scanning softwares. In fact, I work on a similar project, where data of four different Kinect were to be acquired, denoised, filtered and registered. I was in charge of the data processing, by implementing a software in C++, without external libraries such as PCL or OpenCV. By comparing the current project and my past experience, one of the main differences would be the usage of a single camera with a rotating object for the first one, and several cameras with a static object for the second. The second difference is the use of external libraries, in our current case the use of PCL. Using such a powerful library has, of course, a lot of advantages, such as the implementation of multiple state-of-the-art algorithms, with an optimized implementation. But it also has some downsides. In software engineering companies, few libraries are used, but very often. Therefore, software developers become used to these libraries, and using them becomes as simple as using STL data-types. In our case, I never used PCL, and my other group members were just getting introduced to C++. In that regard, learning to use PCL was a real challenge for all of us. Finally, this semester's project is not "new". It is the continuation of last year's MAIA-VIBOT-MSCV project, with this year, as a goal, to bring improvement to those previous projects. My second task was to define the features to improve from the past projects. This point has been detailed in the previous section.

2. Libraries installation

Certainly, and for every group, one of the major issues of this project. As it has been discussed a lot with our teachers, and as every group might write a lot on this topic, I will directly go to the improvements we bring on this matter. I decided to provide, through a Google Drive (https://drive.google.com/open?id=1MmQ-hRlzqGR1E3cuMcJVyMzfS9TeXooE) , all the necessary installation executables, pre-compiled QVTK, and a bash file for automatic environment variable update (all this for Windows 10, 64 bits). The folder is for now empty, and will be filled once I will have access to a good internet connection, as several GB of data are to be uploaded. This installation part took dozens of hours to be solved, due to the lack of training regarding CMake, Pro files and libraries linking. By making all the resources available at the same place, with an installation guide, will save next year students a lot of time. Once I managed to install the libraries/drivers on my machine, I installed (with more or less success) it on my groupmates computers. I would like to thank again Roger Pi Roig for providing me with QVTK precompiled binaries, as I was totally unable to do it on my computer. With the projects installed, we could start to read previous years code (in a better way than just with notepad++).

3. Software design

Reading someone else's code can be difficult. Understanding it can be a challenge. Modifying it is madness, except if proper software design tools are used. And UML (Unified Modeling Language) is one of those tools. Some previous groups gave it a try, but in a really basic fashion. For this time, I decided to spend a substantial amount of time on designing the software long before coding. I first created four packages, representing the main sections needed in our code:

- Core: namely, the core of the project. It should contain all the mathematical tools needed to process the point clouds (See GULNUR - MATHEMATICS). It should be completely independent from the GUI and Qt functions in general, so that the mathematical tools could be used under another IDE.
- GUI: In this folder belong all the forms, headers and source files which deal with interfacing the software (See MLADEN- GUI). It should contain as least PCL functions as possible.
- IO: Contains the logger class, the sensors handlers (See MARCIO - I/O) and file management functions. This last part is not mandatory by itself, as PCL is equipped with I/O module. I chose to still interface the PCL functions, in order to have feedback on the errors, and also

to provide an interface, in case next year students would like to OpenCV based software.

- Storage: Certainly the simplest package. It contains only a Database class, to store our loaded point clouds.

All our C++ files are divided into folders with the same names as the packages. With this very simple first draft of the project, I started to write UML diagrams (all available on our github). For the duration of the project, these diagrams have been modified, updated, deleted and created. A final and general UML diagram will be given in the coming days. I am sure that, even with a single UML lesson, newcomers to software development will be able to save a lot of time by reading our diagrams.

4. Coding habits

UML diagrams are important, but code appearance is even more valuable. With honesty, it was not enjoyable to read last year codes: variables have meaningless names/prefixes/suffixes, too long names, the code is not always indented properly (go to line between each function parameter), documentation and comments are minimal. In that regard, I tried to make use of my experience to give my teammates good coding style/habits. In general, we tried to stick to basics of C++ coding standards, give short but meaningful names to our variables/function, comment tricky parts of the code and, most important of all, document our code using Doxygen tags. All these "small" technicalities make it easy (hopefully) to read, understand and appreciate our work. But the main aim here is to help as much as possible the following groups for the study of our code, and try to make it immune to the DELETE keyword.

