Faculty of Engineering and Information Technology

Human Body 3D Scanner (Virtual me)

Esteban Gabriel Andrade Zambrano

Student Number: 12824583

Project Number: AUT-21-04017

Major: Mechanical and Mechatronics Engineering

Supervisor: Dr. Teresa Vidal Calleja

A 12 Credit Point Project submitted in partial fulfilment of the requirement for the Degree of

Bachelor of Engineering

June 3, 2021

Statement of Originality

I, Esteban Gabriel Andrade Zambrano, declare that I am the sole author of this report, that I have not used fragments of text from other sources without proper acknowledgment, that theories, results and designs of others that I have incorporated into my report have been appropriately referenced and all sources of assistance have been acknowledged.

Refer to Appendix for the signed statement of originality

Signature	Date

Abstract

Human Body 3D Scanner (Virtual me) (12cp)
Esteban Gabriel Andrade Zambrano - AUT-21-04017
Supervisor: Dr. Teresa Vidal Calleja
Major: Mechanical and Mechatronics Engineering.
Objective: Process recorded sensor data from a human body and create a scaled model of the scanned subject that can be used for clothing fitting.

The capability of scanning distinctive models is critical for many industries ranging from fashion to medical and manufacturing. Nevertheless, many of the current implementations are extremely expensive. Hence it imposes a barrier for the technology. Similarly, many of the existing applications do not create highly accurate models, and on various occasions, these models would be discarded. Therefore, this project will create both hardware and software implementation for a human 3D scanner with a moderate budget as well as accurate results in order to be able to produce accurate and scaled models of the scanned subject. The proposed method for data acquisition will be with a custom rig. The rig will be a set of poles with cameras that will capture images from the person that is inside the rig.

Additionally, a Lidar will be used to capture a PointCloud and an Image of the subject inside the rig. With the acquired images from all the cameras as well as the Lidar, a photogrammetric process will be used to obtain an initial mesh. This mesh is the result of the photogrammetric reconstruction process. Moreover, the initial mesh will be processed with a series of developed algorithms in a pipeline sequence. The pipeline will process the mesh with operations such as plane removal, outliers removal and clustering. The Scaling will use the PointCloud from the Lidar to reference and adjust the Scale and Pose of the processed mesh. Once the mesh has been scaled appropriately with the reference PointCloud, a Poisson algorithm will reconstruct and obtain the final scanned model. The results in this report illustrate how with a constrained data set, it is possible to get an accurate scaled scanned model that can be used in many applications.

Acknowledgements

I would like to acknowledge and thank my supervisor, Dr Teresa Vidal Calleja, for her help and support throughout this project. I would also like to recognise and thank Dr Cedric Le Gentil for his guidance, support and input throughout the project development. Furthermore, I would like to acknowledge Mr Asher Katz for his contributions in developing the rig and the image acquisition process component of the project.
On the other hand, I would also like to acknowledge Dr Mark Liu and the UTS Robotics Institute for the assistance and resources provided for completing this project.

Similarly, I would like to acknowledge the Open Source Community, as assets developed by this community helped in the development and completion of this project.

Finally, I would like to thank my family for their massive and ongoing support throughout this project and my studies. In equal importance, I would like to thank God, who has allowed me to develop and complete this project with his countless blessings.

Contents

List of Figures

List of Tables

Nomenclature

Acronym	Meaning
SFM	Structure From Motion
PCL	Point Cloud Library
PCA	Principal Component Analysis
MLS	Moving Least Squares
RANSAC	Random Sample Consensus
SIFT	Scale-Invariant Feature Transform
AKAZE	Accelerated-Kaze Features
PnP	Perspective-n-Point
ZNCC	Zero Mean Normalized Cross-Correlation
ANN_L2	ApproximateNearest Neighbor Matching
acransac	A-Contrario RANSAC
SGM	Semi Global Matching

Introduction

1.1 Reseach Question

"Human Body 3D Scanner: The development of software for 3D data reconstruction of a Human body scanner with multiple sensors"

1.2 Project Contextualization

The project is based on creating a Human Body 3D scanner. It will have two specific streams that include developing the mechatronic design of a 3D scanner for a human and the software development for 3D data reconstruction. This capstone is based on developing the software for 3D data modelling and reconstruction of the Scanned data.

Similarly, the 3D reconstructed model of the human aims to be utilised with the intention to test different fashion clothing items. This has the intent to adjust the sizing of the clothes fittings based on the Scanned data. The clothing models will adjust automatically depending on the dimensions of the data of the scanned model. The project will have different stages that range from testing various sensors for data acquisition, testing other data stitching frameworks to deploy the software in the 3D scanner mechatronic device.

1.3 Problem Definition

Being able to scan different object and models is crucial for many industries. Many applications are used in the fashion industry, medical industry, manufacturing, etc. However, many of the given implementations are extremely expensive, making the technology inaccessible for many companies and users in general. There are many forms of implementations, as there are multiple technologies in the market that facilitate the process in which several devices and software techniques are used. Nevertheless, no current industry application

maximises the potential use of the Human body 3D models. Many of the challenges faced are that the software implementation for 3D reconstruction of the models is not particularly accurate, therefore creating imperfect models that will need to be discarded on many occasions.

Hence, this project component will contribute and develop the technology to produce software that will create accurate models from the gathered data from the sensors. These models will be utilised to try different fashion items and adapt the size fittings accordingly. With the competition of this project, many stakeholders, industries and institutions could rely on accurate software that will allow the creation of a 3D model of a person or object.

