3D - KORN
SOFTWARE ENGINEERING PROJECT REPORT

BY

CANALINI LUCA	**MAIA**
DE LA ROZA EZEQUIEL	**MAIA**
LALANDE CHATIN BENJAMIN	**MAIA**
PAOLELLA ROBERTO	**MAIA**
RAMAN KUMAR UMAMAHESWARAN	**MAIA**
BONHEUR SAVINIEN	**VIBOT**
CLERIGUES GARCIA ALBERT	**VIBOT**
GONZALEZ ADELL DANIEL	**VIBOT**
DOUSAI NAYEE MUDDIN KHAN	**MSCV**
GHIMIRE PAMIR	**MSCV**
MENG DI	**MSCV**

UNDER THE GUIDANCE OF
PROF. YOHAN FOUGEROLLE

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

UNIVERSITY OF BURGUNDY

2016-2017

Contents

List of Figures

Chapter 1

Overview

The main goal of the project is to create a 3D scanning system capable of capturing the full body of a human. Complete 3D scanning systems available in the market for scanning full body of a human being consist of a turn-table that can support a person, as well as a rig for holding and moving the camera. In order to qualitatively approximate such commercially available systems, in this project, we have developed a fully automated hardware-software setup capable of capturing a person in 3D.

Figure 1.1: Application Workflow

The setup consists of a rotating table and a stationary camera. The person is captured at different orientations with respect to the camera and the data is stitched to create a 3D reconstruction. The reconstruction is desired in the form of a watertight mesh. The person is placed on a turning table in front of a stationary depth-sensitive camera and data is collected as the person rotates 360 degrees on the table. We then register the captured data, which is a cloud of points, and use the registered data to create a water tight mesh.

Chapter 2

Project Planning and Management

Project planning and management is one of the key aspect of this project as there are 11 members in the group, each belonging to different nationalities and unique skill sets. This called for the use of proper software methodology in order to make the requirements achievable.

2.1 Development platform

2.1.1 Software requirement

1. Qt 5.7.0 (MSVC 2013, 32 bit)
2. MSVC 2015 build tools
3. PCL 1.8
4. Microsoft Kinect V2 SDK
5. Intel R200 SDK

2.1.2 Hardware requirement

1. Microsoft Kinect V2
2. Intel R200
3. DC Motor
4. Encoder
5. Arduino
6. Wooden platform

2.2 Collaboration platform

Considering the size of the group different collaboration platforms were used for different purposes.

2.2.1 Trello

Trello is a web based project management application. The URL for the trello board is `https://trello.com/b/inpECRpD/unicorns-main`

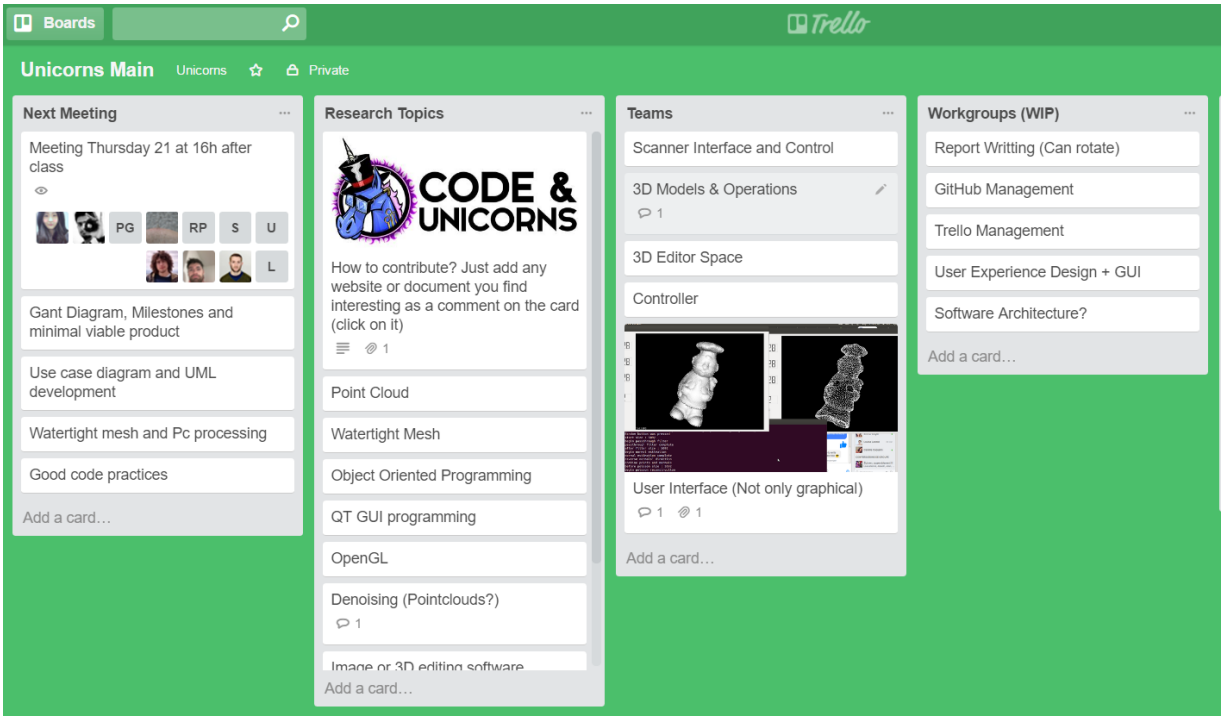


Figure 2.1: Trello board

It was mainly used in the project for the below purposes:

1. Create storyboard
2. Setup meetings
3. Ideas and suggestions
4. Research work

2.2.2 Github

Github is a web based Git repository system used for version control of documents and code. The URL for the github repository is `https://github.com/umaatgithub/3D-KORN`

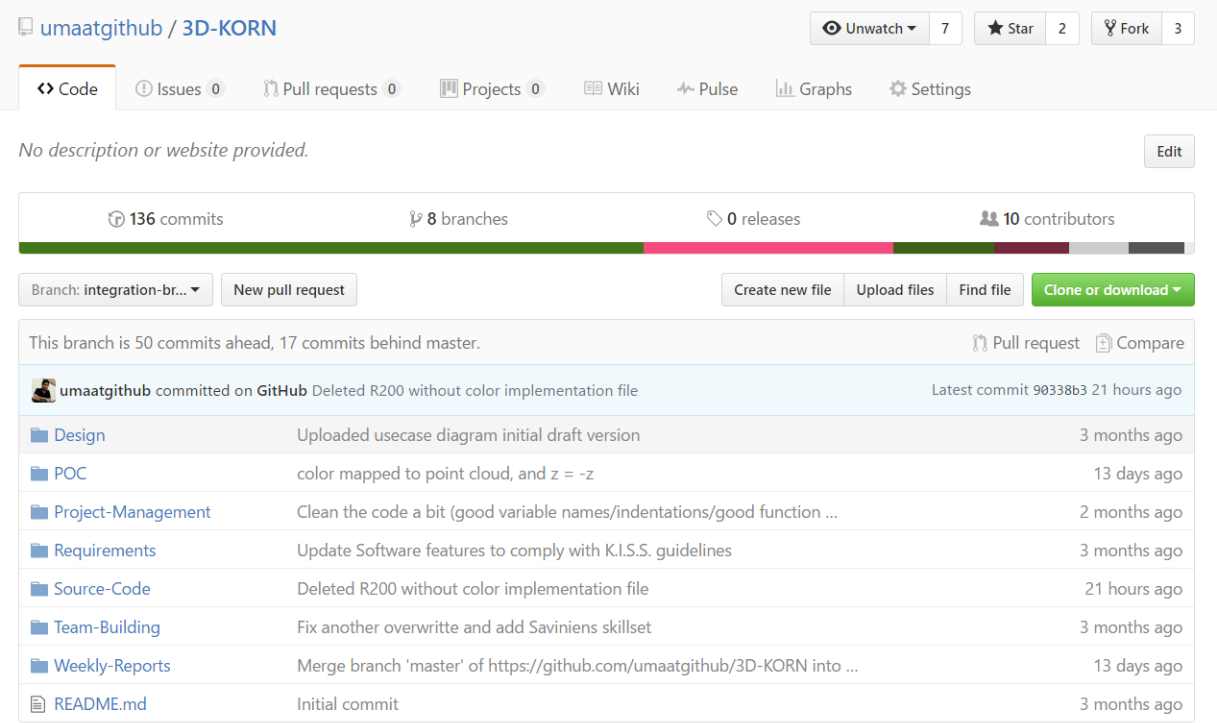


Figure 2.2: Github repository

It was mainly used in the project for the below purposes:

1. Version control of documents and code
2. Collaborative programming
3. Integration platform

2.2.3 Google docs

Google docs is a web based office suite provided by google to allow users to create and edit file in a collaborative environment. It was used in the project for tracking individual activities. The url for task allocation sheet is `https://docs.google.com/spreadsheets/d/1NwtdiDk9ZyLn8eN4s2UB4rhJ3VC3MinktOQm662WW1E/edit?usp=sharing`

TASK DETAIL	TASK TYPE	ALLOCATED TO	STATUS
Create excel sheet for task allocation	MANAGEMENT	UMAMAHESWARAN	COMPLETED
Create class diagram	DESIGN	UMAMAHESWARAN	COMPLETED
KT to team on coding standard	MANAGEMENT	UMAMAHESWARAN	COMPLETED
Create Use case diagram	DESIGN	UMAMAHESWARAN	COMPLETED
Create Gantt chart	MANAGEMENT	UMAMAHESWARAN	COMPLETED
Building PCL 1.8 and retrieveing Pointcloud	POC	ALBERT	COMPLETED
converting .vtk to .stl	CODING	DI	COMPLETED
Build PCL for windows and get the watertight color mesh	CODING	NAYEE MUDDIN KHAN	COMPLETED
Developing turn-table for scanning	DESIGN	PAMIR	COMPLETED
	CODING	PAMIR	COMPLETED
Interfacing QT with turntable	CODING	DANIEL	COMPLETED
Data fusion & registration using PCL	RESEARCH	EZEQUIEL	COMPLETED
	CODING	DANIEL	COMPLETED
Interfacing QT with external hardware	CODING	PAMIR	COMPLETED
Design 3D turn table	DESIGN	PAMIR	COMPLETED
	CODING	LUCA	COMPLETED
Studying and developing a crop-function	CODING	ROBERTO	COMPLETED
Research documentation for PCL	RESEARCH	LUCA	COMPLETED
Research on PCL 1.8\found all-in one installer	RESEARCH	ROBERTO	COMPLETED

Figure 2.3: Task allocation document

2.2.4 Facebook and Messenger group

The facebook and messenger groups were created for ease in communication between the team members.

2.3 Skillset analysis

First Name	Managing	Research	UML	OOP	Qt	Documentation
LUCA	1	3	1	1	1	3
EZEQUIEL	2	4	1	1	2	3
BENJAMIN	2	2	1	2	3	3
ROBERTO	2	2	1	4	2	3
UMAMAHESWARAN	4	3	4	4	3	3
SAVINIEN	3	3	1	4	3	2
ALBERT	4	3	3	4	1	2
DANIEL	3	3	1	3	1	3
NAYEEM	4	3	2	3	2	3
PAMIR	2	4	1	3	2	3
DI	2	2	1	3	1	2

Table 2.1: Member analysis of skills evaluated from 1 - Beginner to 4 - Excellent

2.4 Team structure

The team was divided into 4 groups after survey of skill set of the members. Each group had a mix of experienced and inexperienced people so as to improve the overall standard of inexperienced programmers.

1. Group 1 : Acquisition team
2. Group 2 : 3D Registration team
3. Group 3 : Pointcloud Operations team
4. Group 4 : Graphical User Interface team

2.5 Coding standards

Since the team has to work in a collaborative platform certain standards were followed to make the code easily understandable and readable by everyone.

1. OOPS coding standards
2. Project specific coding standards
    - Class name to begin with *TDK*
    - Member variable name to begin with *mv*
    - Member function name to begin with *mf*

Chapter 3

High level design

The design of the application is the integral part of the software life cycle process where the application starts to take shape. If the design is not robust enough and fails to meet the requirements then the application developed will become instable. Hence lot of research was done before arriving at the final design for the application which will be explained in the coming chapters.

