Software Engineering Project Report

Kinect based 3D Reconstruction of Human Body

Authors:

Roger Pi

Amjad Khan

Farid Ben Ali

Natalia Herrera

Supervisors:

Dr. Yohan Fougerolle

Dr. Cansen Jiang

David Strubel

January 7, 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Overview

3D scanners are now available from a number of commercial sources. They address a variety of target applications, not only for industrial purposes but also for academic ones. In fact, realistic 3D models of human bodies are widely required for computer graphic applications, such as animation, computer games, human computer interaction and virtual reality [1]. However, accurate body scanners are expensive, limiting the availability of 3D body models [2].

1.2 Objective

The main objective of this project is to create home-made acquisition and processing software for 3D scanning of the human body, to approximate such commercially available solutions.

To accomplish this purpose, the person is captured at different orientations with respect to a stationary camera, so RGB images along with per-pixel depth information can be collected. Then, the 3D information is integrated across different views.

For this project, we decided to start implementing code from scratch, using the codes developed by our predecessors as guideline. Throughout this document, we discuss the challenges we faced in designing this system, algorithms and solutions developed for handling scanned models.

1.3 Software requirements and installation

- Qt compiled with MSVC 2015 x64
- PCL 1.8.0 + VTK 7.0.0
- OpenCV 3.0

- Kinect v2 SDK
- Intel RealSense SDK

1.4 Hardware requirements

In this project the Kinect v2 is used as a main hardware. The Kinect v2 released in 2014 is an RGB-D acquisition device developed by Microsoft as a controller for the video game console Xbox One also developed by Microsoft. Kinect contains three elements which are a 1080p RGB camera, a 512*424 pixels depth sensor which contain an IR camera and IR emitters. Thanks to these sensors, Kinect provides us an RGB image, a depth map which means Kinect measures the distance of the surface of an object from the sensor. Its sensing ranges is between 0.5 and 0.45 meters and the image frequency is 30Hz [3].

1.5 3D Scanning process

Now, in the following diagram, we are going to define the 3D scanning process with its different steps:

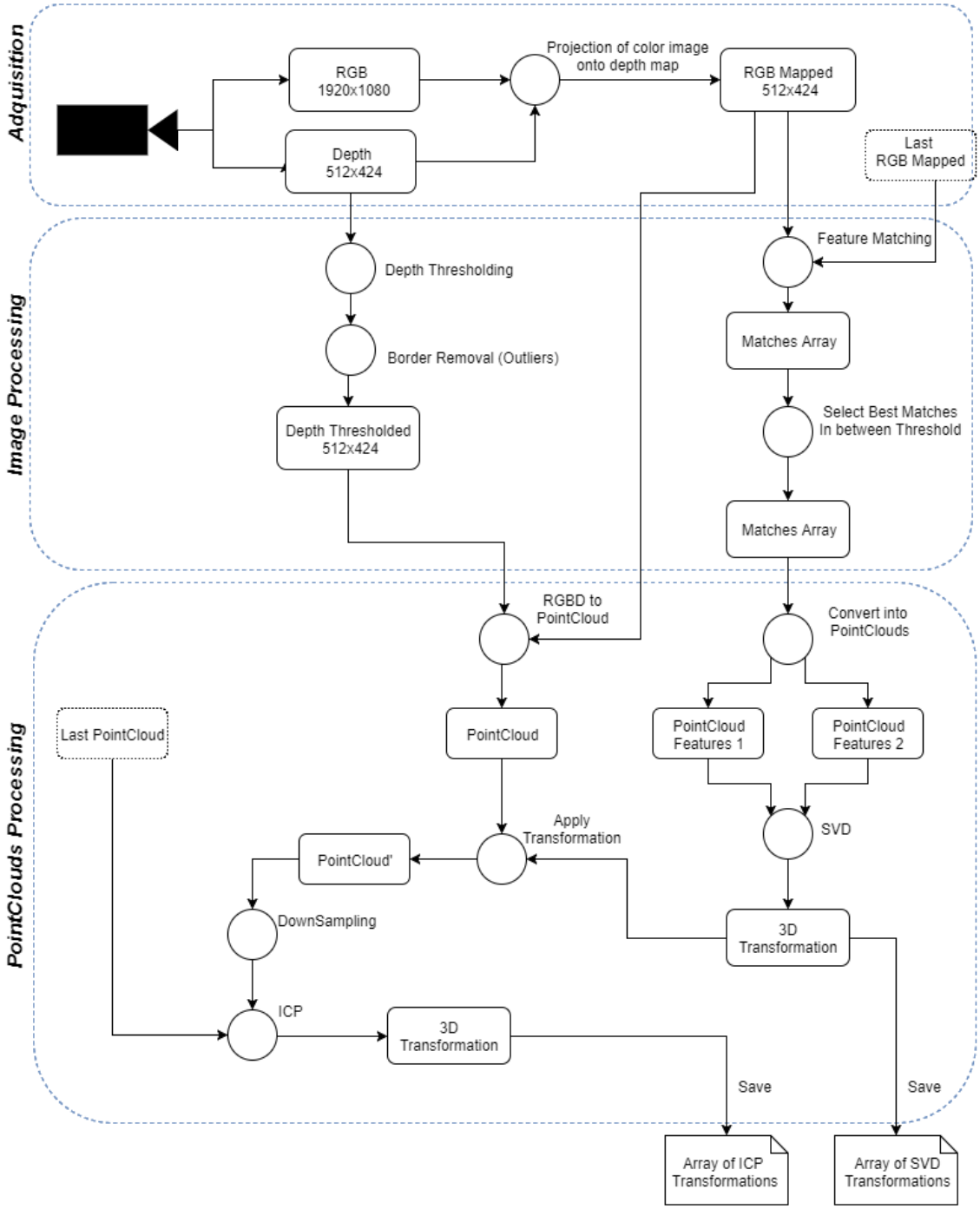


Figure 1.1: General Workflow

Last year projects were based on using ICP to get the transformation between pairs of point clouds. To improve the results, we decided to get correspondences between pairs of color images using feature matching and get the 3D points from these correspondences and calculate the 3D transformation matrix between these sets. This transformation can be used to map both point clouds onto the same coordinate system and then use ICP for a final adjust.

The scanning process can be divided into 3 different blocks: Acquisition, image processing and point cloud processing.The acquisition part consists of obtaining images from Kinect. These images have color and depth information that we can convert to point clouds later.

The next block is using image processing techniques to improve the results. First a threshold will be applied to the depth image to remove the background. Then due to most of the outliers are in the borders of the objects, where the depth sensor is not accurate enough, the borders will be also removed.

Finally, each consecutive pair of color images will be matched using feature matching, to get correspondence points. These correspondences will be filtered so only points of the object will be taken. These correspondence points will be used to find the 3D transformation matrix as a linear system problem.

The last block will be on point clouds. RGB and depth images of consecutive pairs of frames are going to be converted to point clouds and mapped onto the same coordinate system using the transformation obtained from correspondences, and then apply ICP to get the final adjustment between these point clouds. Note that the feature matching can be seen as a initialization of the ICP process.All these transformations between pairs of images are going to be saved so we will be able to transform any point cloud onto any of the coordinate systems.

In order to get the final point cloud, all point clouds are going to be transformed into the same coordinate system, added together and downsampled to get rid of repeated points. Also, there is the possibility to create a 3D colored mesh from the final point cloud.