5. Handyman

Regarding the actual writing of code, I did myself IO/iotools and Storage/Database. This task was not so long, and I chose, instead of taking some work from my beloved colleagues, to be the handyman. I did some small and quickly written functions that are making life easier for my groupmates. I also spent time helping them to debug, implement tricky things according to our level. Overall, my colleagues fully managed the implementation part, as I was simply helping to find ideas to move forward in their work.

6. ICP

This part of the implementation is originally Gulnur's part. But the workload on it, as well as the challenges coming with it, made me decide to spend time helping her on this part. I re-implemented the correspondence functions, as I saw later that the data-types were not matching with the ones for ICP. Last year, most of the teams chose to hard code some values for ICP and filters. This year, as the time was too short to find perfect values (or ranges), the choice of user-inputted values has been made. The user is able to set all the parameters for all the mathematical functions used by using the appropriate window in the GUI. This will allow next year students to have an easy time trying out several configurations (for filter/correspondence/transformations features selection and parameters choice). For now, our filters are working (most of them), correspondences rejection works like a charm, but our "homemade" ICPs are not working. Therefore, we chose to implement, via the Default button in registration parameters, a simple built-in PCL ICP with default parameters, which is currently able to perfectly register up to 4 point clouds in less than 30 seconds.

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).

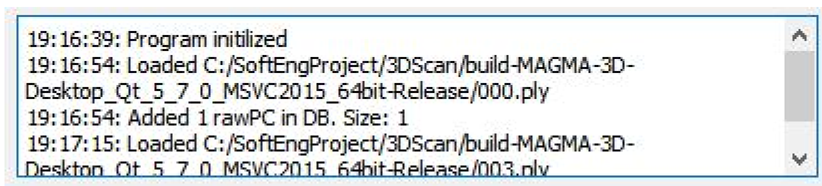

FIG. 1: Logger widget in the main window

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 70º horizontally and 60º 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.

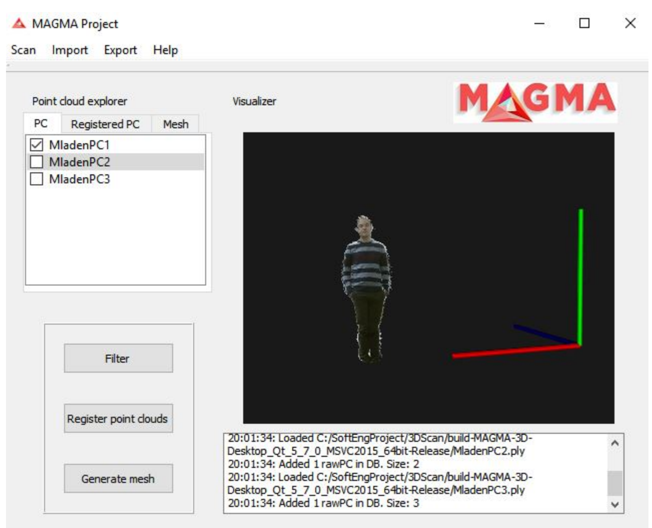

FIG. 2: Display of point clouds

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.

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

3D scanning has a bunch of problems that have been focusing on for a long time. These can be mentioned as follows: optically uncooperative materials, scanning in the presence of occlusions,insuring safety for delicate objects,scanning large objects at high resolution, accurate scanning in the field,handling large datasets,filling holes in dense polygon models and so on. To handle these problems, theory of 3D scanning can be divided into two main parts. These are registration and filtering. Registration part is split into 4 steps to be understandable:

1. **Selection**:The sampling of the input point clouds.
2. **Matching**:Estimating the correspondences between the points in the subsampled point clouds.
3. **Rejection**: Filtering the correspondences to reduce the number of outliers.
4. **Alignment**: Assigning an error metric, and minimizing it to find the optimal transformation.