1.4 Background

Human society has the world comprehension of the surrounding world through visual perception. This principle allows differentiating distinctive kinds of shapes, objects, colours, textures, and the spatial pose of the surroundings. Based on this information, it is possible to analyse the number of objects in a determined location, object type, object size, object pose in different coordinate frames. Thus, it impacts how as a society, we interact with objects or scenes. As a result, it is essential to imitate this perception to acquire real-world data in different formats that include:

- RGB images
- Depth images
- 3D point clouds
- Multispectral images
- Laser readings

All these acquire data can be obtained from a wide variety of commercial or industrial sensors. With this data, it will be possible to use computer processing techniques to model the object or scene (Murcia et al., 2018).

1.5 Applications

In recent years, the use of 3D body scanners has gain importance in several industries. It can aid clothes manufacturers within the fashion industry to obtain accurate body measurement data of body dimensions. As mentioned by Sturm et al. (2013), this new technological approach has the potential to alternate the future of the fashion and clothing manufacturing industry.

With the rise of innovation of 3D image reconstruction, the interest to gather precise measurements of humans has risen. Due to the fact that in the clothing industry is extremely important to create better fittings for different shapes of human bodies. Furthermore, virtual try-on solutions have gain popularity in physical and online retail stores (Spahiu et al., 2014).

On the other side, 3D scanners have gained participation in the medical industry. These systems are described as "non-invasive and low cost, " making them appealing for epidemiological surveys and clinical uses (Treleaven and Wells, 2007). The geometrical measurements could be associated with the shape, size, volume, and surface area of the body parts. It could aid to be a sustainable approach to screen children and patients with obesity, deformities, or specific anatomic defects. Therefore, it will ease the diagnose process and allow treatment and monitor medical conditions holistically and improve the life quality of patients with non-invasive tests. The table below illustrates the use of a 3D scanner in the medical field to identify and monitor various medical conditions. From which the diagnose, treatment and monitor procedures will differ based on the acquired data.

Application	Epidemiology	Diagnosis	Treatment	Monitoring
Measurement				
Size	Anthropometric surveys	Growth defects	Scoliosis	Fitness and diet
Shape	Screening	Abdominal shape	Prosthetics	Obesity
Surface area		Lung volume	Drug dosage	Diabetes
Volume			Burns	
Visualization				
Head		Melanomas	Eating disorders	
Chest			Facial reconstruction	
Whole body			Cosmetic surgery	

Table 1.1: 3D Scanning Applications
(Treleaven and Wells, 2007).

1.6 Overview

As mentioned before, this project consists of developing a 3D scanner that is able to scan a person and use that data to fit virtually different clothing items. This capstone project mainly focuses on the software and processing of the data for the 3D reconstruction.

This project aims to develop algorithms and an implementation pipeline that will use the acquired image data as well as the reference point cloud data in order to produce an accurate and appropriately scaled 3D reconstructed model of a person.
The reconstructed and scaled model, is intended to be utilised as the model for testing

virtual cloth fittings and correctly determining the size of the garments. This technology could have a massive impact in the clothing industry and online retailers. This could solve an issue with the returns policy due to incorrect sizing for different customer groups.

1.6.1 Methodology

To achieve the aim, a rig will be used to capture the data. The rig will consist of a set of poles with multiple cameras. Furthermore, a Lidar Intel RealSense L515 will be used to capture another image and the corresponding point cloud used for reference. Once the rig captures the data (twenty-eight images) and the lidar data(one image & one pointcloud), the images will be processed using a photogrammetry approach with Meshroom. The Processed data from Meshroom will produce a preprocessed reconstruction mesh.

After this, the mesh will be processed in the developed algorithmic pipeline, and it will be scaled appropriately with the reference point cloud from the Lidar. Once the mesh has been processed and scaled, it will be reconstructed appropriately. It will generate a final scaled model of the scanned person.

Once the process is completed, the mesh will be saved to disk as a ply file that can be later used in other applications such as virtual clothing fitting.

1.6.2 Structure

This Report will start with a literature review to analyse and fully understand the algorithms and techniques used throughout the development of this technology. It will help to understand the techniques used in the photogrammetric step and the developed algorithm pipeline.

Following this, details will be presented and illustrated as to how the process is carried out and the obtained results from different data sets. It will be entirely demonstrated how to import the data to Meshroom and what parameters are required to add and modify to obtain the most robust reconstruction. There will be a detailed explanation of what each step executes and outputs.

Furthermore, there will be a detailed explanation of the algorithm pipeline for data processing. It will explain what each component is expected to execute and how it was created and wrapped into a framework.

Literature Review

Being able to scan different objects and subjects has been a challenging task for researchers. Getting an accurate spatial location of the objects is crucial for this type of application. The use of 3D point clouds has facilitated this process as it allowed to obtain the following parameters:

- Depth
- Intensity
- Pulse width
- Light echo

This information can be obtained with different kind of sensors. There is a wide variety of off the shelf sensors that can provide 3D point clouds. These sensors could either be stereo or multiview vision cameras, lasers, time-of-flight sensors (TOF) and structured light sensors as stated by Murcia et al. (2018).

Many Scanning devices will use single or multiple of the sensors mentioned above to acquire data. Once the data is obtained, it essential to have a framework for 3D data modelling and reconstruction. The principle behind 3D data reconstruction is obtained with data fusion from RGB-D sensors. This kind of sensors provide three channels in each RGB image (red, green, blue), and the depth images are mapped to each pixel. Based on this data, 3D point clouds could be generated for data reconstruction.

Similarly, many other scanning devices use photogrammetry as a technique to do the reconstruction. In this particular project, Meshroom was used as a 3D photogrammetric reconstruction software.

Moreover, several open-source libraries were used to process and scale the draft mesh. These open-source libraries include PCL and Open3D. From these, several algorithms were implemented which aided in the data processing, and these will be explained below.