Chapter 2

Project Management

Throughout this section we are going to present a generalize view of the set of skills, tools and techniques used to meet the project requirements.

2.1 System request

System request form	
Need	Software product for performing the 3D scanning of the human body
Functional requirements	• Acquisition duration must not exceed 90 seconds in total • Various processing of the data can be performed offline • The system must provide watertight triangular meshes • The system must allow for the export of 3D textured meshes • The user interface must allow for the visualization of the scan and individual acquisitions.
Time constraint	The project is designed to be finished during the first week of January 2018

Table 2.1: System request

2.2 Human Resource and Time Management

Figure 2.1 shows the work breakdown structure of the project and the involvement of the group members in each task.

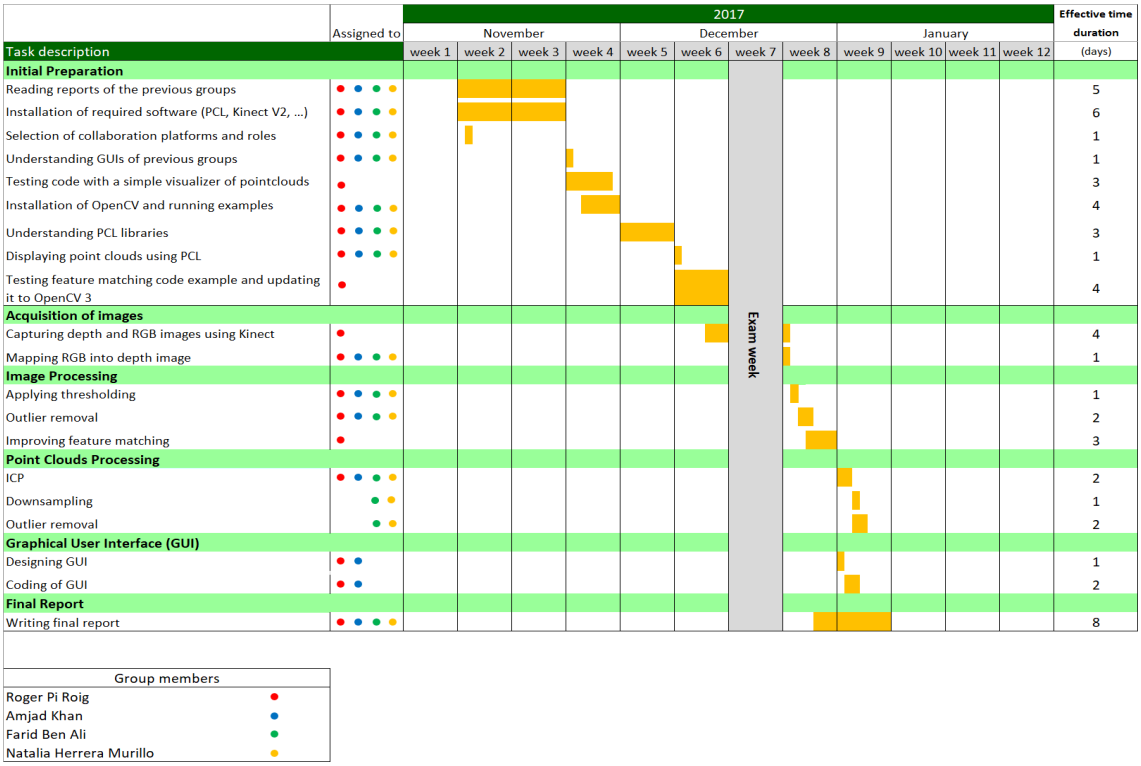


Figure 2.1: Gantt and task allocation Chart

2.3 Communications Management

Communication Management includes the processes used to ensure timely and appropriate generation, collection, storage and disposition of project information, in order to facilitate the participation of everybody involved in the project:

- Github was used for version control of documents and code. The URL for the github repository is https://github.com/Rogerpi/3D-Project
- Facebook group for communication between the team members, setup meetings and post ideas and suggestions.
- Slack for communication between teams and supervisors, and discussion of special topics.

Chapter 3

Image Acquisition

3.1 Acquiring RGB and Depth Images

Kinect allows to take different kind of images. In this section, we are interested in depth images and RGB images. First of all, Kinect captured an RGB image with a resolution equal to 1920x1080 pixels and a depth image with resolution equal to 512x424 pixels. In the depth image each pixel relates to a distance, along the Z axis, between the image plane and the corresponding object in the RGB image.

Figure 3.1: RGB image 1920x1080

Figure 3.2: Depth image 412x424

3.2 Mapping RGB into Depth images

Each sensor that mounted on Kinect v2 is located on a different physical position. Therefore, the data that is able to retrieve from the sensors has different coordinate systems.

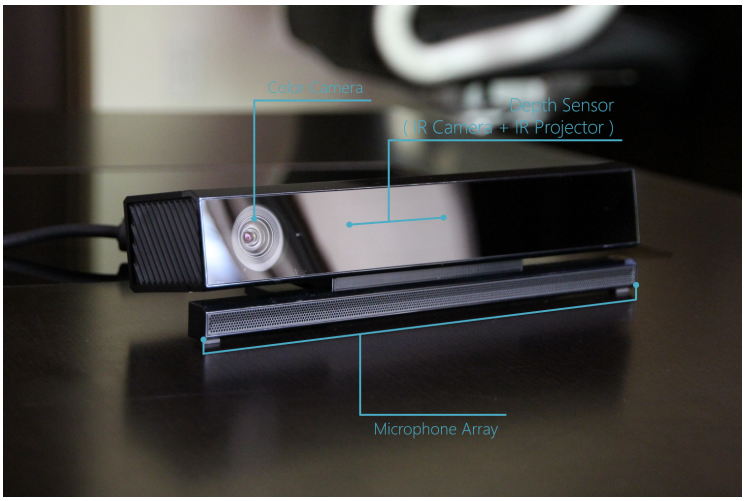


Figure 3.3: Kinect Coordinate Systems

The basic stream data retrieved from Kinect v2 has each one its own coordinate-systems. ICoordinateMapper class of Kinect SDK v2 provides features for mapping each coordinate-system. It has the feature to retrieve mapping in frame and point units.

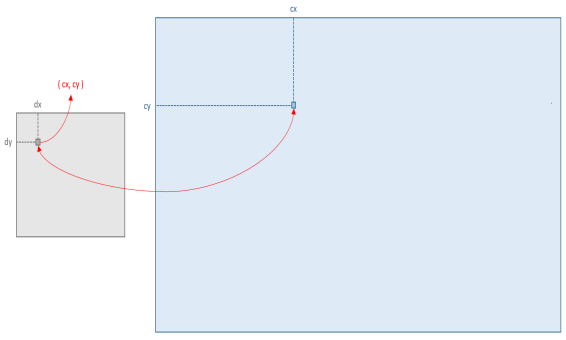


Figure 3.4: Coordinate mapping

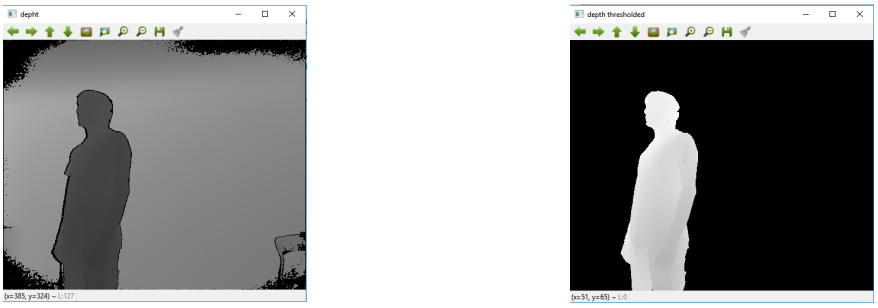
Using IcoordinateMapper class we can retrieve mapping information for each color point (cx, cy) corresponding to each depth point (dx, dy).

