Chapter 4: Technical Implementation Walkthrough

1 Kinect Cameras Calibration

Even the most expensive cameras have lenses that brings a level of deviations to the acquired images. Calibration is about mathematically relating camera measurements to the real world's measurement. The relation between pixels and physical measurements (e.g. meters) is critical for 3D scenes reconstruction in the way we can obtain a model of the camera's geometry and a model of the lens' distortion. These two models define the intrinsic parameters of the camera, correct for lens distortions and interpret the whole geometry of the physical scene.

For this project, our goal was to come out with a dedicated module that provides the user with the calibration parameters of a connected Kinect sensor. Unfortunately, this module was not implemented due to the limited time even though enough research on the matter was done. This can be achieved using OpenCV but for the time being we were contended with existing programs that does the same job. For this, we would like to refer to the GML Camera Calibration Toolbox and the Mobile Robot Programming Toolkit which provide very satisfactory results.

One thing to add is that the intrinsic parameters of the Kinect (at least the Kinect v2) are added by the manufacturers and Kinect SDK knows about them and is making use of them when needed. Therefore, they are not explicitly used in the data acquisition module but will be needed for the registration module as will be discussed later.

2 Data Acquisition (class SE3D::Recorder)

As obvious as it may seem, we need a module for acquiring frames from the Kinect sensor. This has been achieved by using the PCL OpenNI grabber interface as well as the Kinect SDK's libraries since the PCL's does not support the Kinect v2 through its current grabber. As a result, we are able to save color and depth frames from the RGB camera and the IR camera respectively. However, the two sensors are at an offset which means they are not capturing the exact same scene. In fact, they even have different spatial resolutions with the RGB camera taking frames at 1920 x 1080 and the IR sensor at 512 x 424.

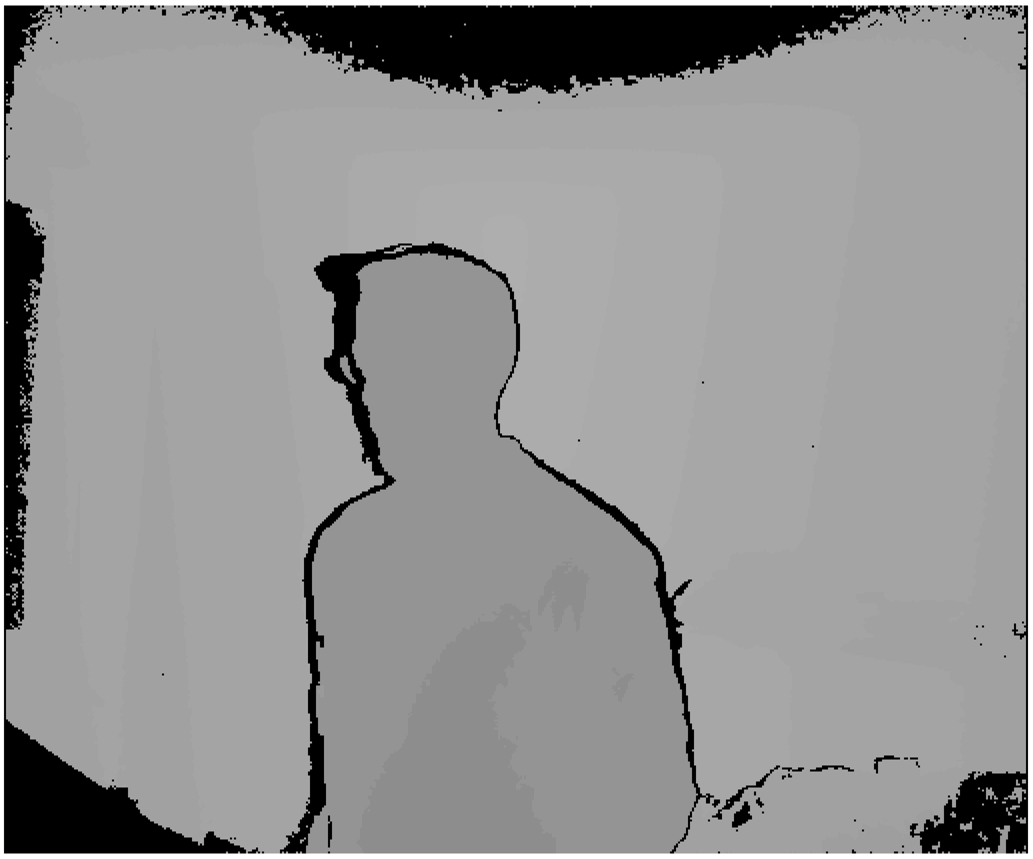
This means the mapping from depth space to color and camera spaces is inherently a non-trivial task. This would imply reducing the resolution of the color frame by down-sampling and then aligning with the depth frame keeping in mind that calibration need to be done first. Luckily, we leveraged some of the multiple useful APIs from the Kinect SDK to handle the transformations for us.