2.1 Photogrammetry

Photogrammetry is described as the associated techniques with performing measurements of real-world objects and terrain features from images as mentioned by Aber et al. (2019). Many of the applications include the quantification of distances, volumes, areas, heights, 3D topographic mapping, measuring of objects, extraction of 3D point cloud for surfaces reconstruction as well as the generation of orthophotographs and digital elevation models.(Aber et al., 2019).

In recent years, with the development of technologies pairing computer vision concepts & algorithms and photogrammetry specifically *Structure from Motion-Multiview Stereo (**SfM-MVS**)* has led to significant advances in 3D surface Reconstruction from images (Aber et al., 2019).

The fundamental principle behind all the photogrammetric measurements is associated with the mathematical & geometrical reconstruction of the path of light rays from the object to the sensor camera in the exact moment of data image acquisition. Thus, the fundamental concept of photogrammetry is the understanding of the geometric characteristics of a single photograph.

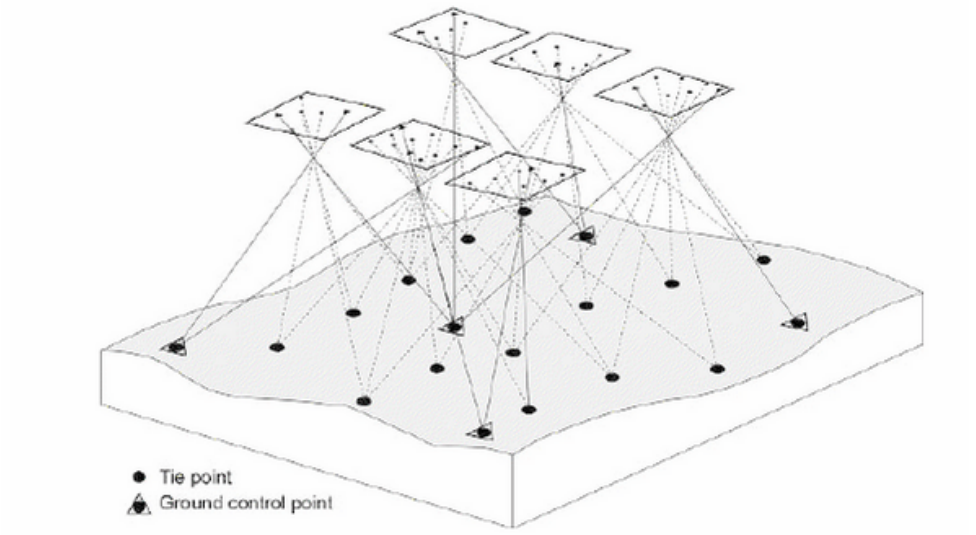


Figure 2.1: Photogrammetry Principle Illustration
(Aber et al., 2019)

2.2 Meshroom

Meshroom is a free, open-source 3D Reconstruction Software based on the AliceVision framework (AliceVision, 2021). AliceVision is a software framework based on photogrammetric computer vision, which focuses on developing 3D Reconstruction and Camera Tracking Algorithms. It aims to provide a robust software foundation with novel computer vision algorithms that can be analysed, tested and reused.
Meshroom was developed as a collobaration project between industry and academia to develop cutting-edge algorithms that are robust and high-quality, crucial for production environments.

Components Pipeline

Meshroom can be downloaded and used in both Windows and Linux, and it will require a powerful CPU to have adequate performance. Also, it will require an Nvidia GPU as it requires running CUDA for different nodes and components. Once meshroom is downloaded, it can run on either platform. When it is executed, the Meshroom GUI will appear, and it will be similar to figure 2.2

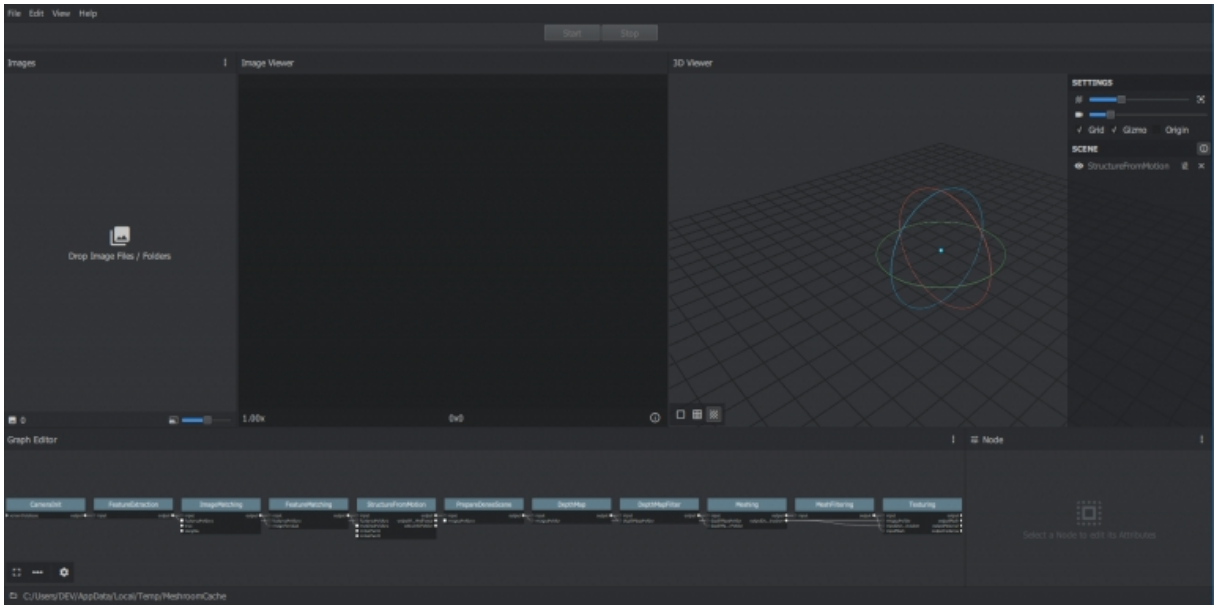


Figure 2.2: Meshroom Graphical User Interface
(AliceVision, 2021)