3.1 Use case diagram

The diagram depicts the different features provided by the application at a high level and the types of users who can have access to the system.

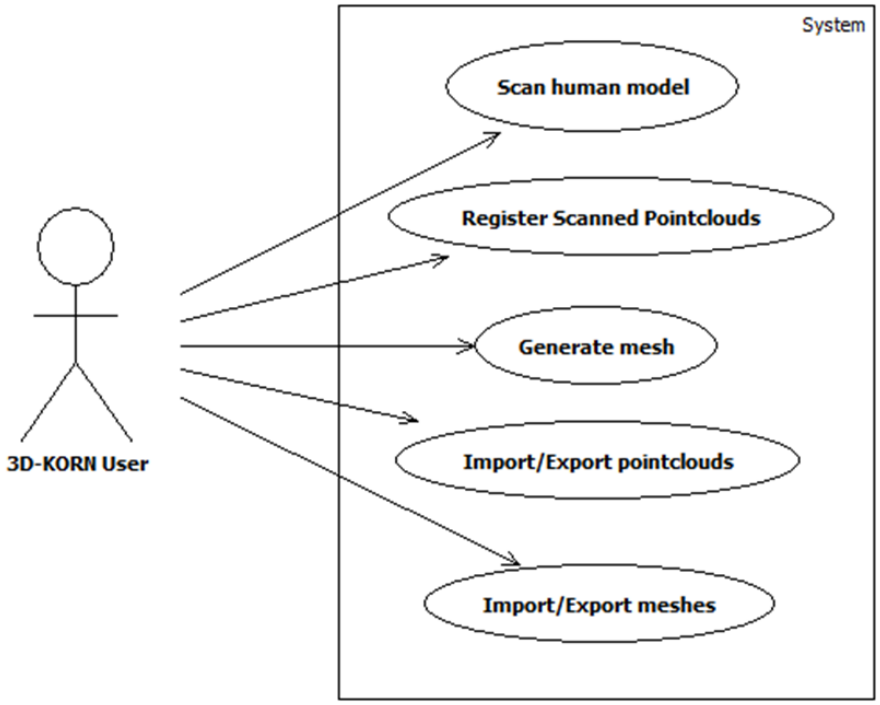


Figure 3.1: Use case diagram

3.2 Class diagram

The main factors taken into account when designing the class diagram are listed below :

1. Reduce complexity
2. Increase modularity
3. Create reusable classes
4. Ease of maintenance
5. Team size

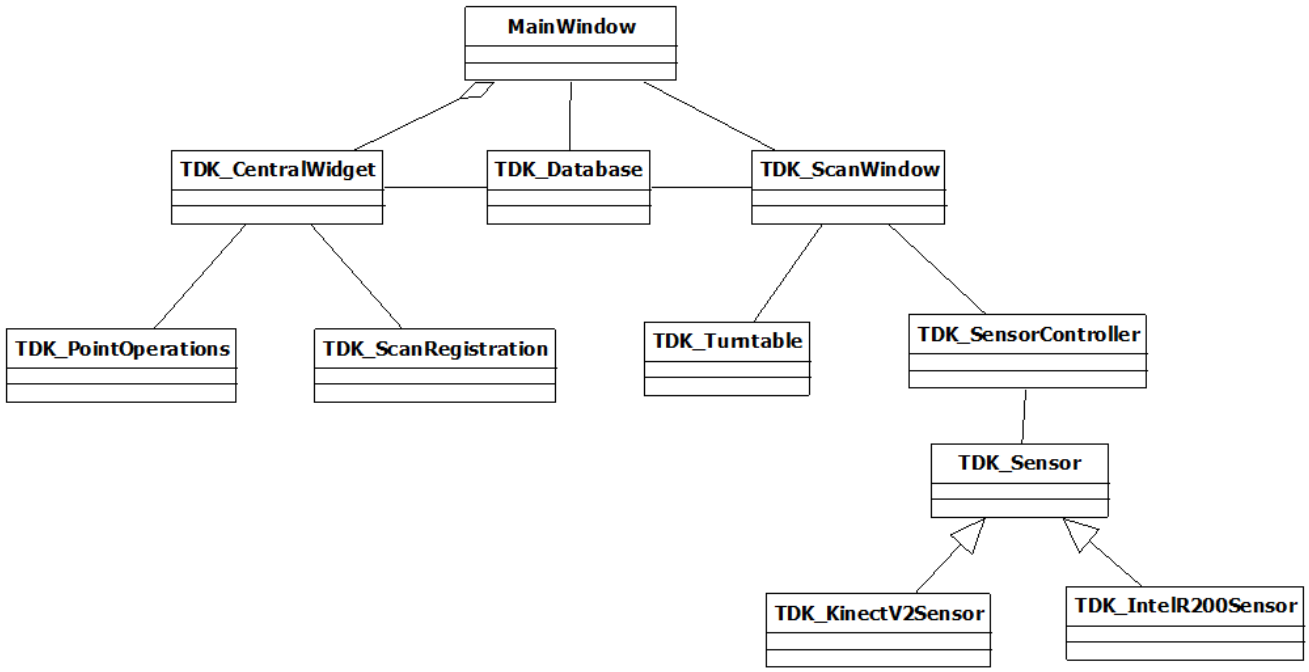


Figure 3.2: Class diagram

Chapter 4

Pointcloud acquisition

4.1 Introduction

Due to the fact that the project depends on a sensor, it is essential to have an acquisition module to obtain the desired sequence of data for all its posterior processing.

In order to enhance the probabilities of being this setup useful for the final user, two sensors have been taken into account and development has been done for both:

1. Microsoft Kinect V2.
2. Intel R200.

Also, in order to have a complete 3D-scanning system, we must assure that the hardware is reliable enough. For that purpose, a complete turntable prototype has been designed and developed thanks to an inter-team agreement in which the Unicorns team has played a major role (see turntable group report for details on that aspect).

4.2 Class structure and functionality

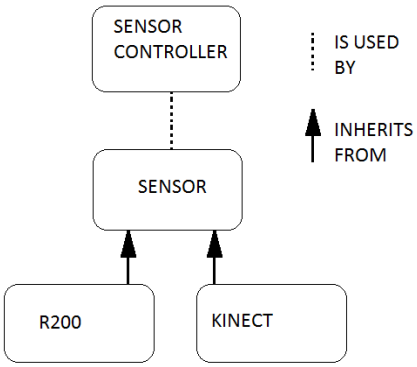


Figure 4.1: Inheritance and usecase diagram for implemented sensors.

The general idea of a sensor for capturing 3D world data is contained in 'sensor', an abstract class. Two classes, one for the Kinect V2 and one for Intel R200, inherit from this class. The pure virtual functions that each of the inheriting classes must implement are listed below:

1.$boolmf\_IsAvailable() = 0$
2.$boolmf\_SetupSensor() = 0$
3.$boolmf\_StartSensor() = 0$
4.$boolmf\_StopSensor() = 0$

We make these functions pure virtual functions so that they must be implemented by a child-class since the implementation is different for different sensor types, and unless these functions are redefined, when they are called using a pointer of parent class type, the method should be different for different types of underlying objects.

The abstract class also makes it very easy to include an additional sensor since the upstream code can deal with the new sensor using a pointer to the abstract class and keep the upstream workflow similar to one for an old sensor.

Besides these functions, we also implement functions for each sensor to control data-capture criteria, particularly a 3D cropping box that filters out data stream that lies outside a defined cuboidal volume in the world (coordinate system), and functions to adjust capture frame-rate and resolution.

4.3 Differences between sensors

One of the strengths of our setup is that it is versatile when it comes to the sensor that can be used.

4.3.1 Microsoft Kinect V2

The Microsoft kinect v2 is a time-of-flight sensor. It uses one infrared camera coupled with an infrared projector system.

It has a 1080p (1920 x 1080) RGB camera and a 512 x 424 pixels depth camera, both of which can stream at 30 frames per second. It's angular field of view in horizontal and vertical planes is 60 and 70 degrees wide respectively and it's sensing range is from 0.5 m to 4.5 m.

The kinect v2 can not work well in sunlight because the infrared light from the

Figure 4.2: Configuration of sensors in kinect.

sun overwhelms the signal from its infrared projector. Also, two kinect v2s working close by can interfere with each other.

4.3.2 Intel R200

Figure 4.3: Configuration of sensors in Intel R200.

The Intel R200 is an infrared-projector assisted stereo camera. It uses a known pattern projected using an infrared laser projector to assist the stereo vision system to identify depth.

It has a color camera with 1920 x 1080 pixels and a depth/IR camera with 640 x 480 pixels. The color camera can stream at 30 fps and the depth camera can stream

at 30 and 60 fps. We hence ask the sensor for aligned capture at 30 fps.

The angular field of view of the sensor in horizontal and vertical planes is approximately 50 and 60 degrees respectively and its sensing range is from 50 cm to 2m.

The sensor has problems perceiving dark textures, as well as depths of brightly lit scenes. The R200 is highly portable because of its small size, and uses a single usb for power as well as data. This is unlike the kinect which is bulky in comparison, and requires a separate powerblock.

4.4 General acquisition procedure in both sensors

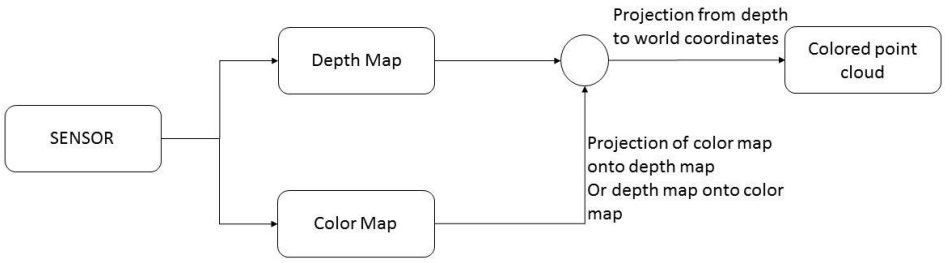


Figure 4.4: General work flow for acquiring 3D (xyz) data from an RGBD sensor. We choose to map color stream onto the depth stream because the depth stream has a lower resolution.

We get two kinds of data streams from both our cameras, RGB color-stream and a monochrome depth stream. The depth map is a gray scale image whose gray-levels correspond to the distance of the pixel from the depth sensor. The depth image is of lower resolution than the color image for both our cameras.

Because the depth and color images correspond with different points in space due to their difference in resolution and their different locations on the sensor, we map one of the images onto to other.

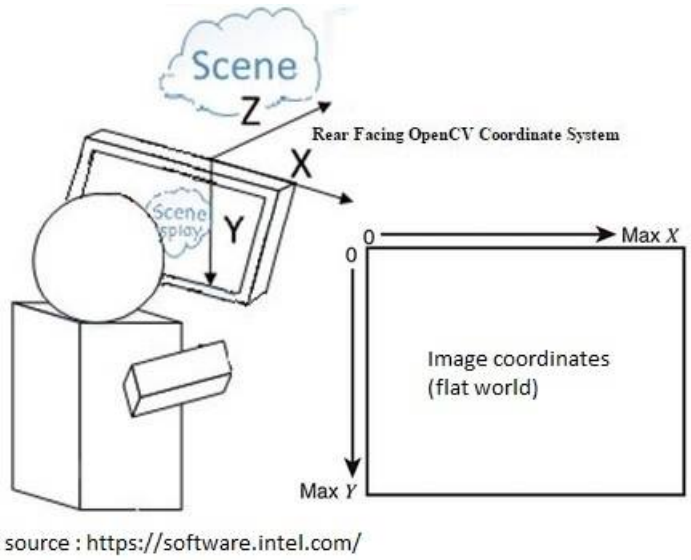


Figure 4.5: World coordinates and Image coordinates. The two are different and mapping functions exist to transform one to the other.

We infer the xyz coordinates of a pixel in the depth image using its xy position in the depth image and its intensity. After the depth image has been mapped to a point in 'camera coordinates' (xyz vs flat xy), we use the mapping between the color and depth images to assign a color to each of our points in the calculated point cloud.