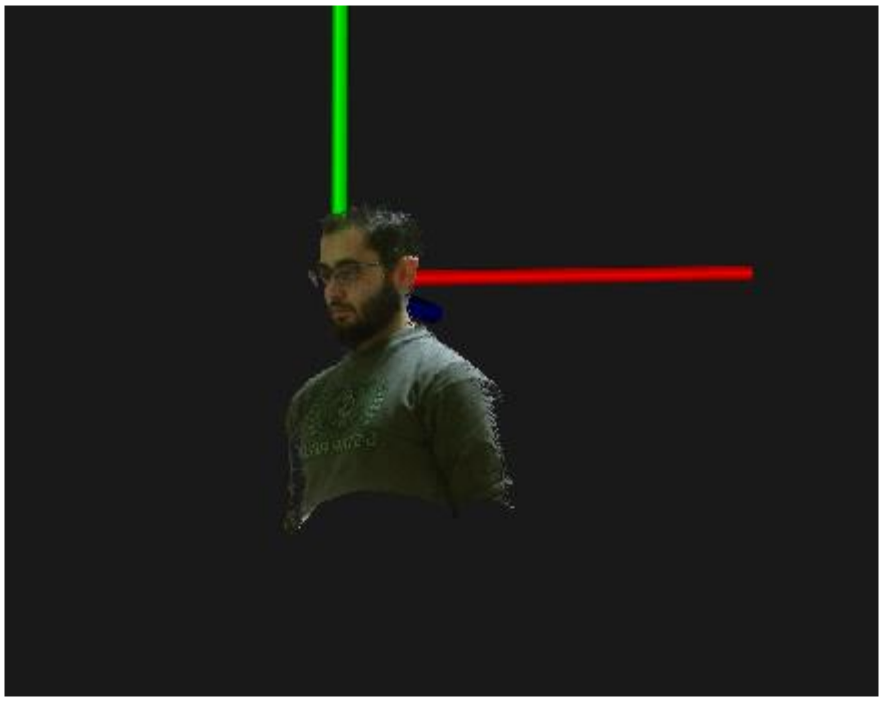
The color and depth frames are saved as PNG files at the resolution of 512 x 424.

In addition, this module can render and save a colored point cloud of the frame.

Finally, it was made such that it only captures within a range helping by that to remove any background or unwanted regions. This means that only the model, if standing at the correct distance from the sensor, is captured without any extra filtering or work as shown in the below generated files.

color frame (.png)


depth frame (.png)