Chapter 4

Image Processing

4.1 Thresholding

A depth threshold method is used to roughly segment the human body data from the background. In our case, we go through all pixels of the depth image and if the depth is outside the threshold range it will be considered as zero. The depth is converted to meters to make it easier to choose.



(a) Full depth image (b) Thresholded depth

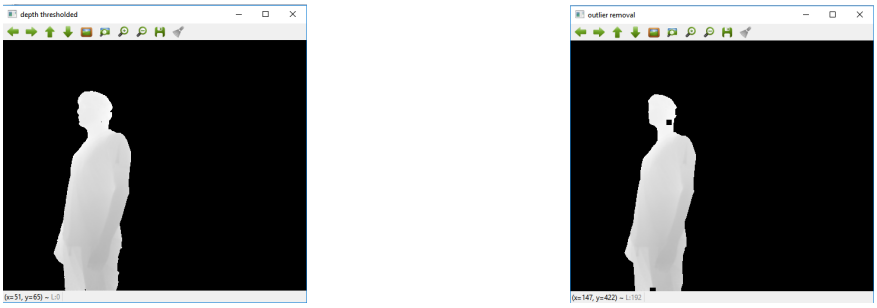
Figure 4.1: Thresholding depth image

Note: the color change because histogram is enhanced for visualization.

4.2 Outlier Removal

In addition, we considered that most of the outliers are close to the borders of the object. Also, color is bad mapped sometimes in borders. For the purpose of removing borders of the object an outlier removal method is applied.

Erosion is applied to the thresholded binarized image, and the result is used as a mask to get a new thresholded image where there are only depth values where the mask is true and 0 where is false.



(a) Thresholded depth (b) Erosion mask applied

Figure 4.2: Outlier removal method

4.3 Feature matching

For any object in an image, interesting points on the object can be extracted to provide a "feature description" of the object [4]. These features must remain at any rotation and scale. There are many ways to extract these points, but we will use the popular SIFT algorithm. These features can be matched between consecutive pairs of images to get correspondence points. The idea is to use these correspondences points to create a linear system as described in the following equation:

$$y = Tx \tag{4.3.1}$$

where y and x represent the set of points of the first and the second image. We use Linear Least Square method (LLS) using Singular Value Decomposition (SVD) in order to adjust at best the equation by minimizing it with the sum of squared differences between the data values and their corresponding modeled values.

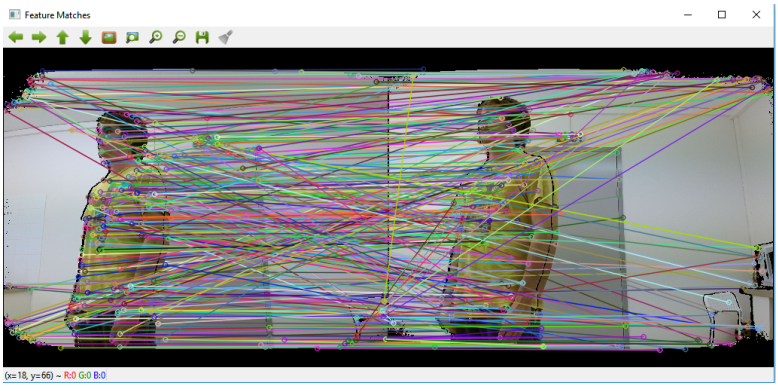


Figure 4.3: Feature Matching between 2 frames

The figure 4.3 shows that there are many matches, but some of them are not correct. A good practice in feature matching is to find the minimum distance between

key points and to take all matches with a distance less than the minimum distance multiplied by a scalar factor depending on how much reliability we are looking for.

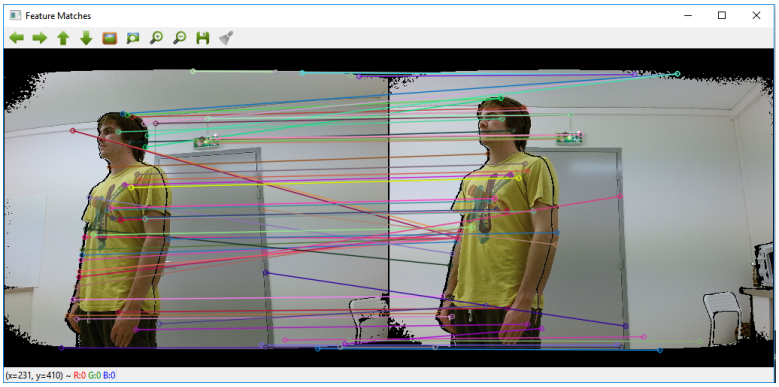


Figure 4.4: Robust Feature Matching between 2 frames

Matches look better on the figure 4.4, but we need to guarantee that matches correspond to the person and not to the background which surround the person.

To fix this issue, we need to threshold the color image using the depth thresholded image as a mask, and compute feature matching without background.

Figure 4.5: Feature Matching without background

On the figure 4.5, most of the matches are inside the body but losing background color information makes the matcher match features that have nothing to do.

Finally, we decided to perform feature matching with color images and select the matches inside the body using the thresholded depth image as a mask, so matches with features at 0 depth points will be discarded. From the remaining matches, we decided to take the best 50 matches.

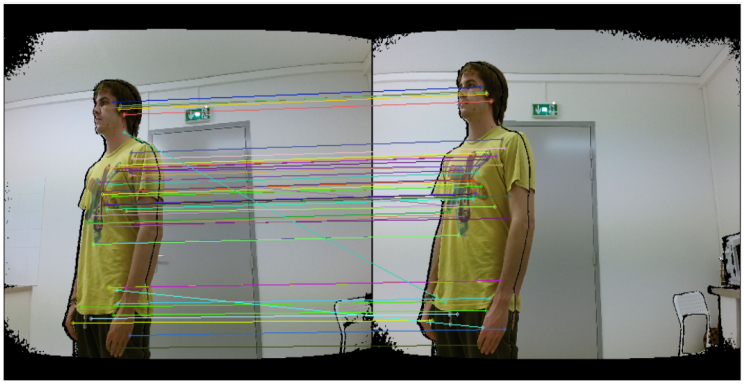


Figure 4.6: Feature Matching between 2 frames

These 50 best matches can be decomposed as 2 sets of points in the image space (i,j). Using these points and the depth image, we can get 2 sets of points in 3D space (X,Y,Z) that we will use get the transformation matrix between them as shown in equation 4.3.1