If the user has enabled volumetric filtering, we include only those points in the returned pointcloud whose xyz coordinates comply with the constraints defined by the cuboidal cropping volume.

Acquisition as a separate thread: Because we desire a stream from a camera, once it has been started, to be displayed continuously on our GUI's scan window, we need to query the sensor regularly for the aligned streams. However, if we do this querying in the same thread as the rest of our program, the program becomes too slow to use due to high latency. We therefore execute the acquisition of pointcloud data from sensor stream as a separate thread. We use boost library to create these separate threads and register the acquisition routine (functions) with this thread, that can then run alongside the main program thread.

4.5 Specifics - Microsoft Kinect V2

The first of the sensors used is the Microsoft Kinect V2. Having as a base what is mentioned in section 4, we define the particular workflow that happens in the specific algorithm for acquiring the data from Kinect and make it shared.

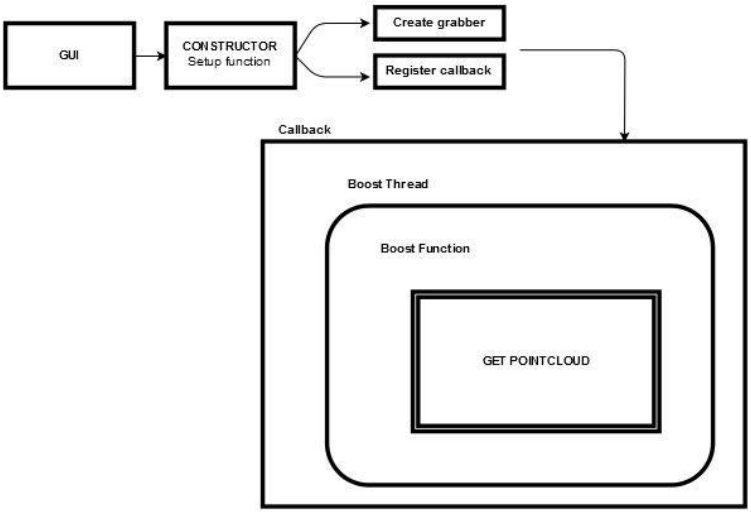


Figure 4.6: Workflow diagram for Kinect sensor.

For understanding how the acquisition behaves with Kinect, we can refer to the figure 4.6, and understand it as a consequence of the user starting the acquisition pipeline:

1. When the starting command is sent from the graphical user interface, the constructor of the Kinect's class executes a setup function.
2. This setup function performs two main actions:
   (a) Create a grabber object of the type kinect2grabber (the class created by the Github user UnaNancyOwen, which is actually a wrapper around the PointCloudLibrary grabber) to be able to acquire data from the sensor.
   (b) Registers a callback that will be executed every time a new valid pointcloud is acquired and will make it shared so it can be accessed by the other parts of our algorithm.
3. A separate thread (created using the Boost library) starts running parallel when the grabber is started and executes the general acquisition procedure (see 4.4).
4. The callback gets executed because of the acquisition of a pointcloud, checks again it is a valid pointcloud and it is made available for the rest of the parts of the algorithm.

4.6 Specifics - Intel R200

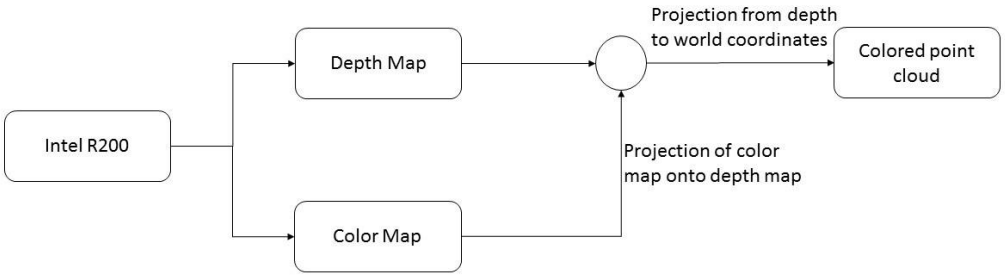


Figure 4.7: Specific workflow of acquisition from Intel R200.

We developed a grabber for the R200 using the Intel Realsense SDK 2016 R2 Documentation available online.

We acquire aligned depth and color images at 30 frames per second from the sensor. The SDK also provides the option for unaligned captures. We acquire color images at resolution of 640 x 480 pixels and depth images at 320 x 240 pixels.

For mapping color image to depth image and the depth image to camera co-ordinates, which are 3D coordinates with origin at the camera's centre, we use the Projection interface available in the SDK. The projection interface can be exposed by first creating a device through a session object, which can be obtained from the device manager interface.

We notice that the depth buffer is 16 bits in length per pixel and the color buffer is 32 bits per pixel, with 8 bits each for B, G, R and A channels, in that order.

4.7 Turntable

In order to achieve, as we have said in the introduction, the complete solution to the problem of a 3D-scanning setup of people, a turning table must be designed. In that way, our team covers from the very first designing of code until the very last screw placed on the turntable, so a complete coverage of the pipeline from designing a product until having it constructed is done.

For that purpose, an inter-team group was arranged for constructing the turntable and programming an interface between the hardware and the software for all groups; the sketch that runs on the Arduino is the turntable software that is accessible to all

groups. In the turntable group, the Unicorns team played a major role in the development of the whole setup (this can be seen in the specific report for the turntable group, as in this report the turntable device is seen as a black box, in which we input commands and from which we receive angle data).

In essence, the whole turntable setup can be summarized in the graph of figure 4.8.

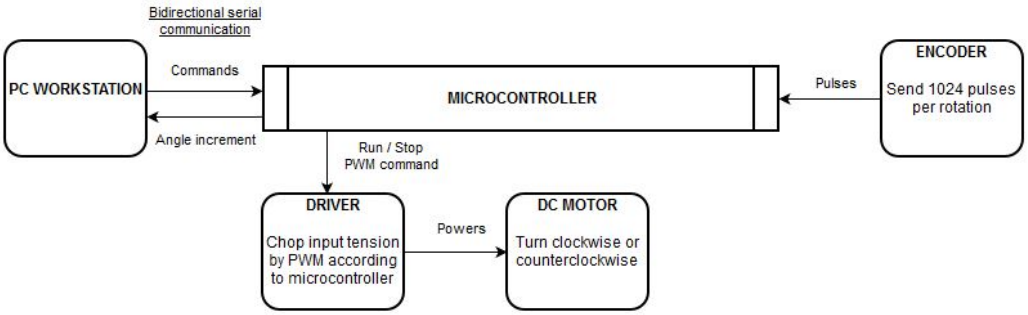


Figure 4.8: High-level workflow of the turntable setup.

As it is seen in figure 4.8, the whole setup is composed of:

1. A microcontroller (in this case an Arduino UNO has been used).
2. An H-Bridge driver L298N.
3. A 12V - 8RPM dc motor.
4. A high precision encoder that delivers 1024 pulses for each rotation that its shaft accomplishes.

When it comes to design an interface with the physical platform, the focus has been on dealing with simple commands, such that eventually the number of them is two:

1. Start rotating the platform.
2. Stop rotating the platform.

Dealing with the algorithm running in the microcontroller, there are two main functions that it performs:

1. Regulate PWM tension (what includes starting and stopping the platform).
2. Send data via serial every time the turntable has accomplished 5 degrees of rotation (changeable by code).

Our team has developed two modes of scanning:

- Manual scan: pointclouds can be acquired independently, turning manually the turntable.
- EncodScan^{TM}: this new approach makes use of the encoder placed on the turntable for accomplishing a robust, highly accurate angle-based scan.

4.8 Conclusion

Implementing our two sensor classes as children of a generic sensor class (inheritance mechanism) was central to developing the classes so that they could have similar interfaces and functionalities.

About the sensors themselves, we concluded that data captured using the kinect was superior in quality to that captured using the R200, particularly because of its greater range which allowed placing the camera far enough to capture full body view.

We also decided to acquire our scans based on the signals we received from the encoder on the turntable. This was for two reasons. First, our ICP (Iterative Closest Point) registration pipeline demanded angular rotation of the turntable between successive scans. Second, the belt drive tended to slip and consequently, stop the rotation of the platform intermittently. Basing the acquisition on timing would have introduced errors in both angle measurements between successive scans as well as the total number of scans we desired to acquire.

The acquisition module has been designed so that the minimum effort has to be done by the user, letting him to make, once the parameters on angle between successive scans and number of rotations have been set, only two decisions regarding the platform: to start it or to stop it. That dramatically improves the overall user-experience and simplifies the turntable.

Chapter 5

Registration

5.1 The registration problem

Once multiple frames from the 3D object have been acquired, we are faced with 3D point clouds that are in different frames of reference relative to the scanned object. To transform all 3D point clouds to the original frame of reference we need to apply 3D transformations to each point cloud to compensate for the rotations and translations occurred between several scans.

Figure 5.1: Example of captured point clouds.

The process of 3D data alignment is generally known as registration. Its corresponding algorithms addresses the problem of finding the optimal transformation that maps a pair of point clouds between them, considering that both datasets are exposed to camera noise during acquisition. Overall, there are two main families of registration methodologies: rigid and non-rigid ones. While the first family assumes that the mapping transformation can be modeled by using six degrees of freedom, the latter one considers the problem of softer bodies whose shape can change during data acquisition [Bellekens 2014].

Registration algorithms can work, independently on the way a pair of point clouds is registered, pairwise or by multiview registration approaches. Although pairwise registration allows easier implementation, generally suffers from the problem of drifting due to the accumulation of small alignment errors. This kind of problems can be addressed and corrected using loop closing algorithms.

5.2 First approach: General Case

Our first approach to the registration problem used pairwise registration based on feature correspondences. The original point cloud was initially downsampled to around 40% of the original data size in order speed up computation by means of data dimensionality reduction. Then, a transformation matrix was estimated with respect to the previous downsampled point cloud using features descriptors. Once pre-alignment was obtained, a final aligning step was conducted and the obtained transformation using downsampled versions was applied to the original raw point cloud to finish the pairwise registration. A flowchart of this registration approach is shown in Figure 5.2.

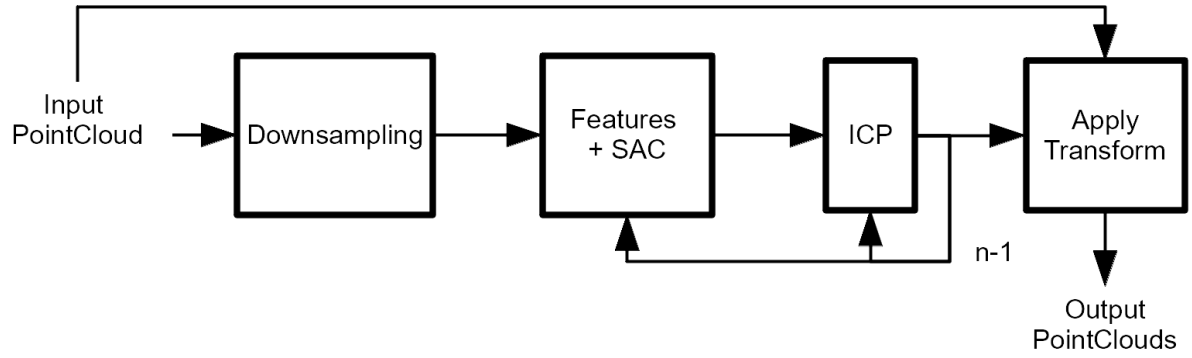


Figure 5.2: Chart flow for the first registration approach.

In the first step, a voxelized grid approach was conducted using the VoxelGrid class in order to downsample the input point cloud. Here, a custom size 3D voxel grid is created over the input cloud and all data inside each voxel is represented by means of its voxel centroid. As result, the input point cloud is described by a lower dimension 3D cloud that still preserves the high-dimension cloud morphology.