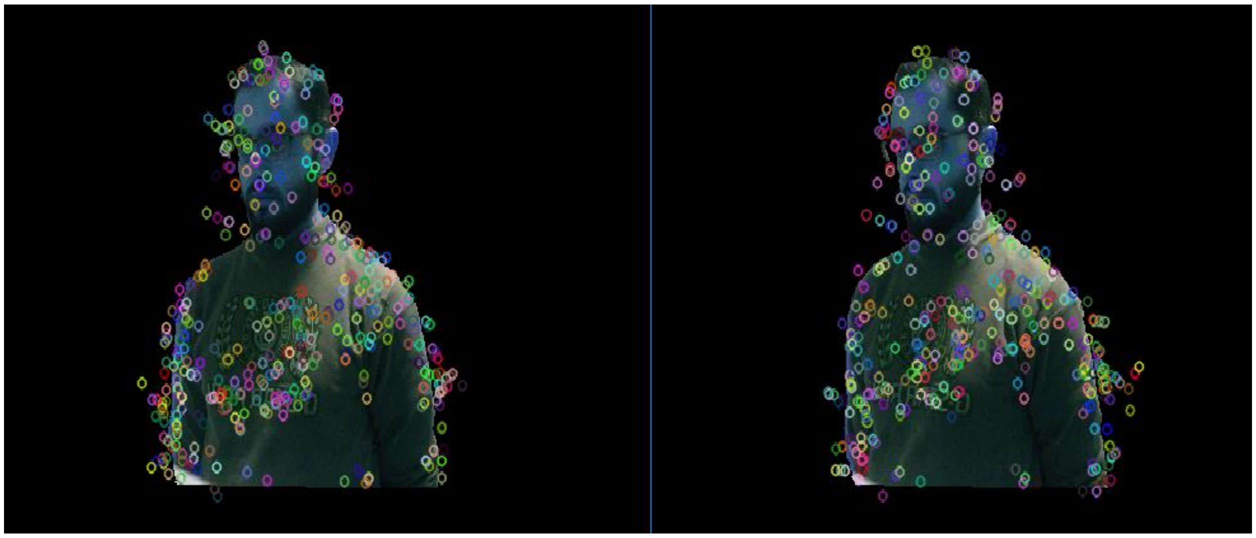

point cloud (.pcd)

3 Model Registration (SE3D::Reconstructor)

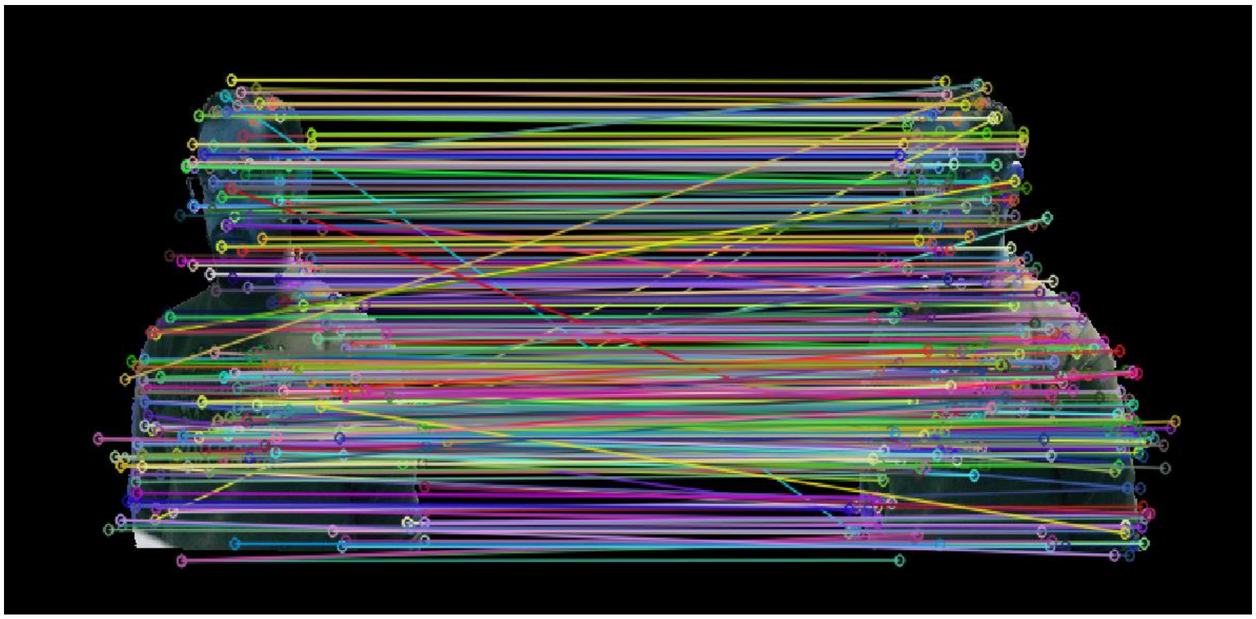
This module deals with the rendering of a complete 360° model through the iterative alignment of multiple 3D point clouds each representing a different view. The approach was to compute the relative positions among the different views while setting a unique coordinate system for reference to optimize the overlap of the model's parts that are common in two or more clouds.