As mentioned before, meshroom is based on AliceVision's Framework. Therefore it follows a photogrammetric pipeline for 3D reconstruction. This pipeline will have the following components:

- Natural Feature Extraction
- Image Matching
- Features Matching
- Structure from Motion
- Depth maps estimation
- Meshing
- Texturing
- Localization

The pipeline will be included in Meshroom GUI, and it will be used in the nodes. All the components in Meshroom will be embedded into different individual nodes that will be executed and produce individual results that will be sent to the next node along in the pipeline. As the nodes represent specific components of the reconstruction pipeline, these should be executed in a defined order.
The nodes in the defined order to reconstruct a 3D model from images include:

1. Camera Initialization
2. Feature Extraction
3. Image Matching
4. Feature Matching
5. Structure From Motion
6. Prepare Dense Scene
7. Depth Map
8. Depth Map Filter
9. Meshing
10. Mesh Filtering
11. Texturing

2.2.1 Camera Initialization

Camera Initialization or *"CameraInit"* loads the image metadata, sensor information, camera parameters and it will generate viewpoints based on the images. It is possible to use a mixture of cameras & focal lengths. This node will generate groups of intrinsics based on the image's metadata. These groups of intrinsics can be adjusted if the cameras have been fully calibrated. The intrinsic include the camera matrix which includes the focal length (f_x,f_y) and principal point (x_0,y_0).

$$CameraIntrinsics = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

The components in the node are described on table 2.1

Viewpoints Input	viewpoints (1 Element for each loaded image) - ID - Pose ID - Image Path - Intrinsic: Internal Camera Parameters (Intrinsic ID) - Rig (-1 - 200) - Rig Sub-Pose: Rig Sub-Pose Parameters (-1 - 200) - Image Metadata: (list of metadata elements)
Intrinsic Camera Intrinsics	(1 Element for each loaded image) - ID - Initial Focal Length: Initial Guess on the Focal Length - Focal Length: Known/Calibrated Focal Length - Camera Type: pinhole', 'radial1', 'radial3', 'brown', 'fisheye4' - #Make: Camera Make (not included in this build, commented out) - #Model: Camera Model - #Sensor Width: Camera Sensor Width - Width: Image - Width (0-10000) - Height: Image Height (0-10000) - Serial Number: Device Serial Number (camera and lens combined) - Principal Point: X (0-10000) Y(0-10000) - DistortionParams: Distortion Parameters - Locked(True/False): If the camera has been calibrated, the internal camera parameters (intrinsics) can be locked. It should improve robustness and speedup the reconstruction.
Sensor Database	Camera sensor width database path
Default Field Of View	Empirical value for the field of view in degree 45° (0°-180°)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output SfMData File	. . . /cameraInit.sfm

Table 2.1: Meshroom CameraInit Node

2.2.2 Feature Extraction

The key objective of this node is to extract distinctive pixel groups, which are to a certain extend invariant to a change in camera viewpoints during the image acquisition process. Thus, a feature in this scene should have similar features descriptors in all the captures images.

One of the most used feature detection method is **SIFT** *(Scale-invariant feature transform)* algorithm. The objective of SIFT is to extract discriminative patches in an initial image that later can be compared to discriminative patches of a second image, irrespective of scale, rotation and translation (Lowe, 2004). As a relevant detail is only present at a specific scale, the extracted patches are centred around the point of interests. Therefore, the SIFT invariance can deal with image transformations that occur when the viewpoints change during image acquisition.

Based on the representation of one image at different scales, SIFT can compute the Laplacian representation's scale-space maxima. This is a defined image energy-based representation, using the differences of Gaussians. This Maxima is associated with the points of interest in the image. Once this is processed, it samples for each of this maximum a square image patch, whose origin is the maximum and "x" direction is the dominant gradient at the origin as suggested by Lowe (2004) andOtero (2015). For each keypoint, a description of these patches is associated.

The description consists of statistics of gradients which is computed in regions around the key point. The size of the area is defined by the keypoint scale and the orientation by the dominant axis. It is also frequently stored in 128 bits.

The number of extracted features could vary due to texture complexity, from one image to other ones or in different image sections. Hence, a post-filtering step controls the number of extracted features to a specified limit. Furthermore, grid filtering is used to ensure an even repartition in the image.

The components in the node are described on table 2.2

Name	Description
Input	SfMData file.
Describer Types	Describer types used to describe an image. 'sift', 'sift*float', 'sift*upright', 'akaze', 'akaze*liop', 'akaze*mldb', 'cctag3', 'cctag4', 'sift*ocv', 'akaze*ocv'
Describer Preset	Control the ImageDescriber configuration (low, medium, normal, high, ultra). Configuration "ultra" can take a long time!
Force CPU Extraction	Use only CPU feature extraction.
Max Nb Threads	Specifies the maximum number of threads to run simultaneously (0 for automatic mode). (0-24) 0
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace).
Output Folder	Output path for the features and descriptors files (*.feat, *.desc).

Table 2.2: Meshroom FeatureExtraction Node Settings

2.2.3 Image Matching

The principle behind this node is to find images that point to the same areas of interest. Therefore, image retrieval techniques are implemented to find images that share content without the demand of resolving all the detailed feature matches. Hence, the goal is to simplify the image into a compact image descriptor. This allows to efficiently compute the distance between all images descriptors.