After downsampling, the correspondence matching between point clouds was eased by using Fast Pointer Feature Histograms (FPFH), which consists in informative and pose invariant local features describing the surrounding surface properties of a point [Rusu 2009]. The method receives the downsampled point cloud achieved in the previous step as well as its estimated point normals. A searching radius is custom set in order to define the region of analysis. The output of this step consists on a set of informative features describing the point cloud.

For pre-aligning two point clouds based on its FPFH descriptors, Sample Consensus Initial Alignment (SAC) was conducted. The method consists in an exhaus-

tive search in the correspondences space (FPFH descriptors of the clouds) in order to find the best transformation that aligns data based on its features [Rusu 2009].

Finally, once a pre-alignment was achieved using FPFH and SAC methods, the Iterative Closest Point (ICP) algorithm was used to register the last point cloud with the previous one. The method uses singular value decomposition to estimate the affine transformation that better aligns data. Iteratively, outliers are discarded and point correspondences are redefined after finding the best transformation matrix [Bellekens 2014]. The search distance for the algorithm is custom determined. Since pre-alignment with SAC was conducted, a short distance was chosen (maximum correspondence distance of 5 cm). Once the best ICP-based transformation for the clouds was found, it was applied to the original high-dimensional cloud jointly with the SAC obtained transformation.

5.2.1 Problems and Limitations with the first registration approach

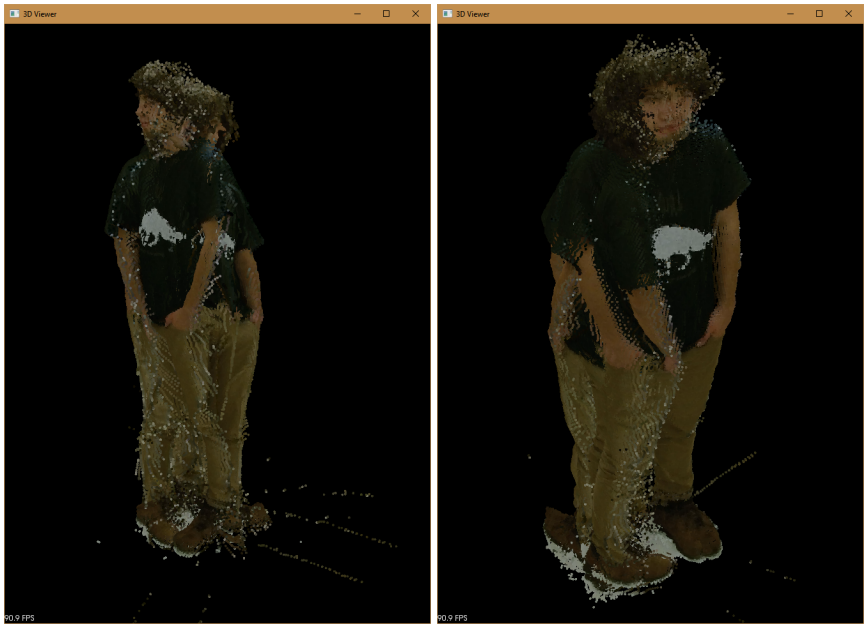


Figure 5.3: Registration obtained with the first approach.

The results shown in Figure 5.3 were not as good as we expected and later analysis led to several conclusions:

- SAC feature matching was not working as expected. An initial pre-alignment was expected after this step. However, in some cases the transformation obtained deviated the point cloud much more than the initial input, adding more complexity for ICP.
- The poor initial pre-alignment was making ICP obtain non-desired results. Mainly, the algorithm get stuck in local minima in the optimization process.

- An important observation after this approach was that only using pairwise registration is not possible for global registration. The main problem underlies in drift multiplication, providing very poor global alignment for the last processed clouds.
- Another conclusion regards data pre-processing. Downsampling step only reduces the size and complexity data, but still remain noise and outliers in the cloud. This suggests the use of outlier removal techniques in order to discard non-informative points.

It was clear the need of severe modifications in the registration approach accounting with the problems and limitations herein found. Hence, an improved registration algorithm was proposed as explained as follows.

5.3 Second approach: Controlled Environment

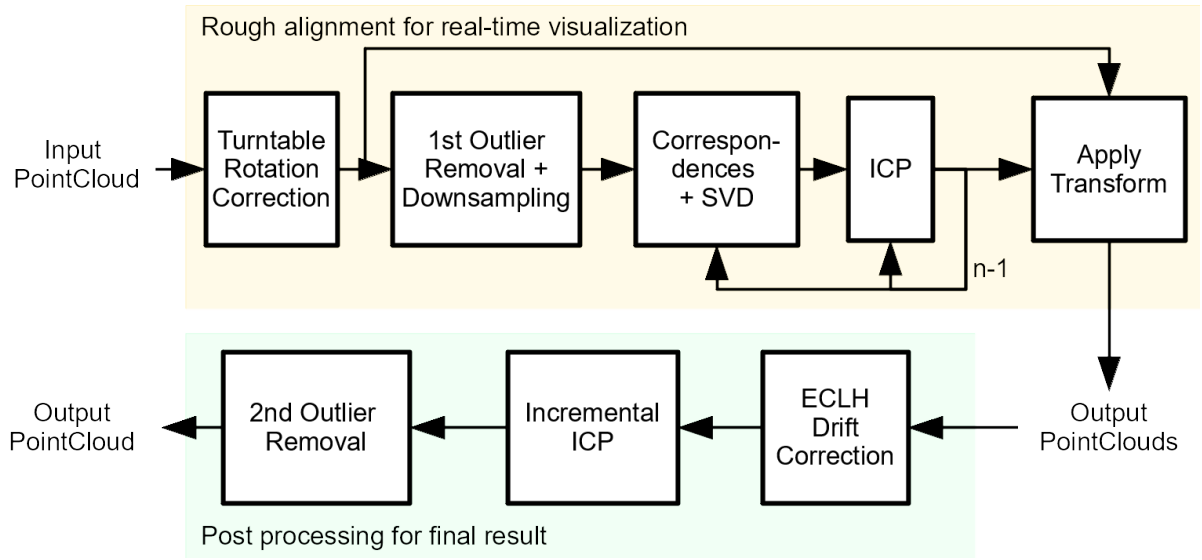


Figure 5.4: Chart flow for the second registration approach.

This second approach aimed to solve ICP getting stuck at local minima by providing a better initial alignment and have two types of registration: real-time registration designed to be fast to show the result in real-time to the user and a post-processing registration slower but much more robust, dependant on having all views from the object, also using the results from real-time registration process.

5.3.1 Pre-alignment knowing turntable parameters

Seeing the bad results the general approach gave, and after some analysis, we reached the conclusion that the main problem was a poor alignment estimation. We thought that we could get a good estimate of the transformations between point clouds since

we know the acquisition process: the general transform from one point cloud to the next is a rotation in the height axis of some degrees around the axis of rotation of the turntable.

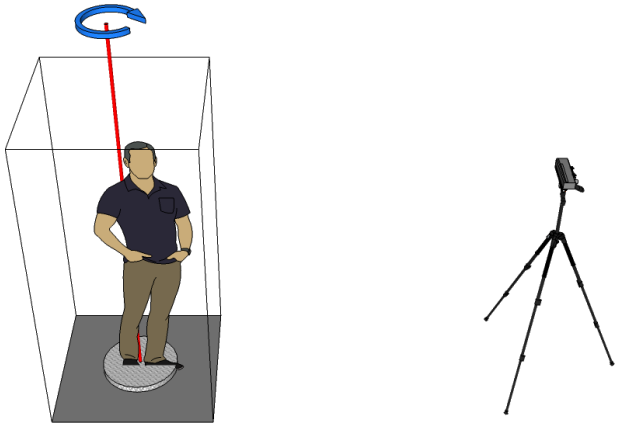


Figure 5.5: Scanner turntable with drawn axis of rotation

If we could have these parameters in our algorithm, we could manually pre align the point clouds for improved robustness. Several tests revealed this strategy to be a really good idea, as shown in Figure X, even with rough estimates of the axis of rotation of the turntable and angle of rotation between scans. They certainly gave a much better starting position for the rest of the registration process.

(a) Raw input clouds **(b)** Rotation compensated

Figure 5.6: Results on the application of rotation compensation

5.3.2 Denoising and Downsampling

In order to improve the registration step performance, a noise reduction and outlier removal step was added in the algorithm. The main goal in this step implies the identification and discarding of those samples from the point clouds that can mess with 3D registration distance error computations. This points are generally asso-

ciated with different kinds of noise and boundary and steep surfaces distortion. In Figure 5.7, an example of a noisy point cloud it is shown.

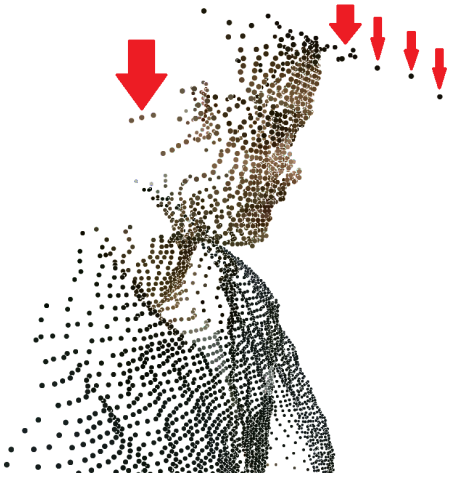


Figure 5.7: Noisy point cloud. Red arrows highlights outliers.

To tackle this problem, statistical outlier removal was used by means of a k-nearest neighbours approach. Here, k-points surrounding a specific point of the cloud are analyzed. The statistical parameters of the sample distribution are estimated (mean and standard deviation), and those points falling out of m-times the standard deviations are removed. Hence, the algorithm requires setting the values k and m for the thresholding. In our implementation, values k=8 and m=2.5 were used.

After cleaning the raw data, point clouds were downsampled using Voxel Grid, the same strategy used in registration approach 1 and described above (section 2).

The result of this two consecutive steps consist on an informative, free from noise downsampled point cloud. In our implementation, values k and m were tuned based on empirical results, accounting to a correct balance between boundaries definition and point-cloud density homogeneity.

5.3.3 Pairwise Initial Rough Alignment

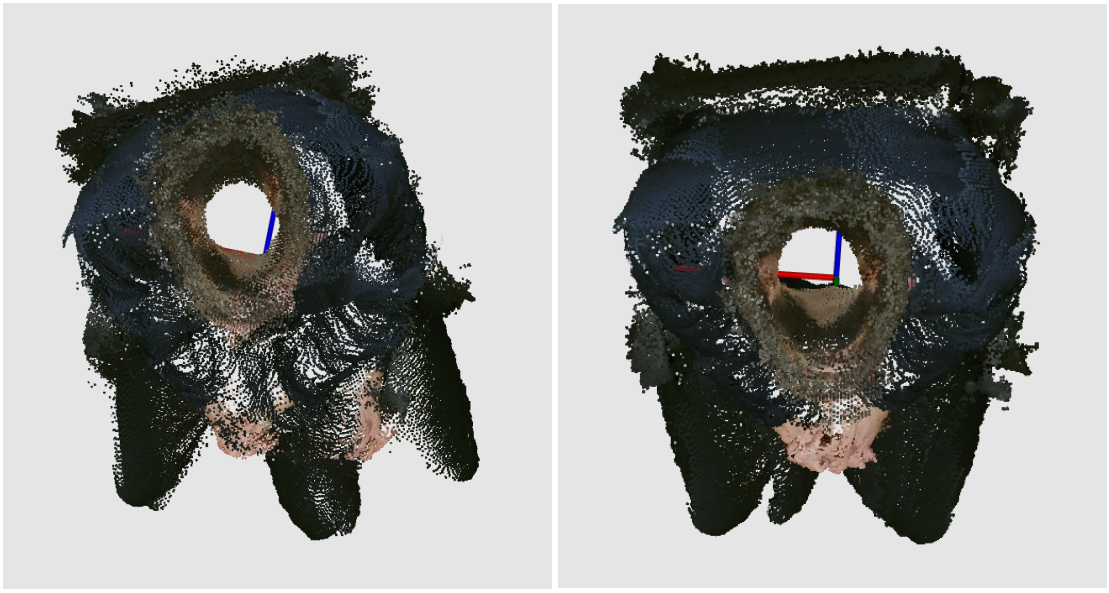
After this pre-processing steps, the rough registration transform is estimated with a similar setup than that of the first approach. We first get a list of correspondent points between the new downsampled point cloud and the previous registered downsampled based this time solely on the normals as descriptors. Then a consensus correspondence rejection algorithm is applied to discard possible outliers. Afterwards,, a linear system is solved that gives the best approximate transform to map each correspondence to the same frame of reference using SVD for efficient computation.