For registration algorithm, it is chosen according to important parameters such as accuracy,computational complexity, convergence rate and number of data. Registration algorithms are two types: Feature based and ICP(Iterative closest point) based. These two different approach for registration methods are used for different priorities. The major advantages of using the feature-based methods over the ICP-based methods are:

1. The registration process is independent of the initial alignment of PCs.

2. It is not necessary to search for all points in PCs to find the corresponding pairs. Thus, redundant or irrelevant points such as outliers, or points that do not have correspondences have no direct effect on the registration.

The main disadvantages can be said like :

1. Point clouds must have distinct features

2. In general, the feature-based methods are slower than the ICP-based methods

Last year projects focused on ICP aigorithms. First of all, we tried to find a good feature based method to implement by searching so many papers. Then, we decided to go back ICP algorithms because of time complexity of feature based algorithms and complexity of these algorithms. Difference between ICP and Feature based algorithms can be shown below figure. Note that, path 1 is for feature-based registration algorithms and path 2 is for ICP algorithms [Marani , Reno et. a]

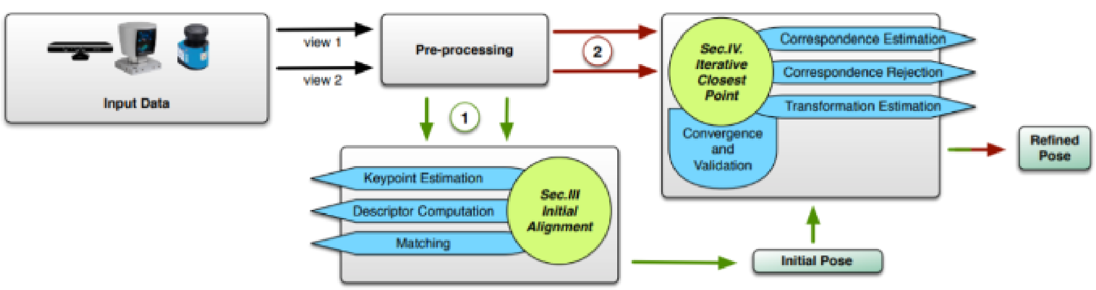

FIG. 3: Difference between ICP and feature based aigorithms

Filtering part of our project comprise of 3 subtitles. These are noise removal,sampling and normal estimation.

1. **Noise Removal**:It can be defined as the process of removing noise from a signal. To remove noise we used as follows:

(a) **Median Filter**:It is one of the simplest filter for 3D point cloud registration. In our project median filter has two parameters. Window size is used for setting the window size of the

filter and maximum movement is limitation of moving while filtering is processing.

(b) **Bilateral Filter**: We implemented bilateral filter for smoothing depth between two point clouds. Bilateral filter has 2 parameters which are called Sigma S and Sigma R. Sigma S is used for tuning size of the Gaussian bilateral fi

lter window to use and Sigma R is used for tuning standard deviation.

2. **Sampling**: Sampling techniques reduce time for registration point clouds.

   (a) **Random Sampling**: We applly random sampling with uniform probability.This sampling is based off Algorithm A from the paper "Faster Methods for Random Sampling" by Jeffrey Scott Vitter. For random sampling, the only parameter is sampling ratio. Sampling ratio is used for making registration fast. In our project, it can be tuned between 2 and 10.

   (b) **Voxel Grid**: It is used to downsample the given point cloud. Its parameters are called x,y and z. These are used for size of the filter.

   (c) **Normal Space Sampling**: It selects randomly (uniform) samples from each bin. Transformation is locked better when normal space sampling is used. It has 2 parameters. These are sampling ratio and number of bins.

   (d) **Covariance Sampling**: It selects the points such that the resulting cloud is as stable as possible for being registered (against a copy of itself) with ICP.

3. **Normal estimation**: We used normal estimation for estimating local surface properties at each point in point cloud.