We will first describe how we went about reconstructing from two views and then with multiple ones.

The pipeline starts with feature detection and description using OpenCV's implementation of the SURF algorithm. The SURF worked well and fast especially that we are dealing with thousands of points from each Kinect frame and hence the correspondence estimation is exponential. From two view images, the key points are extracted, and the descriptors computed (feature vectors). The descriptor vectors are then matched with FLANN (the brute force approach was tested but it was good enough with FLANN and subsequent filtering).

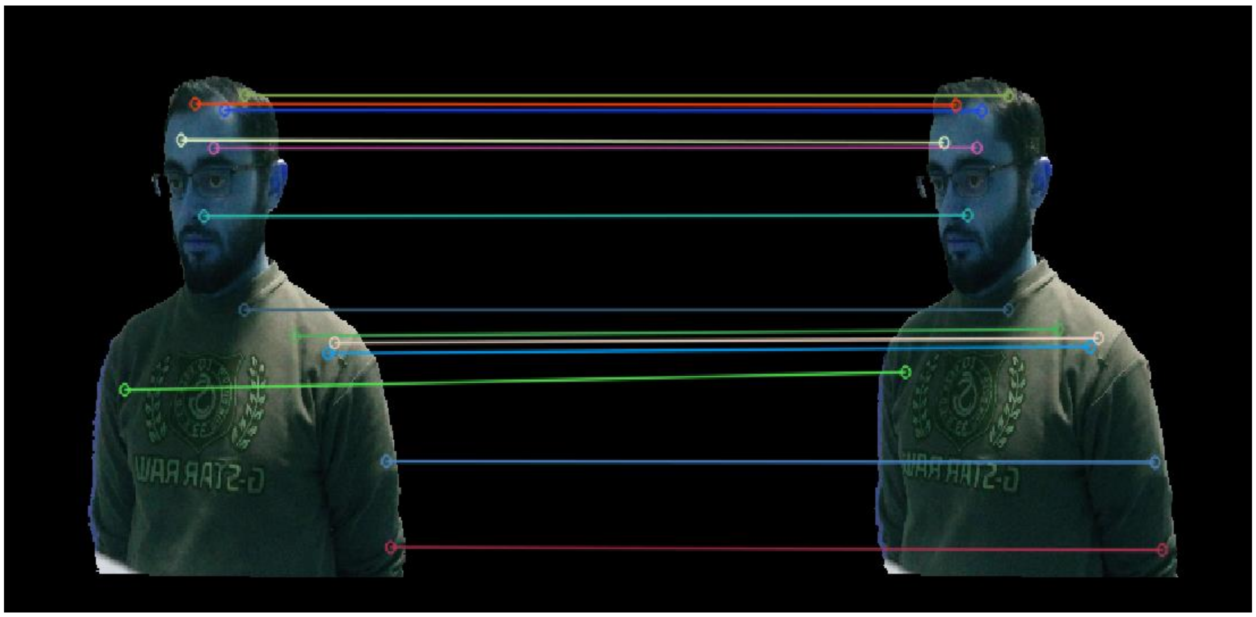


Key points detected on 2 different views

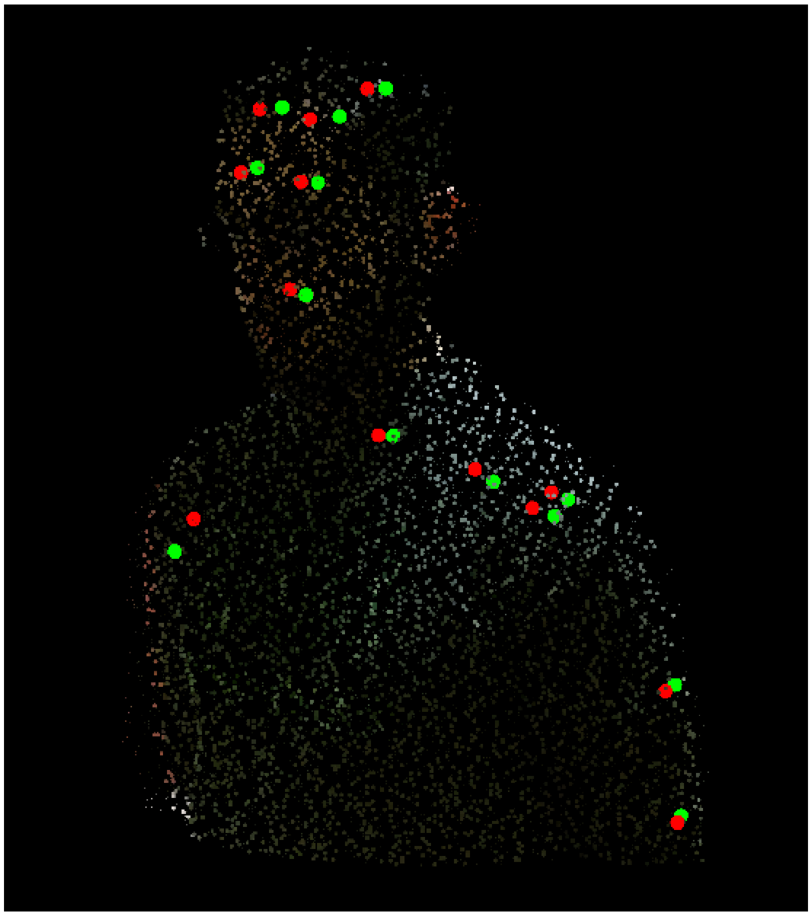


All key points matched (notice some erroneous matches)

The matches have been filtered to only those with good estimation relying on the distances. Another good filtering mechanism we came out with is to limit the matches to evolve within the model only and not outside (which obviously has no region of interest).