Finally, ICP is applied to the result of the previous step to get a finer alignment. The obtained transform is multiplied with the one from the previous step and applied to the original not-downsampled point cloud. At this point the rough initial alignment is done and we either wait for another pointcloud to register or for the user to start the post-processing.

5.3.4 Post-Processing

Closed Loop Drift Correction

The first stage of the algorithm implies pairwise registration of the consecutive point clouds that capture all views of an object. As stated earlier, this way of registering accumulates small errors that create errors when the rotation of the turntable is finishing the full rotation.



(a) Roughly Aligned **(b)** Closed Loop

Figure 5.8: Results on the application of loop closing

As we can see in Figure 5.8 (a), the small error in pairwise registration have induced a significant error when the view returns to the initial position. We can see that the legs are significantly misaligned, causing the appearance of having three legs and two pairs of hands.

This is a common problem in robotics and to solve this issue we used a variant of the well-known SLAM (simultaneous localization and mapping) technique through the ELCH (Explicit Loop Closing Heuristic) implementation in PCL. The result by slightly modifying the initial transforms until the last point cloud matches with the first one is a better global registration. However, this small modification induce at the same time small local misalignments as a consequence, which need further processing since ELCH goal is to provide good global registration and not so much good registration at the local level.

Incremental ICP

To solve inaccuracies and misalignment from the relaxation of the bounds from the loop closing algorithm and to avoid again the typical drift of pairwise registration we used the PCL implementation of Incremental ICP, which performs ICP between the new point cloud and a merge of all previously aligned clouds. Since the global registration provided by ELCH is quite good we can put a really short correspondences search distance to be certain that ICP will land on the global minima and give the best registration possible.

5.4 Class structure and operation

5.4.1 Public interface and using the class

There exist few basic functions through which all the interaction and operation with the class are made:

1. To construct the class two basic parameters are needed:

   `bool registerInRealTime`: Specifies if the rough initial registration will be made at the time of adding the point clouds or in the post processing stage.

   `pcl::PointWithViewpoint scannerRotationAxis`: which encodes the cartesian point coordinates and rotation euler angles around each axis that define a line, which spatially coincides with the turntable axis of rotation. The value of this parameter is calibrated through visual feedback once the scanner is deployed, before starting to scan.

2. Then the consecutive rotated views to register are input with:

   `addNextPointCloud(inputCloud, degreesRotatedY)`

   Where *inputCloud* is the next rotated cloud and *degreesRotatedY* is the degrees rotated in Y axis with respect to the previous pointcloud.

3. (Optionally) Get the current progress of roughly aligned point clouds for display with:

   `getRoughlyAlignedPC()`

   Which returns the merged roughly aligned point clouds.

4. Finally, when all pointclouds have been acquired to close and finish the registration, the post processing of the scan is made by calling:

   `postProcess_and_getAlignedPC()`

   Which post-processes and returns the final merged aligned result.

IMPORTANT: The registerInRealTime flag should not be changed after starting to add pointclouds.

5.4.2 Class architecture and organization

The class internal operation is made by two types of functions:

- Utility functions: Perform one specific operation, the input is passed through arguments and the result is returned. They do not interact with the class variables.

```
pcl::PointCloud<pcl::PointXYZ>::Ptr
mf_outlierRemovalPC(
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud_in,
  const float meanK=8,
  const float std_dev=2.5);

pcl::PointCloud<pcl::PointXYZRGB>::Ptr
mf_outlierRemovalPC(
  const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloud_in,
  const float meanK=8,
  const float std_dev=2.5);

pcl::PointCloud<pcl::PointXYZ>::Ptr
mf_voxelDownSamplePointCloud(
  const pcl::PointCloud<pcl::PointXYZRGB>::Ptr &cloud_in,
  const float &voxelSideLength);

pcl::PointCloud<pcl::Normal>::Ptr
mf_computeNormals(
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud_in,
  const float &searchRadius);

pcl::CorrespondencesPtr
mf_estimateCorrespondences(
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud1,
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &cloud2,
  const pcl::PointCloud<pcl::Normal>::Ptr &normals1,
  const pcl::PointCloud<pcl::Normal>::Ptr &normals2,
  const double &max_distance);

template <typename PointT>
boost::shared_ptr<pcl::PointCloud<PointT>>
mf_iterativeClosestPointFinalAlignment(
  const boost::shared_ptr<pcl::PointCloud<PointT>> &source,
  const boost::shared_ptr<pcl::PointCloud<PointT>> &target,
  const float &maxCorrespondenceDistance,
  Eigen::Matrix4f &icpTransformation);

pcl::PointCloud<pcl::PointXYZ>::Ptr
mf_SVDInitialAlignment(
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &source,
  const pcl::PointCloud<pcl::PointXYZ>::Ptr &target,
  pcl::CorrespondencesPtr correspondences,
  Eigen::Matrix4f &transformation_matrix);
```

- Methods: Perform compound operations using a combination of utility functions. They use the class variables as input and output.

```
bool mf_processCorrespondencesSVDICP();
bool mf_processInPostWithICP();
```

This allows us to change and experiment with different setups and combinations of algorithms without having to make major changes in the operation of the class.

5.4.3 Internal operation overview

Default Parameters in constructor

- The parameter mv_voxelSideLength is set by default to 0.015 to provide uniform density pointclouds of about 30% the size of the original pointclouds.
- The radius search for normal estimation mv_normalRadiusSearch is set by default to 0.05 following the uniform density in voxels of 0.015 provides an optimal neighbourhood search size
- The parameter mv_SVD_MaxDistance is set to 0.15 meters to provide a big enough margin for poorly prealigned pointclouds, but not big enough to risk having local minima at sight.
- The parameter mv_ICP_MaxCorrespondenceDistance is set even more restrictive at 0.05 meters since SVD should provide a good alignment and ICP is only for fine registration, the short search also reduces the execution time considerably as less possible correspondences have to be evaluated.
- Finally, mv_ICPPost_MaxCorrespondanceDistance is set at 0.03 meters for the same reason as before, to gradually decrease the search distance for finer corrections and better execution time.

Extended operation diagrams

The final implementation of the described algorithm for rough registration and the finer post-processing registration are summarised in the diagrams depicted below.

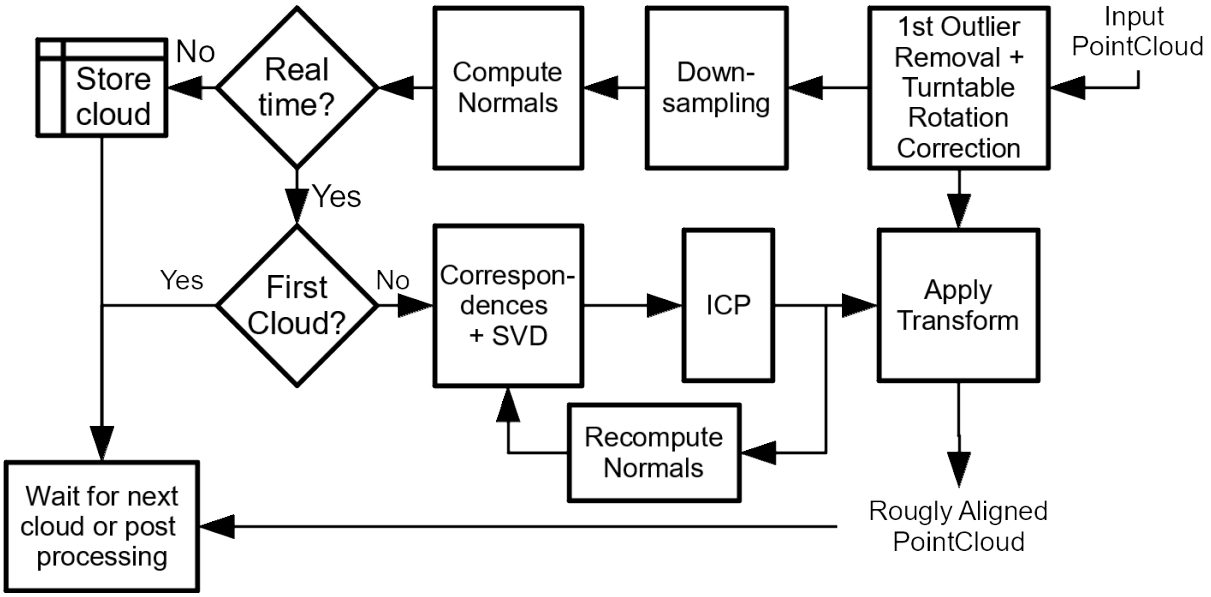


Figure 5.9: Extended internal operation of the ScanRegistration rough registration.

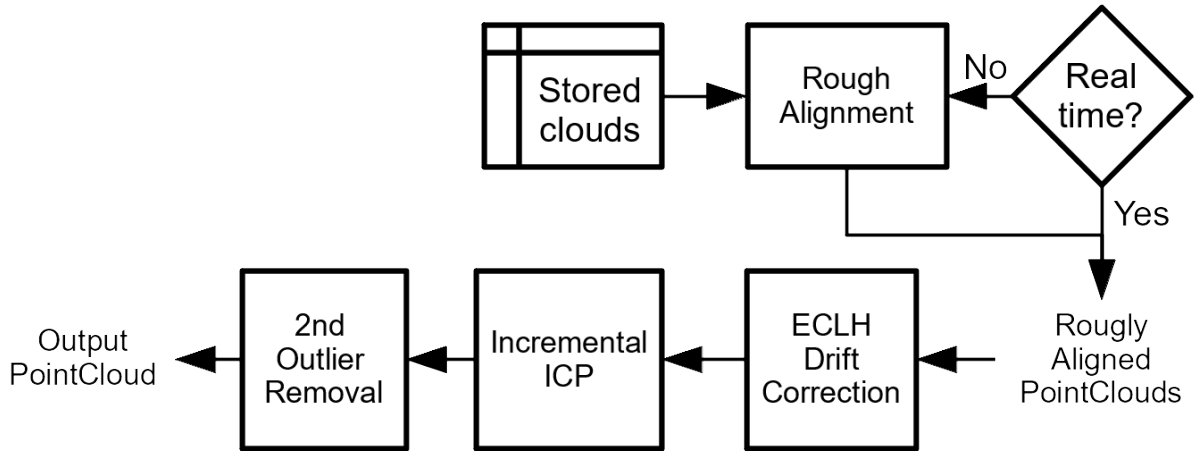


Figure 5.10: Extended internal operation of the ScanRegistration class post processing.

5.5 Results and conclusions

5.5.1 Results

The results obtained are really good at the global registration but with a lot of room for improvement in local registration, having a good deal noise and artifacts. The artifacts and noise in the attached figures may probably be due illumination issues and the noise in the sensor itself, they will be removed with smoothing filters and other post processing for reconstruction.

Figure 5.11: Results on the application of rotation compensation

5.5.2 Conclusions and Future work

We consider the results of our registration algorithm to be good enough considering the technical difficulty, initial skills and knowledge, available time and mentoring and organisation overhead in the project. We have provided an algorithm that can successfully register 3D objects with as few as 10 frames while providing a rough registration preview in real-time. It is also robust to rough parameter estimation and errors in the pre alignment of some frames. The developed class is modular and flexible for ease of future development, while offering a simple use interface and configurable parameters.

Future work should focus mainly on reducing the noise and misalignment derived from sensor noise and non-rigid object registration. The severity of the sensor noise in the registration process has only been noticed in the final stages of development and it hasnt been fully addressed. Consequently, the outlier removal step and point cloud denoising should be further studied and developed. However, even with really good outlier removal and noise reduction, if the scanned person makes small movements between frames (such as breathing), it will still impact the quality of registration for mesh reconstruction. It is suggested to study the integration of non-rigid fine registration algorithms that can account for this misalignments and can adjust them to create a smooth yet detailed final result.