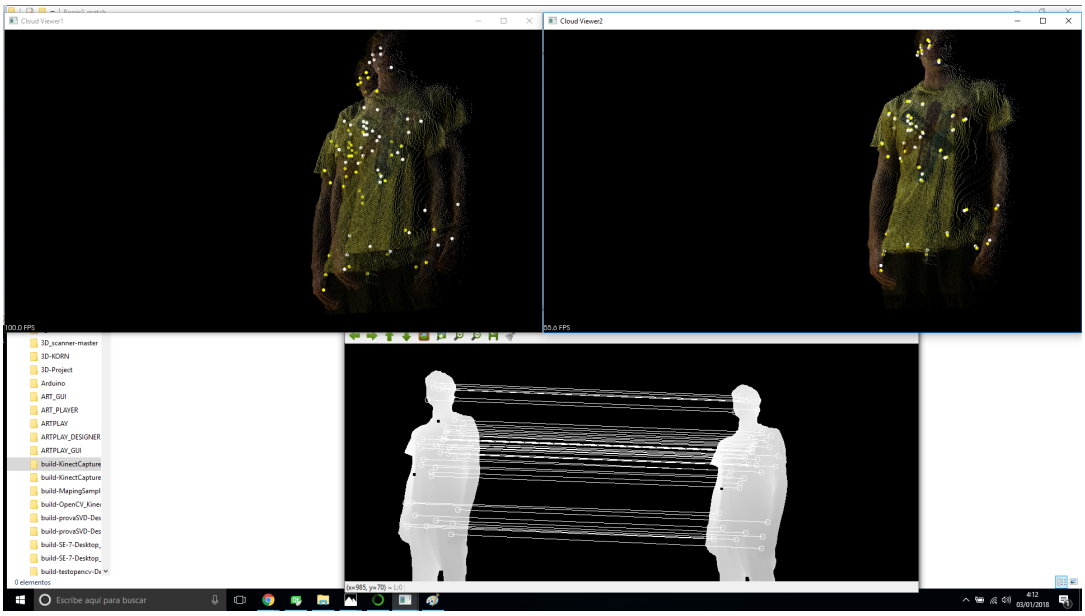


Figure 4.7: Feature 3D points

Figure 4.7 shows 2 consecutive images matched with its correspondents 3D points. On top left point clouds are in his own coordinate system each one. On top right, transformation matrix has been applied to the second point cloud.

When using more than 2 frames, transformation matrices must be chained to transform all frames into the first coordinate system.

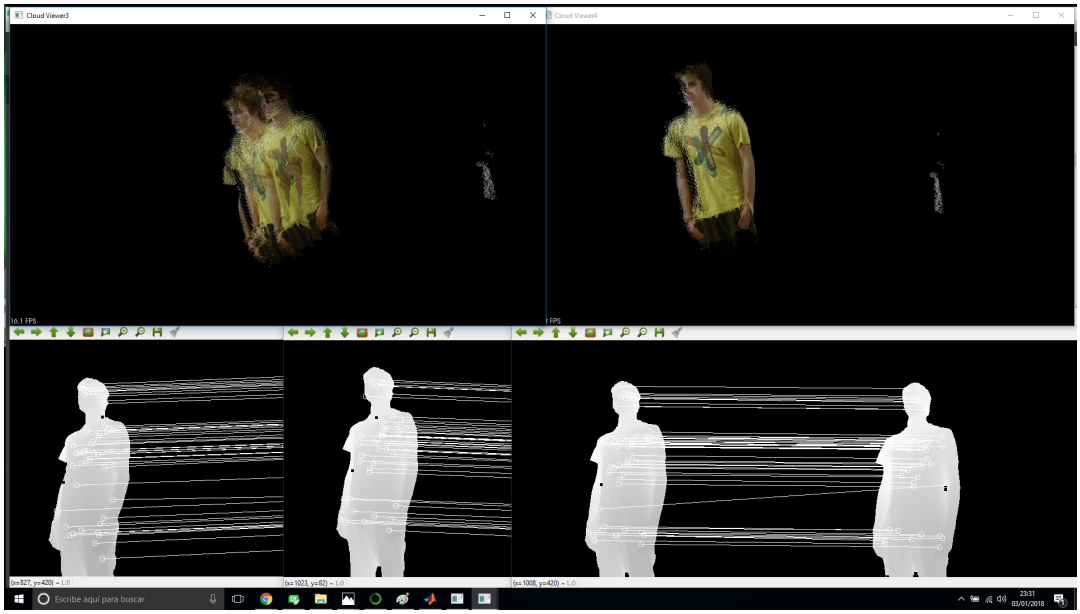


Figure 4.8: Transformation applied to 4 consecutive point clouds

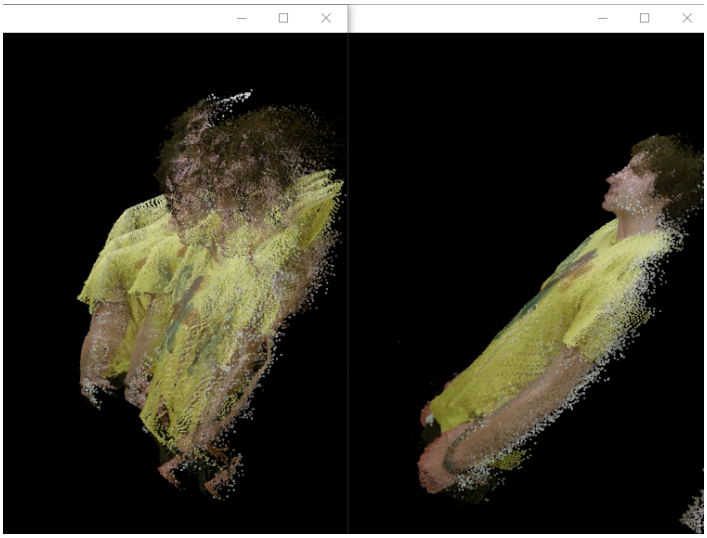


Figure 4.9: 8 consecutive frames transformed

The Figure 4.8 illustrates 3 consecutive point clouds which have been transformed onto the same reference system. On the figure 4.9, 8 consecutive point clouds have been transformed onto the same reference system. As we can see, results are quite close to the expected result.

Chapter 5

Point Cloud Processing

5.1 Iterative Closest Point

So far, feature matching has been used to obtain 3D transformations to compensate the rotation and translation occurred between several scans. However, this alignment is not perfect and bad matches can make SVD get wrong transformation matrices.

Iterative Closest Point (ICP) is an algorithm employed to minimize the difference between two clouds of points. The algorithm iteratively revises the transformation (combination of translation and rotation) needed to minimize an error metric, usually a distance from the source to the reference point cloud, such as the sum of squared differences between the coordinates of the matched pairs. ICP is one of the widely used algorithms in aligning three dimensional models given an initial guess of the rigid body transformation required [5].

Classical ICP is time consuming and often traps in local minimum when the overlapping region is not big enough. This means that same point clouds in different locations can became in different solutions, when there is only one correct alignment.

The idea is to make an initial alignment using feature matching so ICP doesn't get trapped at any local minimum, or at least, at a close one. Also, having a close initial alignment would make ICP converge faster.

ICP has 4 important argument:

1. The maximum number of iterations to make stop the algorithm if it hasn't converged.

2. A convergence criterion (transformation epsilon): If the sum of differences between current and last transformation is smaller than this threshold, the registration succeeded and will terminated. A value of (1e-9) seems quite reasonable and should give good initial results.

3. A maximum correspondence distance sets up a maximum distance threshold where correspondence points with higher distance will be ignored. This depends on how good your initial alignment is.

4. A RANSAC outlier rejection threshold to get rid of correspondence outliers. The method considers a point to be an inlier, if the distance between the target data index and the transformed source index is smaller than the given inlier distance threshold. This is normally set below the camera resolution.