A vocabulary tree is one of the widely used methodologies to generate the image descriptor. It works by sharing all the extracted features descriptors into it. Then it performs a classification process, which compares their descriptors to the ones on each node of the vocabulary tree as mentioned by Nistér and Stewénius (2006). Each feature descriptor is associated with one leaf, which can be stored with a standard index (*The index of this leaf in the tree*). Thus, the collection of leaves indices represents the image descriptor (Nistér and Stewénius, 2006).

The components in the node are described on table 2.3

Name	**Description**
Image	SfMData file
Features Folders	Folder(s) containing the extracted features and descriptors
Tree	Input name for the vocabulary tree file ALICEVISION_VOCTREE
Weights	Input name for the weight file, if not provided, the weights will be computed on the database built with the provided set
Minimal Number of Images	Minimal number of images to use the vocabulary tree. If we have fewer features than this threshold, we will compute all matching combinations
Max Descriptors	Limit the number of descriptors you load per image. Zero means no limit
Nb Matches	The number of matches to retrieve for each image (If 0, it will retrieve all the matches) 50 (0-1000)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output List File	Filepath to the output file with the list of selected image pairs

Table 2.3: Meshroom Image Matching Node

2.2.4 Feature Matching

The critical component of the feature matching node is to be able to match all features between potential image pairs. Initially, the node performs a photometric process that creates matches between the descriptors from 2 separate images. For each feature in image "A", a list of potential features in image "B" is generated. The descriptor space is not linear and defined in space, which causes uncertainty in the validity of the matches due to the absolute distance values. Furthermore, an assumption process associates only one valid match in the other image to remove bad candidates. Hence, for each feature descriptor on Image "A", two of the closest descriptors in the image are used with a relative threshold between them which provides a robust criterion as mentioned by Lowe (2004).

This process will provide a photometric list of feature matching candidates. Afterwards, the images features positions are used in a geometric filtering process that uses epipolar geometry in an outlier detection framework, which is RANSAC (*"Random Sample Consensus"*). Moreover, a random selection process uses a small set of feature correspondences and it computers either the fundamental or essential matrix. Once this process is completed, the number of features verifies and validates this model and iterates through the RANSAC framework (Muja and Lowe, 2009).

The components in the node are described on table 2.4

Name	Description
Input	SfMData file
Features Folder	
Features Folders	Folder(s) containing the extracted features and descriptors
Image Pairs List	Path to a file which contains the list of image pairs to match
Describer Types	Describer types used to describe an image **sift**'/ 'sift_float'/ 'sift_upright'/ 'akaze'/ 'akaze_liop'/ 'akaze_mldb'/ 'cctag3'/ 'cctag4'/ 'sift_ocv'/ 'akaze_ocv
Photometric Matching Method	For Scalar based regions descriptor * BRUTE_FORCE_L2: L2 BruteForce matching * ANN_L2: L2 Approximate Nearest Neighbor matching * CASCADE_HASHING_L2: L2 Cascade Hashing matching * FAST_CASCADE_HASHING_L2: L2 Cascade Hashing with precomputed hashed regions For Binary based descriptor * BRUTE_FORCE_HAMMING: BruteForce Hamming matching
Geometric Estimator	Geometric estimator: (acransac: A-Contrario Ransac // loransac: LO-Ransac (only available for fundamental_matrix model)
Geometric Filter Type	Geometric validation method to filter features matches: *fundamental_matrix** // essential_matrix // homography_matrix /// homography_growing // no_filtering'
Distance Ratio	Distance ratio to discard non-meaningful matches 0.8 (0.0 - 1)
Max Iteration	Maximum number of iterations allowed in RANSAC step 2048 (1 - 20000)
Max Matches	Maximum number of matches to keep (0 - 10000)
Save Putative Matches	putative matches (True/False)
Guided Matching	the found model to improve the pairwise correspondences (True/False)
Export Debug Files	debug files (svg/ dot) (True/False)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output Folder	Path to a folder in which computed matches will be stored

Table 2.4: Meshroom Feature Matching Node

2.2.5 Structure From Motion

The Structure From Motion Node carries out the 3D points reconstruction from the input images. The key behind this node is to be able to comprehend the geometric relationship behind all the views from the input images and decipher the rigid scene structure (3D PointCloud) along with the pose (position & orientation) and internal calibration of the cameras as mentioned by Cheng et al. (2014). This is an incremental process pipeline associated with a growing reconstruction process. It initially computes a starting two-view point reconstruction which is iteratively extended by adding new views (Cheng et al., 2014).

It starts by fusing all feature matches between all images and store them in tracks. Each track represents a point in space, which is visible from multiple cameras. Nevertheless, at this stage of the pipeline, many outliers are present, therefore during matches fusion, incorrect tracks are removed (Fischler and Bolles, 1981).

Afterwards, the incremental algorithm chooses the best initial image pair. This choice is crucial for the quality of the final reconstruction. Hence it should provide robust matches and reliable geometric information as suggested by Moulon et al. (2012). Therefore, this image pair maximises the number of matches and the repartition of features correspondences in all images. On the other hand, the angle between the cameras must provide reliable geometric information. Then, it computes the fundamental matrix between these two images considering the first image as the origin of the coordinate system (Kneip et al., 2011). As the pose of the first two cameras is known, it is possible to triangulate the matching 2D features into 3D points.

Once this step finishes, the *Next Best Views Selection* algorithm selects all the images that have enough associations with the 3D reconstructed features. Based on 2D-3D associations, it performs the resectioning of each of the new cameras (Lepetit et al., 2008). The resectioning is a **Perspective-n-Point** algorithm (***PnP***) in a **RANSAC** framework that finds the pose of the camera that validates the features association. Similarly, a non-linear minimisation process is performed on each camera to refine the end pose (Nister, 2004).