Chapter 6

3D Reconstruction

6.1 The 3D Reconstruction Problem

After completing the 3D registration we pass to the next step that is the 3D reconstruction
One of the largest challenges in reconstructing surfaces robustly and accurately is dealing with the many anomalies that are present in data that are obtained from the scanning of real-world objects
Before to be able to reconstruct the 3D model we has faced some problems like the noise on the point cloud or high density of point.Inaccuracies in scanning devices createsq noisy data,and point sampling is often non-uniform.
Constructing a surface in the presence of these data anomalies is a difficult problem.In addition scanning technologies have driven a dramatic increase in the size of datasets available for reconstruction, with datasets now exceeding one billion point samples.As a result, space and time efficiency have become critical in the development of effective reconstruction algorithms surface reconstruction
The reconstruction can be summarized like in scheme below:

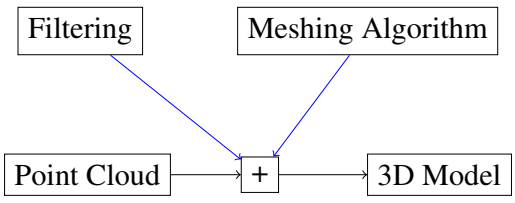
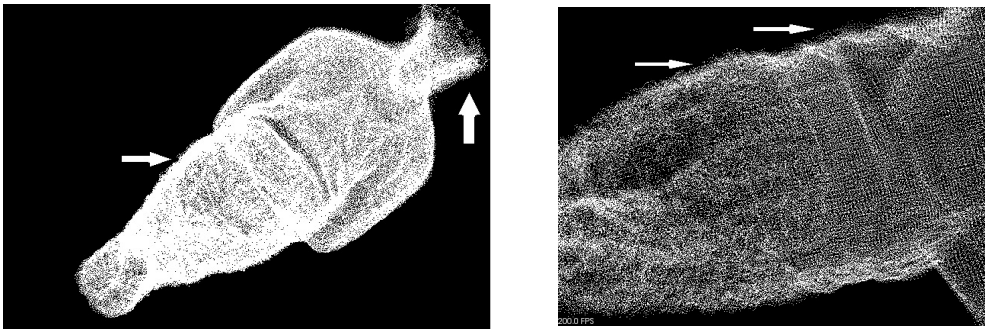


Figure 6.1: 3D Reconstruction

In our specific case the Point Cloud that we receive from the registration class was quite noisy, with a high density of point and we front even another problem ”the extralayer of the surface” for better understand this problem we can see the following image:



(a) high density point cloud **(b)** extralayer of the surface

Figure 6.2: Common issues for 3D reconstruction

The different techniques used for making 3D reconstruction will be discussed and tested from different aspects.This is done in order to find the most optimal reconstruction technique.

6.2 Preparing the cloud

Before going into the implementation of the algorithms : greedy triangulation, marching cube and Poisson,it is necessary to look at the input data that is provided to these algorithms and how it is processed, so before give the point cloud tho this algorithm we have to do some operation on the point cloud:

6.2.1 Filtering

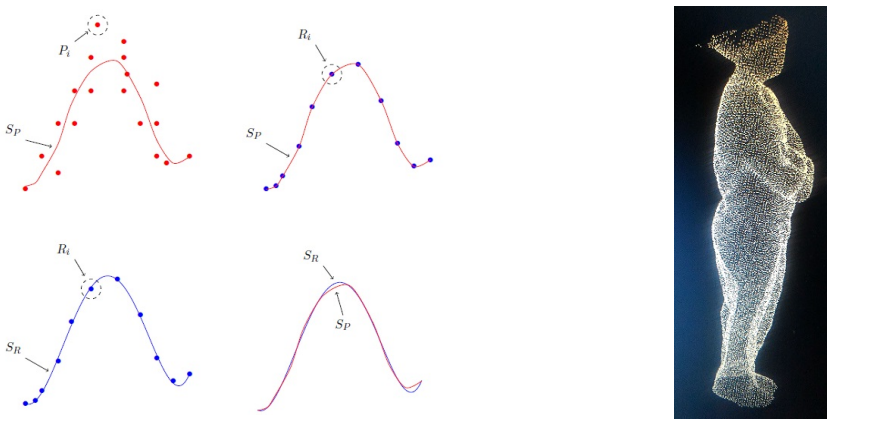
Because the received point cloud it was oversampled first of all we decide to elaborate on it some filtering operation:

- PassThrough this filter iterates through the entire input Point Cloud,and automatically filtering the non-finite points and the points outside to one presetted interval specified
- The VoxelGrid class creates a 3D voxel grid, as a set of tiny 3D boxes in space, over the input point cloud data. Then, in each voxel all the points present will be approximated 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.

6.2.2 MLS smoothing

The first step taken after preparing the cloud is to smooth it using Moving Least Squares(MLS). As explained by [Alexa, Behr, Cohen-Or, Fleishman, Levin, and Silva, 2003] the idea behind MLS is to have a data set of points $P = \{pi\}$ that

constructs a smooth surface SP , however, instead of using the original data set of points P, a reduced set $R = \{ri\}$ is created and a new surface SR is defined.



(a) MLS

(b) Point Cloud after MlS

Figure 6.3: MLS Process

The MLS smoothing is done before any reconstruction algorithm is applied. Another smoothing method is also used, however only applied after the reconstruction is complete.

6.2.3 Normalization

In order to use the cloud properly for the reconstruction, a cloud with both x, y and z values and normal information is needed. Many methods of surface reconstructions require normal associated with the point clouds, the normal can either be oriented or unoriented. Normals that do not have a direction are called unoriented normal, which means it is to be expected the normal to be pointing either on the inside or outside of the surface. This sort of information is useful to determine planar regions in a point cloud, projecting a point onto an approximated surface or achieving covariance matrix. An oriented normal on the other hand have consistent directions and it is known which point outside or inside of the surface.[Bergeret al.].
The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem. The solution for estimating the surface normal is therefore reduced to an analysis of the eigenvectors and eigenvalues (or PCA Principal Component Analysis) of a covariance matrix created from the nearest neighbours of the query point.`pointcloud.org`.

6.3 Meshing algorithms

At this point the point cloud is ready for the surface reconstruction.
As I mentioned before there are many algorithms for 3D reconstruction. In our work

we compare the reconstruction of 3 of them.

6.3.1 Greedy Triangulation

This is an iterative algorithm developed by ”Zoltan Csaba Marton”. The focus of Fast Triangulation(FT) is to keep a list of possible points that can be connected to create a mesh.
FT is obtained by adding short compatible edges between points of the cloud, where these edges cannon cross previously formed edges between others point. Fast Triangulation contains accurate and computationally efficient triangulations when the points are planar. However, a big difference can be noticed when the triangulations are non-planar.
The method works by maintaining a list of points from which the mesh can be grown and extending it until all possible points are connected. It can deal with unorganized points, coming from one or multiple scans, and having multiple connected parts. It works best if the surface is locally smooth and there are smooth transitions between areas with different point densities.
In the Point Cloud Library, fast triangulation works locally, however it allows the specification of the required features to be focused on. Such parameters are neighbourhood size of the search, the search radius and the angle between surfaces. The surface created by this algorithm is not continuous and can contain holes, this was one of the reason that lead us to not use FT for reconstruct the meshes.
In the image below is illustrated the result obtained:

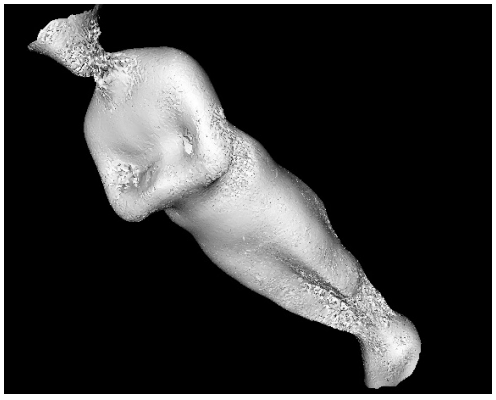


Figure 6.4: Triangulation

6.3.2 Poisson

There are a multitude of surface reconstruction techniques, however each one of them poses a number of difficulties when it is applied to data points. But because watertight mesh was one of the prefixed setted point,at the end, we decided to use Poisson Algorithm Reconstruction. The Poisson reconstruction uses the Poisson equation, which is also known to be used in systems that perform tone mapping, uid

mechanics and mesh editing. The Poisson reconstruction approached by Michael Kazhdan, Matthew Bolitho and Hugues Hope has set as goal to reconstruct a watertight model. It need in input the normal direction, respect to the surface, for each points of the cloud.
By its nature, this algorithm can only be used with watertight or closed object. This can be problematic when we get the points cloud through Kinect, in fact the acquisition that we make not always respect this Property, as we can see in our model the head is not a close object. In our specific case at the head and feet can be noticed that the Poisson reconstruction connects the regions, which poses some limitations in the use of Poisson reconstruction, even more Poisson reconstruction has a high noisy sensitivity, we can see this in the images below:

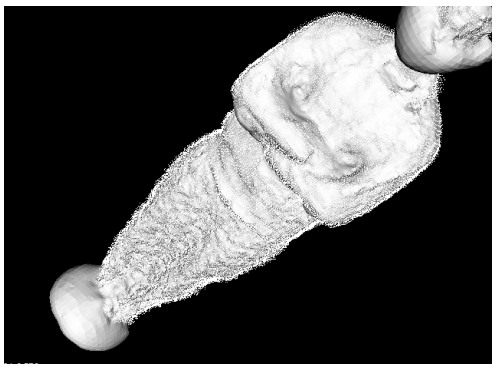
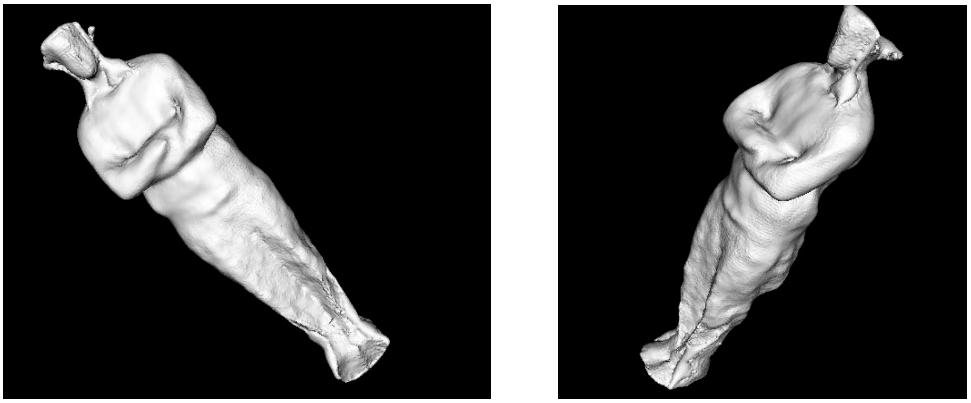


Figure 6.5: Poisson Reconstruction

6.3.3 Postprocessing algorithm Laplacian

The obtained result was still a lot noisy, for having a better result in terms of surface problem we decide to apply a post smoothing method.
The smoothing method used is MeshSmoothingLaplacianVTK. This is a laplacian smoothing method from the VTK library. As described in the documentation for the original VTK smoothing method: *"The effect is to "relax" the mesh, making the cells better shaped and the vertices more evenly distributed". [Visualization Toolkit*, 2015].



(a) Poisson after Laplacian Filter **(b)** Poisson after Laplacian Filter

Figure 6.6: Laplacian Filtering

6.4 Conclusion

A criteria for the success of the various reconstruction algorithms is that they are fast and reliable. For that purpose, the computation time was observed with various settings and each mesh is manually looked at for checking how reliable and accurate the mesh is.
Based on the used reconstruction methods accuracy level, time, computational complexity, it can be assessed that the Poisson reconstruction present the most positive features, however even if Triangulation gaves good result in time of accuracy and speed, the image was no watertight meshes.
Therefore, based on the reconstruction method being applied using PCL it can be assessed that Poisson would provide the most optimal mesh, even tough the computation time is still a little high.