There are many different ICP variants [5]. We decided to use ICP normal, where the ICP is performed on the normal vectors of the point clouds.

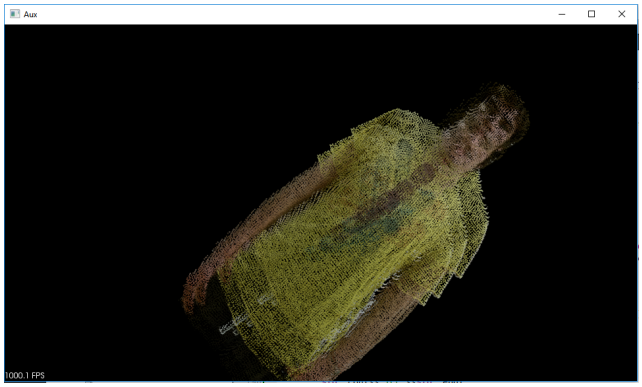


Figure 5.1: ICP with bad parameters

The figure 5.1 shows a point cloud that have been loaded and translated 0.1 m in X direction. Then ICP has been used to align it back to its position. We can see that ICP brings it closer, but considering that the two input clouds are exactly the same the result is bad.

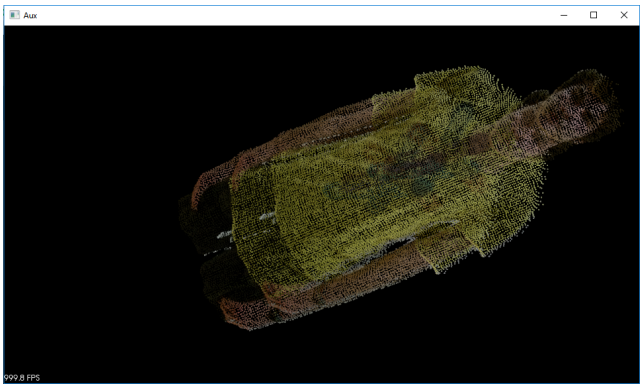


Figure 5.2: ICP correct result

After setting up a smaller maximum correspondence distance and a lower transformation epsilon ICP converges into a good solution, as seen in figure 5.2.

5.2 Downsampling

In order to reduce the points and get the points of interest in the point cloud data, the downsampling is performed prior to ICP algorithm. In this project, the voxel grid filter in PCL is employed to get the downsampled data. In PCL, the VoxelGrid class creates a 3D voxel grid (a set of tiny 3D boxes in space) over the input point cloud data. Then, in each voxel (i.e. 3D box), all the points present will be approximated (i.e., downsampled) with their centroid. This approach is a bit slower than approximating them with the center of the voxel, but it represents the underlying surface more accurately. Two examples of downsampling using different distances (5cm and 1cm) are shown in Figure 5.3 and 5.4 respectively.

Figure 5.3: An example of voxel grid based downsampling using 5cm distance

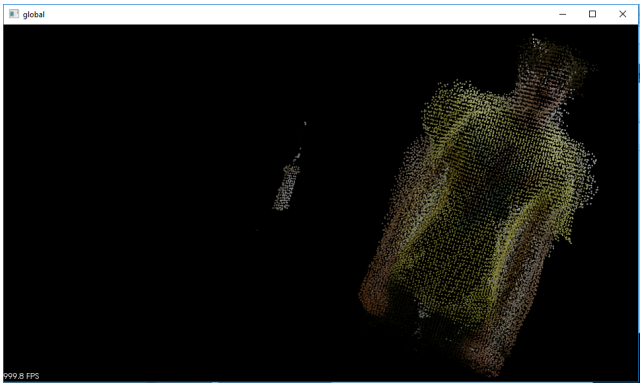


Figure 5.4: An example of voxel grid based downsampling using 1cm distance

5.3 Outlier Removal

The oulier removal process removes all unwanted points in the point cloud data. In this project, the radius based outlier removal technique provided in PCL. The RadiusOutlierRemoval in PCL filters points in a cloud based on the number of neighbors they have. Iterates through the entire input cloud and for each point, retrieves the number of neighbors within a certain radius. The point will be considered an outlier if it has too few neighbors as determined by the given number of minimum neighbors with given radius.

The neighbors found for each query point will be found among all points of input cloud, not just those indexed. The method only indexes the points that will be iterated through as search query points. Figure 5.5 and 5.6 show before and after outlier removal using radius outlier removal technique.

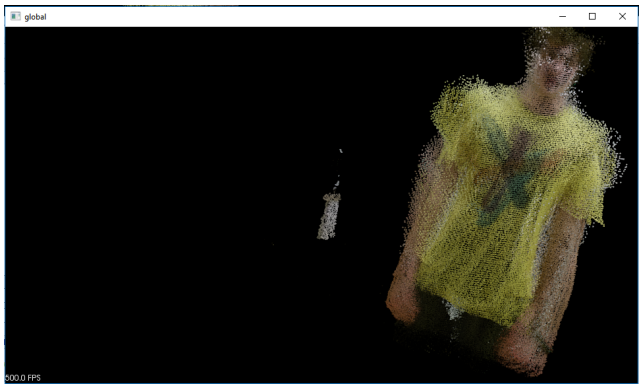


Figure 5.5: The point cloud before outlier removal

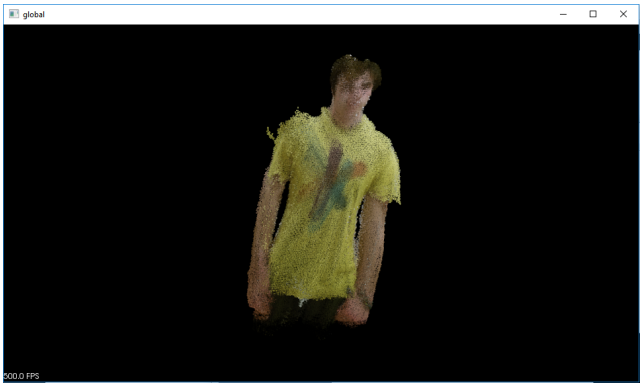


Figure 5.6: The point cloud after Radius Outlier Removal

Chapter 6

Graphical User Interface

The graphical user interface (GUI) provides an easy access to new users of the developed application. In order to integrate all the developed functions in the project, the GUI has been developed with the help of QT designer. In the following section a brief description of the GUI with its functionality is presented.

6.1 Main Window

The development of GUI is designed as simple as possible so that new user can use the functionality of the main application easily. The main window is shown on Figure 6.1, it contains two parts, one is for editing and controlling the input or output parameters and the other one for displaying functionality.