Since we used Iterative closest point algorithm in our project, registration window first ask normal computation parameters. For normal computation, it is needed to be known maximum depth chance and smoothing size. Maximum depth chance is defined as threshold for computing object borders based on depth changes. Smoothing size is the factor for tuning area for smoothing surfaces. Correspondence represents a match between two entities (e.g., points, descriptors, etc). This is represesented via the indices of a source point and a target point, and the distance between them. In this project we focused on estimation and rejection of correspondence.

1. **Correspondence Estimation**: Correspondence estimation is the process of pairing points pi from the source point cloud P to their closest neighbors qjin the target cloud Q. This is a greedy approximation of finding the ideal correspondences (pi , qj) between the two clouds. [Holz et al.]. In this project, maximum distance is used for tuning correspondence estimation as parameter.

2. **Correspondence Rejection**: Rejection is also another issue for 3D point cloud registration. To achieve our goal, we set the parameters as called median distance, 1 to 1, RANSAC surface normal and organized boundary. They are cascaded in previous sentence order. Briefly, we would like to explain aiming of these parameters. Median distance is used for tuning distance between two different correspondence and Ransac is used to neglect false correspondences icp algorithm.

3. **Transform Estimators** For PCL, ICP is equal to SVD. For improvement of our project, we have added 4 different methods for estimating as follow:

   (a) **SVD**: Singular value decomposition method can be called as basic ICP in our project.

   (b) **LLS P2S**: It is called linear least square point to surface. In each iteration of ICP, there is always point to plane error. Linear least square optimization is added for removing ICP point to surface approach handicap. [Low]

   (c) **LM P2P**: Levenberg Marquardt point to point method. Basically, it can be said that Levenberg Marquardt algorithm is a version of non linear least square problems. Note that, previous option is capable of using linear least square optimization. This method is a combination of gradient descent method and the Gauss Newton method. [Gavin]

   (d) **LM P2S**: Levenberg Marquardt point to surface method. Since Levenberg Marquardt is used for non linear optimizing problems, for ICP point to surface method, it is good to use LM P2S for non linear objects.

In conclusion, I (Gulnur Ungan) chose above mentioned filters for our project. Previous year projects put just one filter. I would like to put more than one filter to see basically differences between filters and each of filter has different priority as its duty. This means that, MAGMA can show different views of filtering. I implemented filters in our project. For ICP, since its complexity, Antoine Merlet helped me for implementing all those 4 diffferent ICP types.

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 4), 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 5), 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.

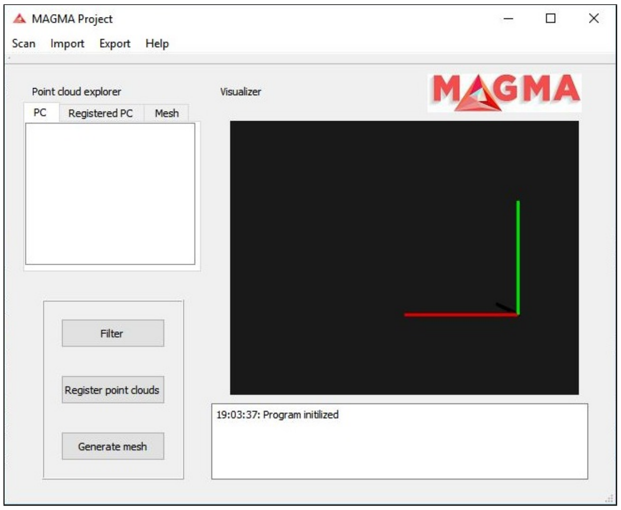

FIG. 4: Main Window

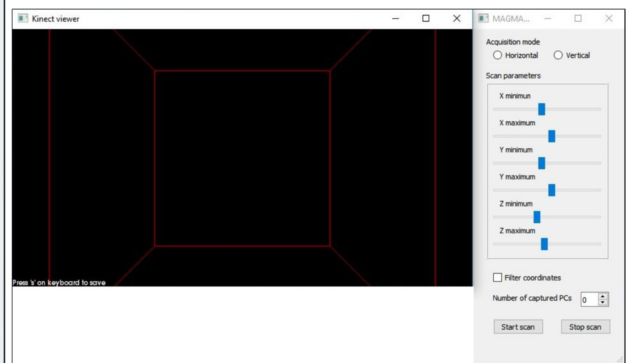

FIG. 5: Scan Window

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 6 and 7, 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.