Chapter 7

Graphical User Interface

This chapter will introduce the Graphic User Interface (GUI) developed, its functionality and an analysis of the structure and the different mechanisms. The GUI has been divided into 2 independent windows, namely *Edit Window* and *Scan Window*.

7.1 Edit Window

The *Edit Window* (figure 7.1) is the core of the GUI. It is the window from where the user has the options to open the scan window, import and export point clouds and meshes, register pointclouds and generate mesh.

Figure 7.1: Preview of the Edit Window

The functioning of the window is based on its 4 different components which will be detailed below.

7.1.1 Menu bar

The menu bar (figure 7.2), located in the upper left corner, give the possibility to the user to import and export points clouds and meshes.

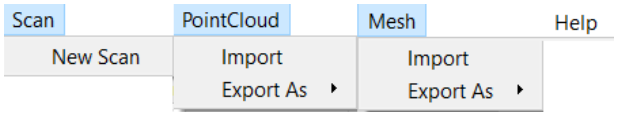


Figure 7.2: Preview of the Menu Bar

The point clouds, under the format .ply or .pcd, could be imported from the local computer or directly from the scanning interface (Section 7.2).
The meshes, under the format .stl and .vtk could be also import from the local computer or generating using the module *Point Cloud Operations*.By importing from the local computer, the possibility to import files at the same time has been added.

To export a point cloud or a mesh, it's required to specify the format of the file by selecting it from the sub menu *Export As*.

7.1.2 Point Cloud Explorer

The Point Cloud Explorer (figure 7.3) has been designed from a QTabWidget containing 3 QListWidget, each associate to a specific type of file: Point Cloud, Registered Point Cloud and Mesh.

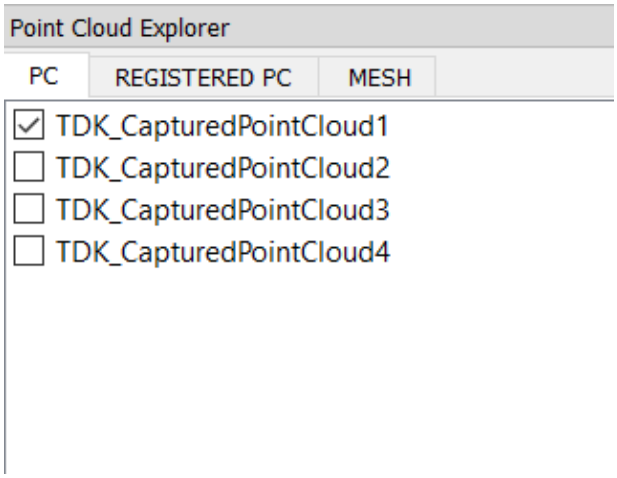


Figure 7.3: Preview of the Point Cloud Explorer

Each time a file is imported or generated, a QListWidgetItem is automatically added to the QListItem corresponding to the type of the file.

In order to process operations on the point clouds or to show them on the *Point Cloud Visulizer widget*, the QListWidgets give the possibility to the user to select multiples files by providing an associate check box to each QListWidgetItem.

7.1.3 Point Cloud Operations

Different operations can be processed to the point clouds such as the registration between several point clouds and the generation of a mesh.

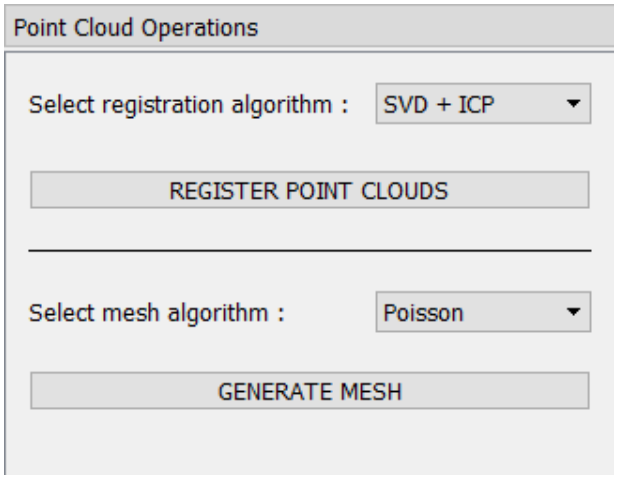


Figure 7.4: Preview of the Point Cloud Operations

The Point Cloud Operations module (figure 7.4) allows the user to choose the algorithms of registration and generation of mesh.
By default, these algorithms are respectively Singular Value Decomposition and Iterative Closest Point for the registration and Poisson Surface Reconstruction for the generation of mesh. When one of these operations is applied, the generated file is directly added to the QListWidget of the *Point Cloud Explorer*.

7.1.4 Point Cloud Visualizer

The Point Cloud Visualizer (figure 7.5 module is based on a QVTKWidget which give us the possibility to display a point cloud or a mesh.

Each time a QListWidgetItem of the *Point Cloud Explorer* is checked or unchecked, the point cloud or mesh associated is added or removed to the QVTKWidget. Multiples files can then be displayed at the same time.
In order to have a better visualization, it's possible to zoom in / zoom out and rotate around the axis [x, y, z].

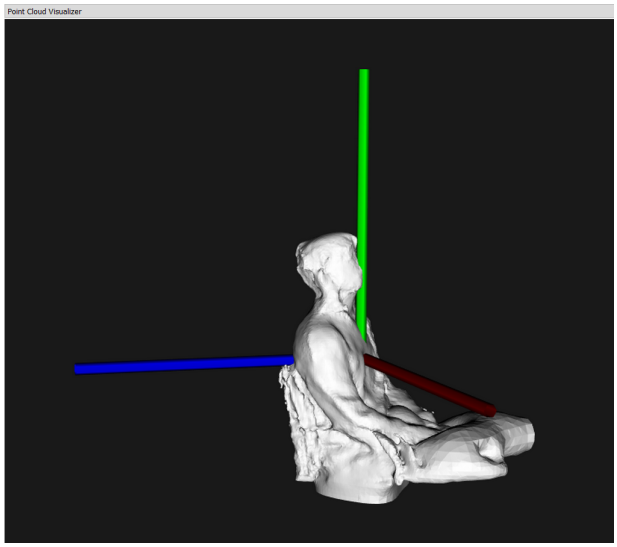


Figure 7.5: Preview of the Point Cloud Visualizer while displaying a mesh

7.2 Scan Window

From the Scan Window (figure 7.6, the user will have the possibility to interact directly with the sensor and the platform to get a series of point clouds, which will be display on the Point Cloud Visualizer, similar to the one of the Edit Window (section 7.1.4).

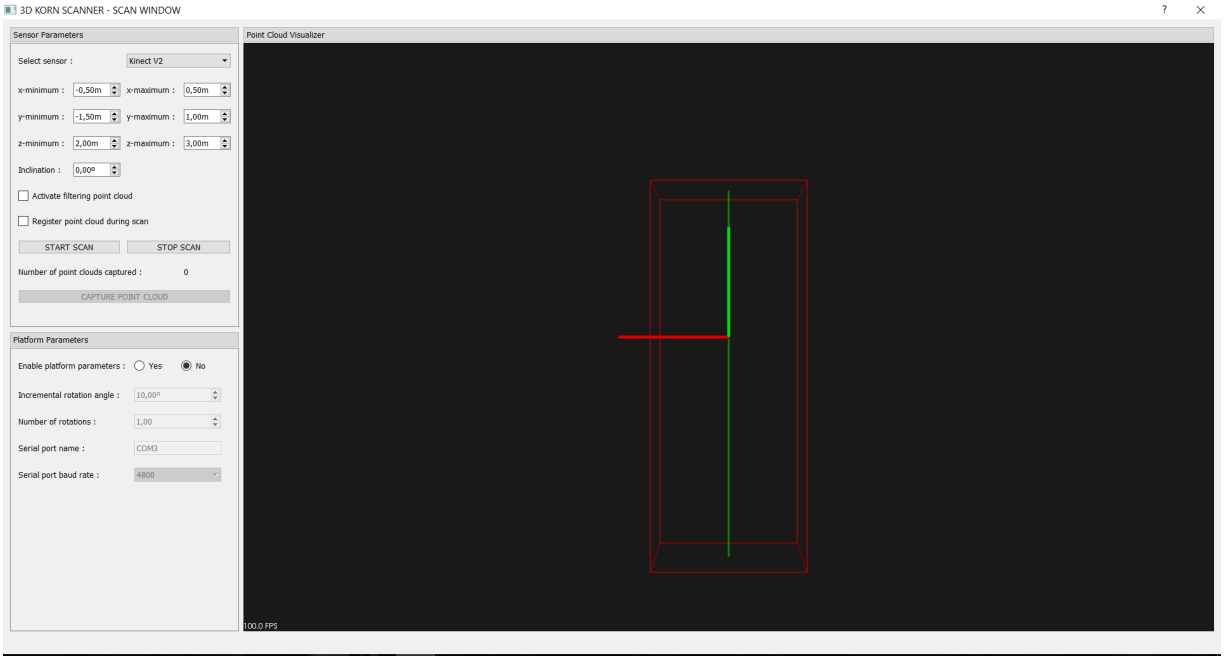


Figure 7.6: Preview of the Scan Window

7.2.1 Platform Parameters

The user has the possibility to tune parameters relative to the platform such as incremental angle of rotation, the number of complete rotation, or to the mode of

acquisition, the name and the baud rate of the serial port.

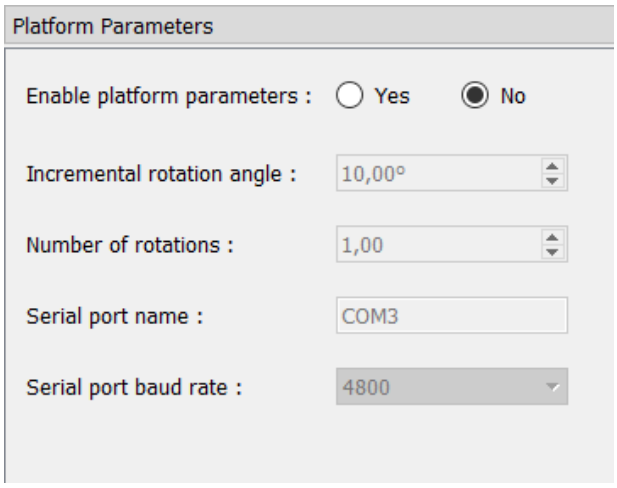


Figure 7.7: Preview of the Platform Parameters

7.2.2 Sensor Parameters

Due to the range of sensor which could be used for the acquisition of point clouds, like Kinect V2 or R200, a module has been added to let the user tune the parameter of his sensor (figure 7.8).

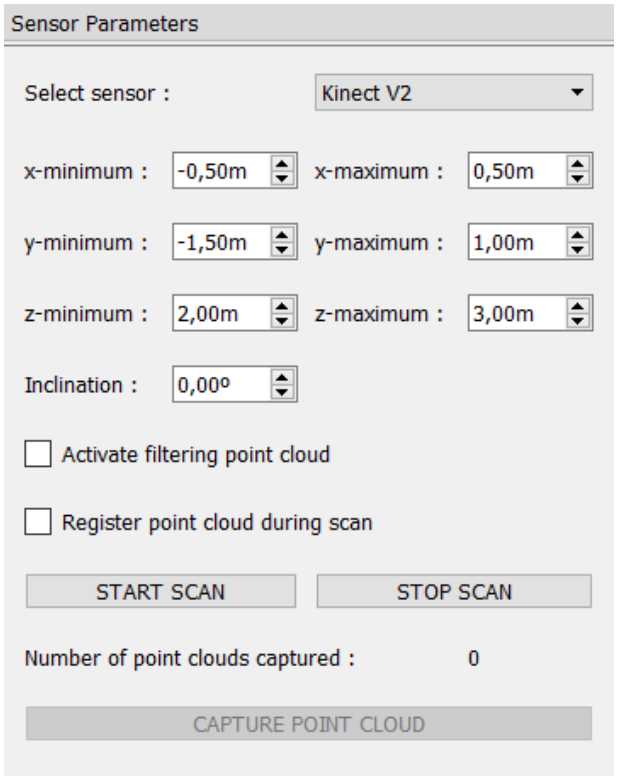


Figure 7.8: Preview of the Sensor Parameters

Besides the possibility to select the sensor, the user can define the delimitations

and orientation of the scanned area. He can also decide to apply directly the filtering and registration processes.