From these new camera poses, some tracks become visible by two or more resected cameras. A Bundle Adjustment process starts and refines everything such as: intrinsic and extrinsic parameters of all cameras, as well as the position of all 3D points as suggested by Shah et al. (2014). A filter processes the result of the Bundle Adjustment process and removes all observations that have a high reprojection error or not enough angles between viewpoints.

The new points triangulation process uses more image candidates for the best views selection. It iterates by adding cameras and triangulating new 2D features into 3D points and removes 3D points that become invalid in the process until no more new views can be localised (Shah et al., 2014).

The components in the node are described on table 2.5

Input	**SfMData file**
Features Folder	Folder(s) containing the extracted features and descriptors.
Matches Folders	Folder(s) in which computed matches are stored.
Describer Types	Describer types used to describe an image. 'sift', 'sift*float', 'sift*upright', 'akaze', 'akaze*liop', 'akaze*mldb', 'cctag3', 'cctag4', **'siftocv', 'akazeocv'
Localizer Estimator	Estimator type used to localize cameras (acransac, ransac, lsmeds, loransac, maxconsensus).
Observation Constraint	Observation constraint mode used in the optimization: Basic: Use standard reprojection error in pixel coordinates, Scale: Use reprojection error in pixel coordinates but relative to the feature scale
Localizer Max Ransac Iterations	Maximum number of iterations allowed in ransac step. (1-20000) 4096
Localizer Max Ransac Error	Maximum error (in pixels) allowed for camera localization (resectioning). If set to 0, it will select a threshold according to the localizer estimator used (if ACRansac, it will analyze the input data to select the optimal value). (0.0-100-0) 0.0
Lock Scene Previously Reconstructed	This option is useful for SfM augmentation. Lock previously reconstructed poses and intrinsics.
Local Bundle Adjustment	It reduces the reconstruction time, especially for large datasets (500+ images) by avoiding computation of the Bundle Adjustment on areas that are not changing.
LocalBA Graph Distance	Graph-distance limit to define the Active region in the Local Bundle Adjustment strategy. (2-10) 1
Maximum Number of Matches	Maximum number of matches per image pair (and per feature type). This can be useful to have a quick reconstruction overview. 0 means no limit. (0-50000) 1
Minimum Number of Matches	Minimum number of matches per image pair (and per feature type). This can be useful to have a meaningful reconstruction with accurate keypoints. 0 means no limit. (0-50000) 1
Min Input Track Length	Minimum track length in input of SfM (2-10)
Min Observation For Triangulation	Minimum number of observations to triangulate a point. Set it to 3 (or more) reduces drastically the noise in the point cloud, but the number of final poses is a little bit reduced (from 1.5% to 11% on the tested datasets). (2-10)
Min Angle For Triangulation	Minimum angle for triangulation. (0.1-10) 3.0
Min Angle For Landmark	Minimum angle for landmark. (0.1-10) 2.0
Max Reprojection Error	Maximum reprojection error. (0.1-10) 4.0
Min Angle Initial Pair	Minimum angle for the initial pair. (0.1-10) 5.0
Max Angle Initial Pair	Maximum angle for the initial pair. (0.1-60) 40.0
Use Only Matches From Input Folder	Use only matches from the input matchesFolder parameter. Matches folders previously added to the SfMData file will be ignored.
Use Rig Constraint	Enable/Disable rig constraint.
Force Lock of All Intrinsic Camera Parameters.	Force to keep constant all the intrinsics parameters of the cameras during the reconstruction. This may be helpful if the input cameras are already fully calibrated.
Filter Track Forks	Enable/Disable the track forks removal. A track contains a fork when incoherent matches lead to multiple features in the same image for a single track.
Initial Pair A	Filename of the first image (without path).
Initial Pair B	Filename of the second image (without path).
Inter File Extension	Extension of the intermediate file export. ('.abc', '.ply')
Verbose Level	Verbosity level (fatal, error, warning, info, debug, trace).
Output SfMData File	Path to the output sfmdata file (sfm.abc)
Output SfMData File	Path to the output sfmdata file with cameras (views and poses). (cameras.sfm)
Output Folder	Folder for intermediate reconstruction files and additional reconstruction information files.

Table 2.5: Meshroom StructureFromMotion Node

2.2.6 Prepare Dense Scene

This node undistorts the images and generates EXR images.

The components in the node are described on table 2.6

Name	Description
Input	SfMData file
ImagesFolders	Use images from specific folder(s). Filename should be the same or the image uid.
Output File Type	Output file type for the undistorted images. (jpg, png, tif, exr)
Save Metadata	Save projections and intrinsics information in images metadata (only for .exr images).
Save Matrices Text Files	Save projections and intrinsics information in text files.
Correct images exposure	Apply a correction on images Exposure Value
Verbose Level	[‘fatal’, ‘error’, ‘warning’, ‘info’, ‘debug’, ‘trace’]
Output	MVS Configuration file (desc.Node.internalFolder + ‘mvs.ini)
Undistorted images	List of undistorted images.

Table 2.6: Meshroom Prepare Dense Scene Node

2.2.7 Depth Map

This step in the pipeline retrieves the depth value of each pixel for all cameras that the StructureFromMotion Node processed. It uses the Semi-Global Matching method as proposed by Hirschmuller (2005).