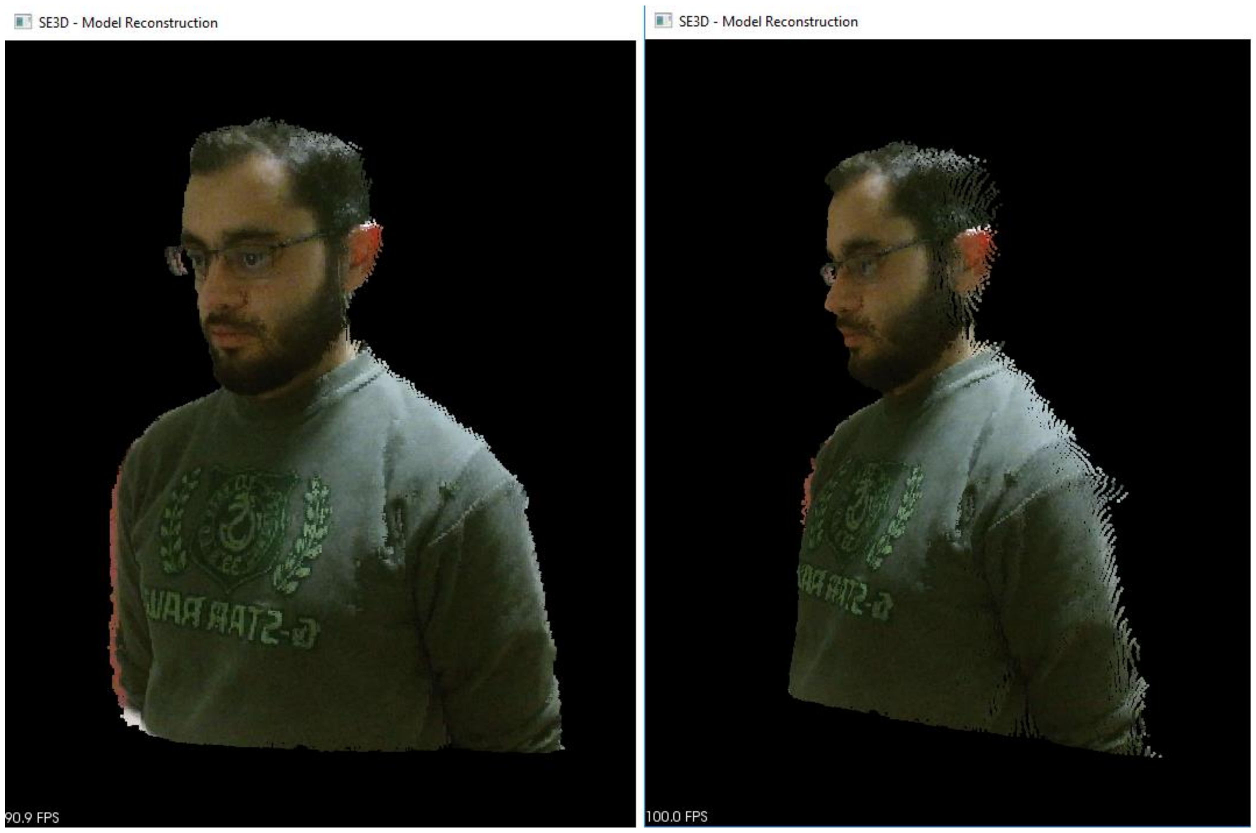


The matching after filtering



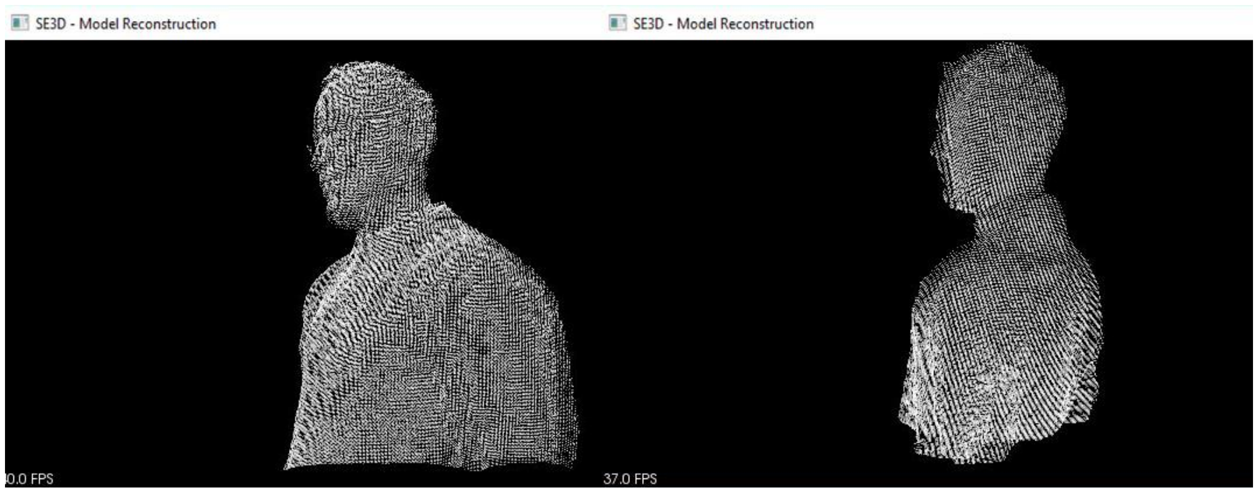
Visualizing features differences between 2 clouds

Next is to perform a registration from the 2 views after rendering them as points clouds (reference and sample) along with their corresponding features. Yet, some additional purging of the correspondences is done using the RANSAC algorithm. Then, the transformation between the two features clouds is estimated based on SVD thanks to the PCL's ICP interface. The transformation matrix is then applied to the sample cloud before being fused with the reference cloud leading to the alignment.



Result of 3D registration from 2 clouds (the model has been rotated left and up in the second screenshot to demonstrate the 3D effect)

Now that we could register 2 clouds, we can incrementally register, in pairs of two, other clouds representing more views. A general transformation is computed to transform all clouds to an initial cloud's frame.