To perform a 3D scan, the user has to manually start the procedure. When the scan is completed, the series of point clouds are displayed on the Point Cloud Visualizer module and the number of point clouds captured is updated. Using these informations, the user could verify the quality of the scan and decide to remake the scan or move to the Edit Window (section 7.1)

Chapter 8

Pointcloud Cropping

8.1 Introduction

This is a class that allows the user to crop specific ranges of points: the general idea behind it can be simplified as PassThrough filter, that may be used to maintain or not specific part of the point cloud.
The use of this class has been thought just after the 3D registration: the user has already decided the object to scan and the scanning process has been completed. Before creating a watertight mesh with the TDK_Operations class, it can be useful to crop specific parts of the scanned objects (point cloud). For making an example, during the acquisition we could have included parts which we are not interested in (for example the rotating basis where the scanned object is collocated, or something in the background) so we want to eliminate them; and this is the goal of the TDK_Cropping class functions.
The use of this class is not a mandatory step during the entire work flow, but it depends on the decisions of the user.

8.2 First Implementation

The user can select to crop points along all the three main axes (x, y and z). The main idea is that for each axis, he or she can specify a range of points, choosing a maximum and minimum value: the points that are outside that range are discarded and the other are maintain. So it appears clear, as we already mentioned, that the general idea of this class is similar to a Passthrough filter which maintain the points inside a range that the user can specify. The filtering process can be operated along all the axes and it is reversible: if we want to restore a part that we have cropped, we can do that.

To make the process more precise and easy for the user, we implemented three sliders in a specific GUI part (one for each axis). The slider are very easy to use and

allow the user to choose graphically the parts to discard. We decided to use this approach, instead of inserting manually numerical values of the range to crop, because it is more user-friendly.
One important aspect of the class is that before using the sliders to filter the point cloud volume, the values that the sliders can have is related to the actual size of the point cloud of interest. So it is mandatory to find the specific range of coordinates within our volumes is included. In fact, from the 3D registration we receive a rectangular volume where the point cloud of interest occupies just a part of that. So we decided that the cropping-process should be used not in the whole box but just on the actual points of the volume of interest. For doing that, along each axis we look for the minimum and the maximum values of the points cloud volume. After that, we use these coordinates for setting the initial and final point of cropping: for example, if we want to filter the points along the x-coordinate, we can fix the maximum x components and we can move the x slider in the GUI to crop all the points that are between the fixed maximum value and the actual value specified by the slider.

In the first idea, we decided to crop just fixing a maximum of one axis and moving the specific slider. Then we thought that it would be more useful to give the user the opportunity to specify to fix the maximum or the minimum. The reason of this is obvious if we think that the points to be eliminated may be on the top or on the bottom of a specific axis in the volume: so it is very important to let the user free to select from which part he or she wants start to crop.

8.3 Final Implementation

In the final class we find seven functions, divided in public and private as follows:

- *public functions*
    (a) TDK_FindMaxMin
    (b) TDK_CubeCroppings
    (c) TDK_CropPlanesPointsFilling
    (d) TDK_SliderRemapedCroppings
    (e) TDK_SliderRemapedCroppingswithFilling
- *private functions*
    (a) TDK_RandomPlane
    (b) TDK_FillingCroppedPlane

The TDK_FindMaxMin function is used for finding the maximum and minimum coordinates of the point cloud of interest. As we already explained, this step

has been implemented because it is users interest to crop the point cloud, and for doing that we have first to understand the maximum and minimum values (x, y and z) of the volume of interest.
This function is a void function that takes as input a pointer to a point cloud of type PointXYZ, and six double variables that are used also to store the maximum and the minimum of the volume on each axis (for example, minx and max are used to store the max. and min. x-components of our point cloud). Both the pointers and the six variables are passed by reference, and it increases the speed of computing; also the pointer to the point cloud is put const because it is not our interest to modify it during the call. The core of this function is a for that check all the points of the point cloud; inside the for there are three if-condition and each of them is used to check the min or the max for each coordinates.

The TDK_SliderRemapedCroppings is used to update the values spaced by the sliders with the actual values of the point cloud. Essentially it enables the sliders used by the user to work properly and to crop in the right position the volume of interest.
This function is a void function that takes as input two pointers to two point cloud of type PointXYZ: one is the input and the second is used to store the result of the cropping. It takes as input one const double variable that is the length (or number of levels) of the slider. It also take six const double variables that are the min. and max value of the sliders (two for each coordinate), then an other const 6 double variables which are the maximum and minimum values of the volume of interest to crop. All these double variables (passed by reference) are used for remap the range and the values taken by the sliders. Inside this function, we have defined other six variables that are used to store the values taken by the slicers which the user have moved for cropping: these double values are actually the result of the movements of the sliders and their values specify the limits of the cropping. Inside this function we call the function TDK_CubeCroppings, that it is for cropping the point cloud.

The TDK_CubeCroppings is finally the function that performs the cropping of the volume of interest. We have already said that the cropping process is actually identical to performing a PassThorugh filter: in the body of the function we define a PassThrough filter, used for cropping on each axis.
This function is void function and its input are two pointers to point clouds of the type PointXYZ: one is passed as constant because it the input of the filtering process and it is not our interest to modify it, the second is the pointer to the point cloud used to store the output of the cropping. Then there are other six double variables, passed as constant and by reference: these are the values assumed by the sliders (recall to the previous function). Inside the function we define a PassThrough element of type PointXYZ, and for each axis we indicate the field we want to crop (i.e., the

specific components that can be x, y or z). Then we set two limits of the cropping, for example minx and max for the x-axis: in this way we are saying to the filter to maintain all the points that are included in this range of values. Then we set in input point cloud, and we specify where we want to store the output of the filtering. This formalism is repeated three times, one for each axis.

(a) Original pointcloud

(b) Cropped pointcloud

Figure 8.1: Pointcloud cropping

We decided to provide another type of cropping. This variant does the same job of previous function, but it provides also a filling process of the area that has been cut. In particular, when the slider are moved for filtering the points on the point cloud of interest, they leave an empty space at the area of the volume they have cropped. This is due to the fact that the point clouds of our interest are points clouds which have points in the perimeter of the object, but the internal part of them is empty. So when we tried to cut this type of point clouds, we faced the problems that the cropping process left empty the area of the volume of interest on the coordinates we decide to stop the filtering. So we decided to provide another version of the cropping that contains also the function to fill the empty space left by the filtering process. Moreover, we thought to maintain both the versions, because there may be cases when one is preferable to the other and vice versa.

The TDK_SliderRemapedCroppingWithFilling is a function that it is identical to the TDK_SliderRemapedCroppings. The only difference is that we call the function TDK_CropPlanesPointsFilling.

The TDK_CropPlanesPointsFilling is a function that is used for combining the cropping with the filling process of the empty space left by the filtering. This function is void function and its input are two pointers to point clouds of the type PointXYZ: one is passed as constant because it is the input of the filtering process and it is not our interest to modify it, the second is the pointer to the point cloud used to store the output of the cropping. Then there are other six double variables, passed as constant and by reference: these are the values assumed by the sliders.

Then there are also other six double variables which are the maximum and the minimum values of the volumes of interest. Inside the function two others pointer to point cloud of the type PointXYZ are defined, then there are two integer variables, one defined as ”int numberofpoints ” and another ”int value ”. Then we define six if conditions, each one used for the maximum or the minimum values assumed by each slider. Lets take for example the first one, where the condition is true when the value assumed by the slider is less then the maximum values of the point cloud. If this condition is true, we enter inside the if and set value equal to the value assumed by the slider (the same value that is used in its own if condition). Then, three other functions are called : TDK_RandomPlane, TDK_CubeCroppings (already defined) and TDK_FillingCroppedPlane. These functions are used for creating the a random plane and for cropping the entire volume of interest plus the new random plane, according to a specific shape.

The TDK_RandomPlane is a function used to create a new plane with one coordinate fixed by the slider.
This function takes as input a pointer to the point cloud defined in the TDK_CropPlanesPointsFillings: it is used to store the plane that is going to be created. Then it takes the integers number of points and value (both defined previously), then six double variables which are the max. and min. of the volume of interest, and an integer that specify the axis which the slicer has been moved on. Inside this function there is a switch with three cases: each case is chosen according to the axis specified by the integer value in the call of this function. In each case we perform a for , repeated numberofpoints times. Inside the for, the new plane is created inserting points. Each point is putted in a 3D location, whose coordinates are given by a fixed component (specified by the int value) and two other random components.

The TDK_FillingCroppedPlane is a function that is used to crop the random plane previously created, according to the shape of the volume of interest. In particular, this function takes as input the cropped volume of interest for obtaining the shape of it, and then it uses these informations to crop the random plane, that is another input of this function. All of this is conducted creating a convex set with the points of cropped point cloud, and then cropping it.
This function takes as input the pointer to the cropped point cloud (obtained previously applying the same TDK_CubeCroppings), the pointer to the random plane created before and another pointer to a point cloud of type PointXYZ that is used for storing the output of this function. Inside the function, we allocate memory for another point cloud. Then we create an empty element of type ConvexHull applied to a PointXYZ type. It takes as input the cropped point cloud, we set the dimension equal to three (we specify that we are dealing with a 3D volume), we create a vector that is going to store the vertices of the shape of the cropped point cloud. Then

we compute (i.e,reconstruct) a convex hull for all the points given and we store the resultant points lying on the convex hull in a new pointer to a point cloud of type PointXYZ and we save also the resultant convex hill polygons (as a set of vertices) in the vector previously created.
After the creation of the convex hull, we consider the random plane of points that we have created previously. Firstly, we create an element CropHull that is used for filtering the points that lie outside the 2D closed polygon that we have generated using the convex hull. So we set as input the random plane, then we specify the point cloud and the vertices the hull refers to. Then we set the condition to crop outside the specific shape, and then we save the output of this cropping using a pointer. Then we save all in the point cloud passed by pointer as input to the function.

Chapter 9

Conclusion and Future Scope

9.1 Conclusion

The main goal of the project has been reasonably achieved in spite of critical deadline and large team. The application is considered to be robust to handle changes both in front end UI and back end algorithms and libraries. Additional efforts were made to achieve tasks such as to make the application compatible with both the sensors provided and automatic pointcloud capture by interfacing with the encoder.

9.2 Future Scope

The project has huge scope for future enhancements including :

- Improving mesh generation algorithm
- Crop functionality in UI
- Multiple rotations to scan the person completely and without losing data
- Noise removal

References

[1] *A survey of rigid 3D pointcloud registration algorithms*; Bellekens, B., Spruyt, V., Berkvens, R., Weyn, M.

[2] *Fast Point Feature Histograms (FPFH) for 3D Registration*; R. B. Rusu, N. Blodow, and M. Beetz

[3] *KinectFusion: Real-Time Dense Surface Mapping and Tracking*;

[4] *DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time*;

[5] http://doc.qt.io/

[6] http://pointclouds.org/documentation/

[7] http://www.cplusplus.com/

[8] https://github.com/UnaNancyOwen/KinectGrabber

[9] https://software.intel.com/sites/landingpage/realsense/camera-sdk/v1.1/documentation/html/index.html?doc_devguide_introduction.html

[10] https://codeyarns.com/2015/11/26/how-to-use-kinect-v2-on-windows/

[11] https://www.udacity.com/course/how-to-use-git-and-github--ud775

[12] http://www.vtk.org/Wiki/VTK/Tutorials/QtSetup

[13] https://en.wikipedia.org/w/index.php?title=Point_set_registration&oldid=732521446