For each image, it selects the **"N"** best & closest cameras around. It chooses front-parallel planes based on the intersection of the optical axis with the pixels of the selected neighbouring cameras (Hirschmuller, 2005). This produces a volume **W, H, Z** with several depth candidates per pixel, and it estimates the similarity for all of these. The similarity is calculated by the *Zero Mean Normalized Cross-Correlation (ZNCC)* of a small patch in the principal image, which is reprojected into at the other camera, which creates a volume of similarities as suggested by Strecha et al. (2006). For each neighbouring image, it accumulates similarities in this volume. Nevertheless, this volume is noisy; therefore, there is a filter in each step along the X & Y axis, which reduces the isolated high values. Finally, it selects the local minima and replaces the selected plane index with a depth value stored into a depth map (Scharstein et al., 2001). This depth map has banding artifacts as it based on the original selection of depth values as mentioned by Scharstein et al. (2001). Thus, a refinement process calculates the depth values with sub-pixel accuracy.

The components in the node are described in table 2.7

Name	Description
MVS Configuration File:	SfMData file.
Images Folder	Use images from a specific folder instead of those specify in the SfMData file. Filename should be the image uid.
Downscale	Image downscale factor (1, 2, 4, 8, 16)
Min View Angle	Minimum angle between two views.(0.0 - 10.0)
Max View Angle	Maximum angle between two views. (10.0 - 120.0)
SGM: Nb Neighbour Cameras	Semi Global Matching: Number of neighbour cameras (1 - 100)
SGM: WSH: Semi Global Matching	Half-size of the patch used to compute the similarity (1 - 20)
SGM: GammaC	Semi Global Matching: GammaC Threshold (0 - 30)
SGM: GammaP	Semi Global Matching: GammaP Threshold (0 - 30)
Refine: Number of samples	(1 - 500)
Refine: Number of Depths	(1 - 100)
Refine: Number of Iterations	(1 - 500)
Refine: Nb Neighbour Cameras	Refine: Number of neighbour cameras. (1 - 20)
Refine: WSH	Refine: Half-size of the patch used to compute the similarity. (1 - 20)
Refine: Sigma	Refine: Sigma Threshold (0 - 30)
Refine: GammaC	Refine: GammaC Threshold. (0 - 30)
Refine: GammaP	Refine: GammaP threshold. (0 - 30)
Refine: Tc or Rc pixel size	Use minimum pixel size of neighbour cameras (Tc) or current camera pixel size (Rc)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output	Output folder for generated depth maps

Table 2.7: Meshroom Depth MapNode

2.2.8 Depth Map Filter

This node is in charge of processing the DepthMap Node results as the original depth maps will not be entirely consistent. Specific depth maps will be interpreted to see areas that are occluded by other dept maps. Thus, this step of the pipeline isolates these areas and ensures depth consistency.

The components in the node are described in table 2.8

Name	Description
Input	SfMData file
Depth Map Folder	Input depth map folder
Number of Nearest Cameras	Number of nearest cameras used for filtering 10 (0 - 20)
Min Consistent Cameras	Min Number of Consistent Cameras 3 (0 - 10)
Min Consistent Cameras Bad Similarity	Min Number of Consistent Cameras for pixels with weak similarity value 4 (0 - 10)
Filtering Size in Pixels	Filtering size in Pixels (0 - 10)
Filtering Size in Pixels Bad Similarity	Filtering size in pixels (0 - 10)
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace)
Output	Output folder for generated depth maps

Table 2.8: Meshroom Depth Map Filter Node

2.2.9 Meshing

The primary purpose of this node is to construct a dense geometric surface representation of the scene. Initially, it fuses all the processed depth maps into a global octree, and where it is compatible, it will merge depth values into the octree cells. Afterwards, the node performs a 3D Delaunay tetrahedralization process. Subsequently, a complex voting procedure computes the weighs on cells and weights the faces connecting the cells as mentioned by Jancosek and Pajdla (2014).

A Graph Cut Max-Flow (Boykov and Kolmogorov, 2004) is applied to efficiently reduce the volume, which represents the extracted mesh surface. Furthermore, it filters bad cells on the surface and applies a Laplacian filtering process on the mesh to remove local artifacts.

The components in the node are described in table 2.9

Name	Description
Input	SfMData file.
Depth Maps Folder	Input depth maps folder
Filtered Depth Maps Folder	Input filtered depth maps folder
Estimate Space From SfM	Estimate the 3d space from the SfM
Min Observations For SfM Space Estimation	Minimum number of observations for SfM space estimation. (0-100) 3
Min Observations Angle For SfM Space Estimation	Minimum angle between two observations for SfM space estimation. (0-120) 10
Max Input Points	Max input points loaded from depth map images (500**000** - 500000000)
Max Points	Max points at the end of the depth maps fusion (100**000** - 10000000)
Max Points Per Voxel	(500**000** – 30000000)
Min Step	The step used to load depth values from depth maps is computed from maxInputPts. Here we define the minimal value for this step, so on small datasets we will not spend too much time at the beginning loading all depth values (1- 20) 2
Partitioning	(singleBlock, auto)
Repartition	(multiResolution, regularGrid)
angleFactor	(0.0-200.0) 15.0
simFactor	(0.0-200.0) 1.0
pixSizeMarginInitCoef	(0.0-10.0) 2.0
pixSizeMarginFinalCoef	(0.0-10.0) 4.0
voteMarginFactor	(0.1-10.0) 4.0
contributeMarginFactor	(0.0-10.0) 2.0
simGaussianSizeInit	(0.0-50) 10.0
simGaussianSize	(0.0-50) 0.1
minAngleThreshold	(0.0-10.0) 0.01
Refine Fuse	Refine depth map fusion with the size of the new pixel defined by angle and similarity scores.
Add Landmarks To The Dense Point Cloud	Add SfM Landmarks to the dense point cloud.
Colourize Output	Whether to colourize output dense point cloud and mesh.
Save Raw Dense Point Cloud	Save dense point cloud before cut and filtering.
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace).
Output Mesh	Output mesh (OBJ file format). mesh.obj
Output Dense Point Cloud	Output dense point cloud with visibilities (SfMData file format). densePointCloud.abc

Table 2.9: Meshroom Meshing Node

2.2.10 Mesh Filtering

This node filters and removes the unwanted elements from the resulting mesh.