3D registration from multiple clouds (now we have a 360° model)

Chapter 6: Individual Contribution and Retrospective

Ali Berrada

a- Contribution

My contribution to this project is on 2 areas: project management and software delivery.

Project management:

At the very beginning, I have volunteered to be the project manager within my team and after consensus by all, I was so for about a month. My aim was to be the best team in terms of management as I was set to introduce industry standards of IT project management. During my short period as project manager, I have implemented a Scrum methodology and made use of Taiga.io for this purpose. I have also elaborated an on-boarding document to serve as a guide for the team to start up on the project by getting the tools of interest installed. I have run regular meetings online with Skype Enterprise which is used by many companies for efficient team work. For each meeting I will set reminders and document the points and ideas discussed. Also, I have delivered few knowledge sharing sessions. Finally, I have contributed to the final report mainly on the technical section.

Software delivery:

Firstly, the environment set up was a burden to most people due to very poor installation guidelines from previous projects. I have therefore came out with a comprehensive and step-by-step installation guide tested on a fresh Windows 10 environment that will make it much easier for future students (full length screen-recorded videos are also available). Secondly, as part of the initial technical work partition done within the team, I was tasked to pick up and handle OpenCV. I was therefore taking some time to get deeper on OpenCV to acquire strong fundamentals of it and be able to effectively use it for the project. I later had to break from OpenCV to start learning about PCL as the team faced reasonable difficulties with it and therefore it needed new ownership. This was unfortunately done in a very short period and late stage but I could manage to wire up things and come out with the different modules of the software. Additionally, I have used CMake as the build management tool for our project.

b- Retrospective

For me, this project was very interesting as it exposed me to a new theoretical topic evolving about 3D registration. I would hear about and see people using the Kinect for playing video games but little had I known about the mysteries it inboxes or the way it works or can be employed to do cool things such as 3D scanning. I also discovered the world of lenses from studying calibration. I appreciated how some of the concepts I have studied in other courses such as Linear Algebra and Image Processing revealed to be very much relevant. I have used previously different frameworks and libraries in Java and JavaScript (Spring MVC, Hibernate, AngularJS, Cordova, etc.) but this was the first time I make use of C++ external libraries or interfaces such as Boost, OpenCV and PCL. It was not an easy task at first and the learning curve was rather steep initially but soon I started to get the hang of it and could pick up the skill. Also, I have to admit that I did not give proper and enough time to tackle this project. This was due to my heavy load schedule during the whole semester. In fact, I struggled a lot with the courses as the last time I attended a class/studied was back in 2014. Rarely I would understand much from the lectures and it would take me several hours of self-study to cover an hour of lectures. This is in addition to the multiple homeworks which I would give priority as they were the earliest in terms of deadline. So basically, the only period I could really start putting my focus on this project was after the final exam week and I can estimate it would have been enough for me to deliver a very decent software but unfortunately, I ended up with other courses' projects which similarly I couldn't progress extensively during the lectures and examination periods. So, I can say that I've put on my sweat for a couple of days only, unfortunately. This being said, I wished we could have been given until the far end of the semester, so I would have been able to refine, improve and optimize the software.

On the other side, I used to have a laptop running Windows 7 with 4 GB of RAM. I have resisted immensely to switch to Windows 10 and spent days trying to find a way to run the previous years project on Win7 until I was left with one issue which is the inability to install the Kinect SDK 2.0 on Win7. I have then moved to Win10 but I knew I needed something else equally as important which is more RAM. With 4 GB, compilation was a total nightmare having to wait for very long (say 15 minutes at least). I have then upgraded to 12 GB which made compilation time take less than a minute. The move to Win10 and therefore installing the environment from fresh took me whole weekend (heavy windows updates under Acacias' network…) but I had the good instinct to record on video the whole process of setting the environment from basically 0 to compiling and running previous projects showing all the tricks and problem fixes.

SE3D advantages

- A software for 3D scanning that does the job
- Ultimate installation and setup guide
- OpenCV
- PCL
- CMake (even better, used within Qt with MSVC compiler)
- OOP