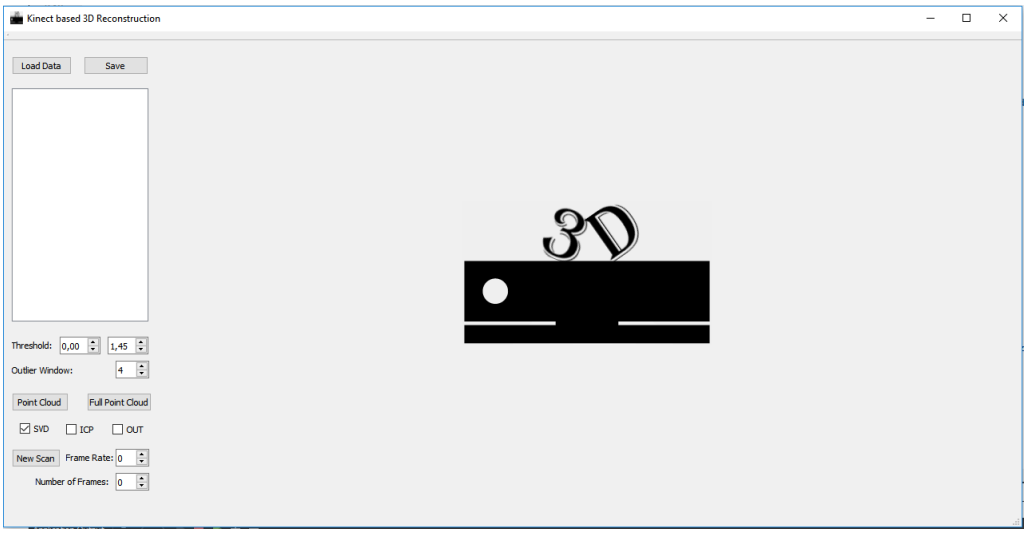


Figure 6.1: Preview of Main Window of Graphical User Interface (GUI)

6.2 Edit and controls

The edit and controls part includes load data, save, list, threshold, number of outlier window, point cloud generation, registration algorithm selection and new

data acquisition functionality as shown in Figure 6.2. The Load Data button provides the user to load the depth or RGB image data from the storage location and its names display in the List Widget with their extensions. Two spin boxes of Threshold label provides the user flexibility to set the lower and higher depth limits. The Outlier Window spin box is used to set the size of the window in outlier removal process. Point Cloud button provides the user registration of two point clouds of two images based on the selected algorithm from the check boxes such as SVD, ICP and OUT (Outlier Removal). Similarly, the Full Point Cloud returns the point cloud registration of all the loaded images in the GUI. There are also acquisition functionality if user wants to acquire new data from the Kinect device on a selected frame rate and number of frames by inserting values in two given spin boxes.

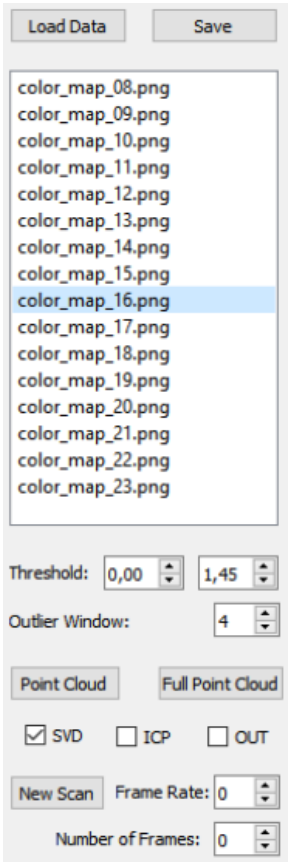


Figure 6.2: Preview of edit and controls of GUI

6.3 Display Window

The processed images are displayed in the view area of the GUI as shown in Figure 6.3. Each two or more images can be selected from the list by mouse cursor to generate their processed version in the main display window integrated to GUI. On each new operation of point cloud generation a new window is opened to display the results.

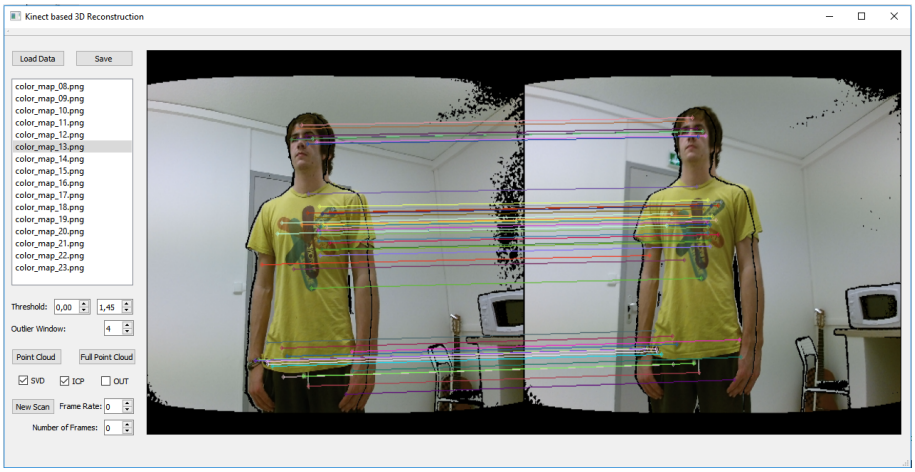


Figure 6.3: Preview of display window of GUI

Chapter 7

Results

For the final result, we tried to do a 360 degree scan and use our approach (SVD + ICP) to register all the obtained point clouds.