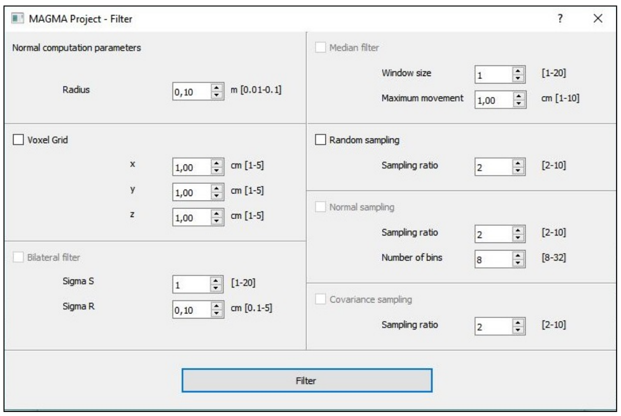

FIG. 6: Filter Window

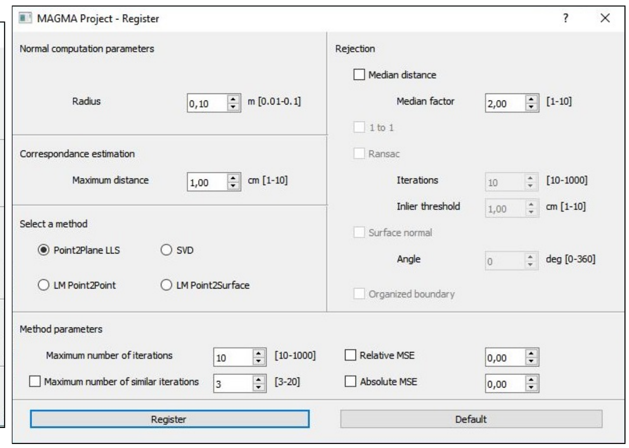

FIG. 7: Register Window

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

Taking into consideration all the bullet points stated in Target section, as well as the elaborations of the individual and the group work on the project, we can conclude that we managed to reach the goal that was set. We can justify this statement by making a brief resume of the accomplished objectives. We did manage to build a GUI based on forms instead of a manual implementation, which makes it way more understandable and easier to manipulate. Also, we strongly argue that our code organization is well executed, in a sense that it is well organized and structured, documented and commented, with all the necessary diagrams, manuals and debugging tools to make the user experience pleasant. The code, and the project as a whole is quite consistent and undemanding to follow - this makes the reverse engineering for the next generation a reasonably fair task.

We did not, however, implement the meshing part, but the current state of the project is a fairly good platform to build upon. Parameter refinement is easily achievable, different modules of the project are as independent as possible, which makes the software understandable and improvable. Another good argument to support the idea of the project being a good platform to build upon is the fact that the mathematical tools implemented offer a wide variety of methods for filtering and registering and one might decide on the optimal ones fairly easy, and develop on that.

To conclude, even though the project was started from the scratch, and even though our skillset and the given amount of time are arguably insufficient, the current stage the project is at is very noteworthy and remarkable.

VI. FUTURE WORK

For the next generation of students who will work on this project, we propose some improvements that can be done on the software:

- Logger: implementation without using macro, making it typesafe
- std::cout and Qdebug redirection to the logger, for easy printing of matrices for example
- Implement meshing techniques of the registered point clouds
- Horizontal / vertical acquisition modes: allows the user to choose between vertical and horizontal orientation of the Kinect sensor.
- Save captured files in other extension than ".PLY"
- Generate configuration files for settings, be able to load them as well
- Full implementation of the software as Qt and PCL independent
- Use the viewports of the display, to show the base and result with colored distance metrics
- For nw, the software cannot really be expanded full screen. It would be great, reagarding user experience, to implement full screen mode

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