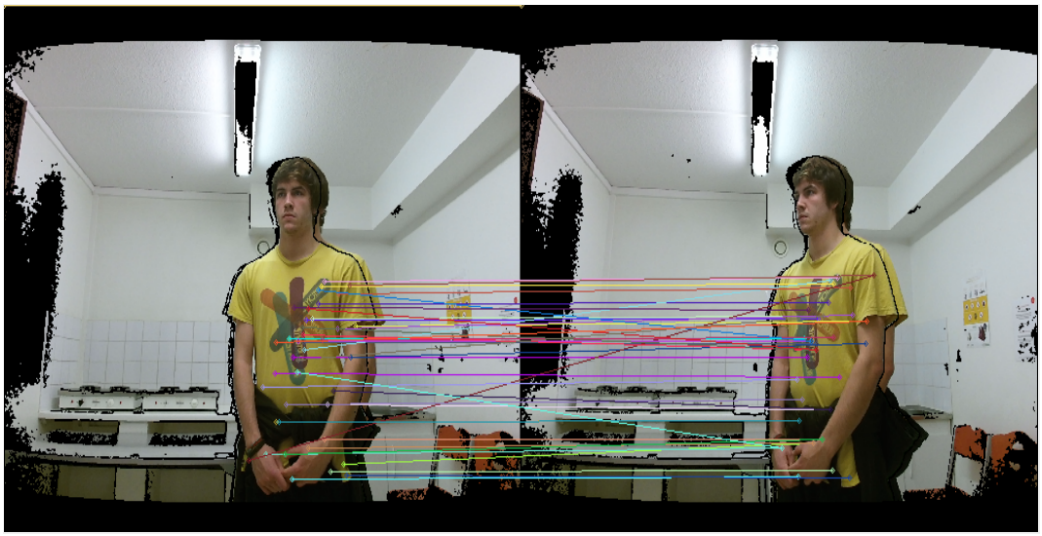


Figure 7.1: Scanning frame 1

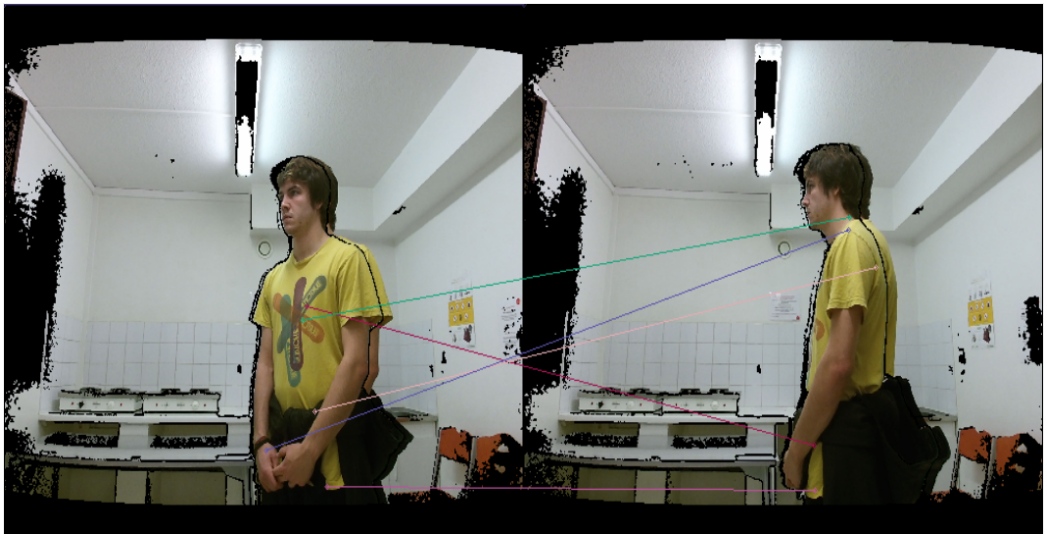


Figure 7.2: Scanning frame 2

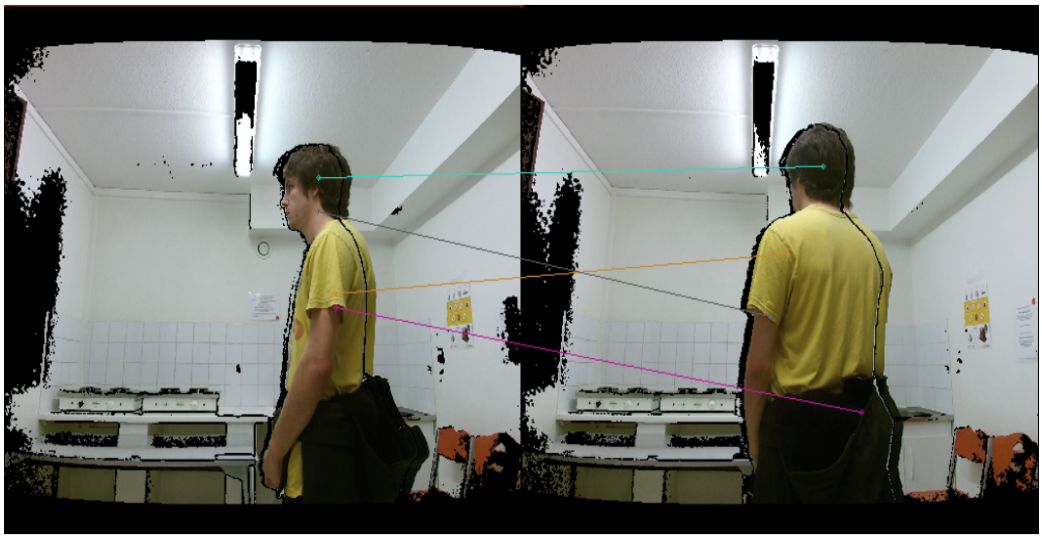


Figure 7.3: Scanning frame 3

As we can see on Figures 7.1, 7.2 and 7.3, feature matching doesn't work fine when big changes appear between frames. Actually, to make the feature extractor more robust we chose specific t-shirt with pattern that makes the extractor focus there.

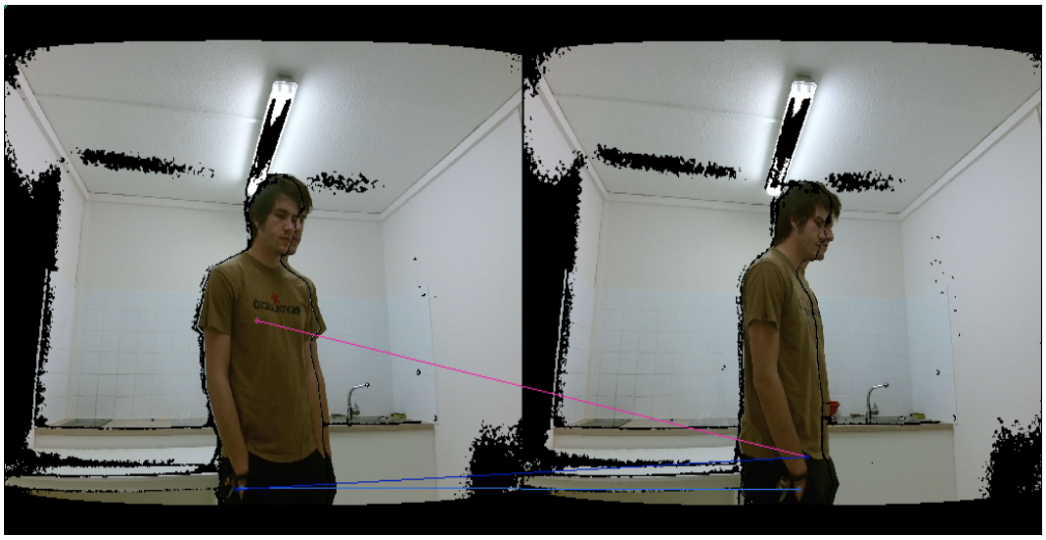


Figure 7.4: Bad features

As we can see on figure 7.4, only few features are matched inside the body, all the others are matched outside. Of course we cannot use only 3 matches to solve it as a linear system, even less when the features are matched incorrectly.

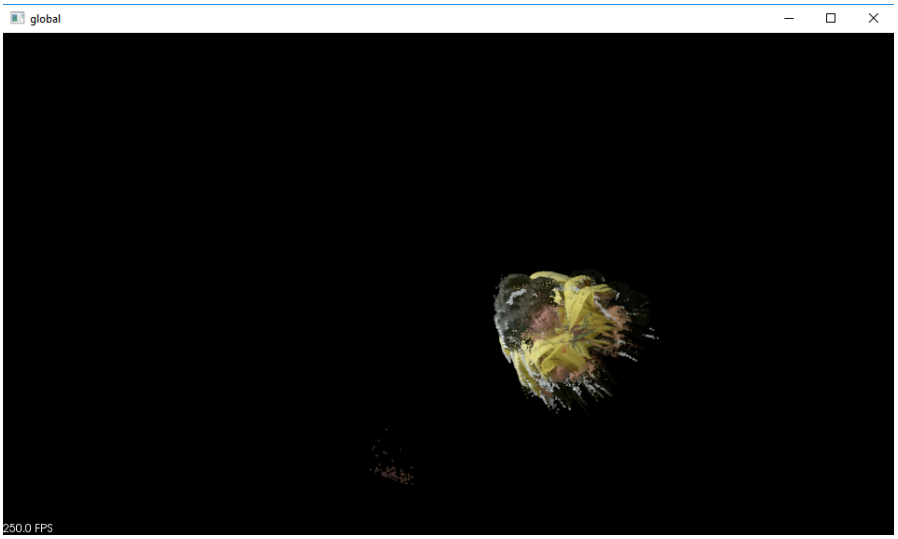


Figure 7.5: 360 degree scan original frame

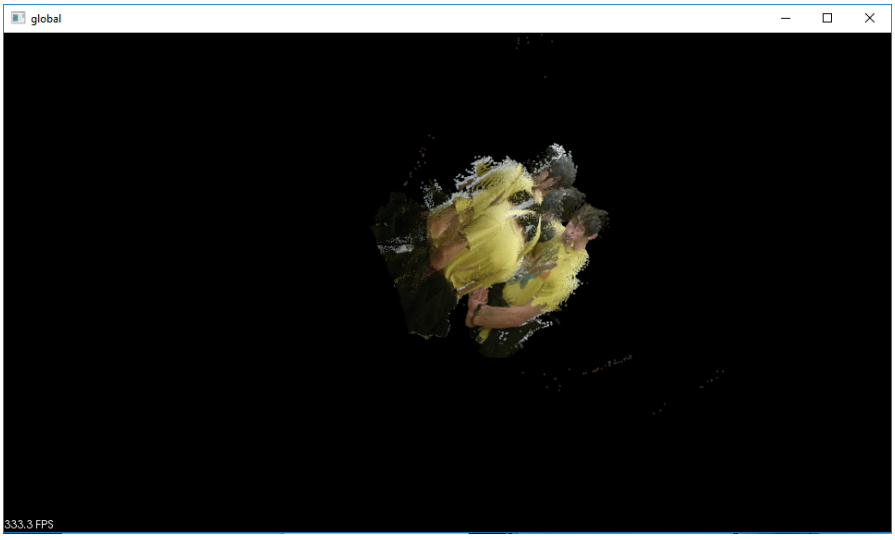


Figure 7.6: 360 degree scan using svd transformations

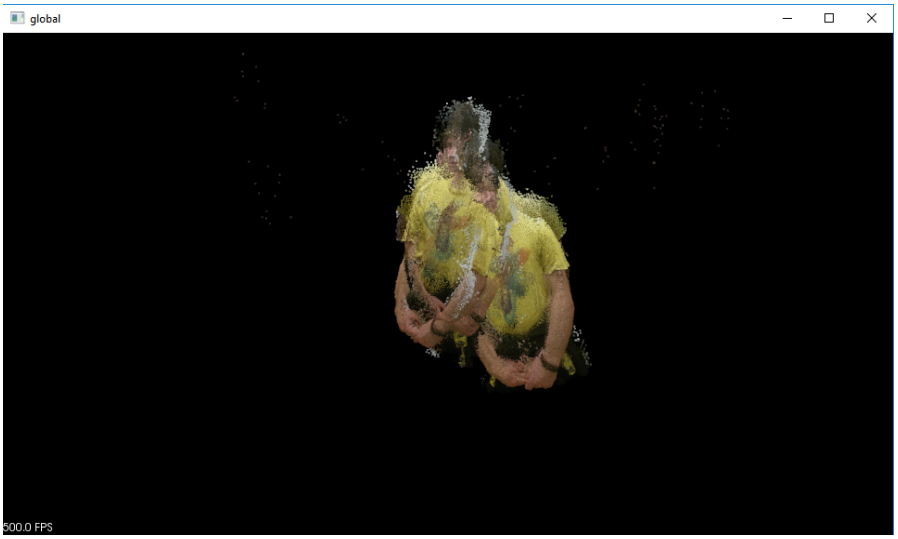


Figure 7.7: 360 degree scan using svd + icp transformations

The transformation matrices are stored in arrays that save transformation between pairs of frames, so any bad transformation matrix will make future point clouds carry this error.

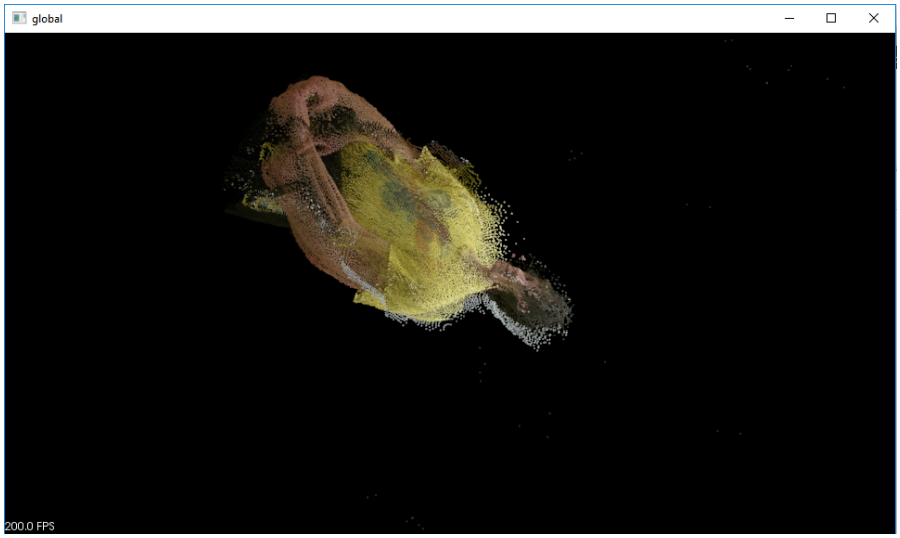


Figure 7.8: 360 degree scan using ICP transformations

So in some cases, if we carry big errors on SVD will make registration output worse rather than only using ICP, as seen on Figure 7.8.

The more frames we get the more error transformations we will carry on. The principal idea was to do 360 scan in 6-8 frames maximum. However, SIFT didn't

work as expected when the object has yaw rotation, because most of the features change between frames and this make the matcher take lots of false matches. So we conclude that SIFT is not good enough for obtaining transformations.
Next improvements could come trying to use other feature extractors as:

- Speed Up Robust Features (SURF)
- Binary Robust Invariant Scalable Keypoints (BRISK)
- Fast Retina Keypoint (FREAK)
- Fast Local Descriptor for Dense Matching (DAISY)
- Video tracking as Optical Flow tracking (OpenCV)

We would like to focus our attention on optical flow tracking and, specially to DAISY feature extractor.
The DAISY local image descriptor is based on gradient orientation histograms similar to the SIFT descriptor. It is formulated in a way that allows for fast dense extraction which is useful for e.g. bag-of-features image representations. This seams to be robust between frames with yaw rotation because its basically used to extract depth from different color frames.
The idea would be to use it to extract and to match the features and use the depth image given by Kinect to extract the 3D points of the matches.

Chapter 8

Conclusion

In this project, we tried to design a low cost 3D scanner. We studied the tools and material that could be useful to initiate the project. We had also studied the previous developed projects provided by the supervisor. Then, we had to decide what software and tools were suitable to proceed the project in order to achieve the objectives. As our initial objective was to improve one of the previous year projects, however during the practical implementation we had took very different approach on each steps that means we did not use or utilize any part of the previous year's projects except the main ideas. Therefore, starting from scratch not only helped us to understand and learn new techniques but also reduces the computational complexity of overall development as compared to previous projects.

Bibliography

[1] J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan, "Scanning 3d full human bodies using kinects," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 4, pp. 643–650, 2012.

[2] A. Weiss, D. Hirshberg, and M. J. Black, "Home 3d body scans from noisy image and range data," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1951–1958, IEEE, 2011.

[3] D. Pagliari and L. Pinto, "Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors," *Sensors*, vol. 15, no. 11, pp. 27569–27589, 2015.

[4] M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, "Image features detection, description and matching," in *Image Feature Detectors and Descriptors*, pp. 11–45, Springer, 2016.

[5] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pp. 145–152, IEEE, 2001.