The components in the node are described in table 2.10

Name	Description
Input	Input Mesh (OBJ file format)
Filter Large Triangles Factor	Remove all large triangles. We consider a triangle as large if one edge is bigger than N times the average edge length. Put zero to disable it. 60 (1 - 100)
Keep Only the Largest Mesh	Keep only the largest connected triangles group (True/False)
Nb Iterations	5 (0 - 50)
Lambda	1 (0-10
Verbose Level	
Verbose Level	[‘fatal’, ‘error’, ‘warning’, ‘info’, ‘debug’, ‘trace’]
Output mesh	Output mesh (OBJ file format) internalFolder + ‘mesh.obj

Table 2.10: Meshroom Mesh FilteringNode

2.2.11 Texturing

The principle of this component is to create a texture of the generated mesh. If the mesh has no correspondent UV, it computes automatic UV maps. It uses a basic UV mapping approach to reduce the texture space as proposed by Levy et al. (2002).

For each triangle, it uses the visibility information associated with each vertex to retrieve texture candidates. It filters the cameras that do not have a good angle to the surface to favourable front-parallel cameras to average the pixel values. Furthermore, it uses a generalization of multiband blending as described by Burt and Adelson (1983). Thus, it averages more views in the low frequencies in comparison to the high frequencies.

The components in the node are described in table 2.11

MVS Configuration file	**. . . /mvs.ini**
Input Dense Reconstruction	Path to the dense reconstruction result (mesh with per vertex visibility)
Other Input Mesh	Optional input mesh to texture. By default, it will texture the result of the reconstruction.
Texture Side	Output texture size 1024, 2048, 4096, 8192, 16384
Texture Downscale	Texture downscale factor1, 2, 4, 8
Texture File Type	Texture File Type 'jpg', 'png', 'tiff', 'exr'
Unwrap Method	Method to unwrap input mesh if it does not have UV coordinates Basic (>600k faces) fast and simple. Can generate multiple atlases LSCM (<= 600k faces): optimize space. Generates one atlas ABF (<= 300k faces): optimize space and stretch. Generates one atlas
Fill Holes	Fill Texture holes with plausible values True/False
Padding	Texture edge padding size in pixel (0-100)
Max Nb of Images For Fusion	Max number of images to combine to create the final texture (0-10)
Best Score Threshold	0.0 to disable filtering based on threshold to relative best score (0.0-1.0)
Angle Hard Threshold	0.0 to disable angle hard threshold filtering (0.0, 180.0)
Force Visible By All Vertices	Triangle visibility is based on the union of vertices visiblity.True/False
Flip Normals	Option to flip face normals. It can be needed as it depends on the vertices order in triangles and the convention change from one software to another.
Visibility Remapping Method	Method to remap visibilities from the reconstruction to the input mesh (Pull, Push, PullPush).
Verbose Level	verbosity level (fatal, error, warning, info, debug, trace).
Output Folder	Folder for output mesh: OBJ, material and texture files.
Output Mesh	Folder for output mesh: OBJ, material and texture files. internalFolder + 'texturedMesh.obj
Output Material	Folder for output mesh: OBJ, material and texture files. internalFolder + 'texturedMesh.mtl
Output Textures	Folder for output mesh: OBJ, material and texture files. internalFolder + 'texture_*.png

Table 2.11: Meshroom Texturing Node

2.3 PCL

The **Point Cloud Library (PCL)** is a large scale open-source project used for point cloud processing. The PCL framework contains multiple state-of-the-art algorithms ranging from feature estimation, registration, filtering, model fitting to segmentation (Rusu and Cousins, 2011). PCL is a cross-platform library, and it is compatible with Windows, Linux and macOS.

The PCL framework is divided into a series of modular libraries such as:

- Filters
- Features
- Keypoints
- Kegistration
- kdtree
- octree
- Segmentation
- Sample Consensus
- Surface
- Recognition
- IO
- Visualization

These series of modular libraries have a series of embedded algorithms that can be used in scenarios such as filtering outliers from noisy data, stitch multiple 3D point clouds,

segment relevant parts of the scene & point cloud, extract key points and compute image descriptors to recognize real-world objects based on geometric appearance, among many other possible implementations.

The selected components and algorithms that were used in this project include:

- Statistical Outlier Removal Filter
- Pass Through Filter
- Voxel Grid Filtering
- Plane Model Segmentation
- KdTree
- Euclidean Cluster Extraction
- Principal Component Analysis
- Moving Least Squares
- Poisson Surface Reconstruction

2.3.1 Statistical Outlier Removal Filter

This component of the PCL library aims to remove noisy measurements (outliers) from a point cloud dataset using statistical analysis techniques (Rusu and Cousins, 2011).

Typically, Laser Scans produces point cloud datasets with various point densities. Similarly, measurement errors generate sparse outliers that can corrupt the results of the dataset. This complicates the process of estimating local point cloud characteristics in particular surface normals and curvature changes. Hence, leading to erroneous values that can create failures in the point cloud registration process. Some of these irregularities in the dataset can be addressed with a statistical analysis performed on each point's neighbourhood and removing the points that do not meet specified criteria as mentioned by Rusu and Cousins (2011). The sparse outlier removal is built on the computation of the distribution of point to neighbours distances of the source dataset (Rusu and Cousins, 2011). For each point, it computes the mean distance from the points to all the corresponding neighbours. This process assumes that the resulting distribution is Gaussian with a standard deviation and a mean. Then all points whose mean distances are outside an interval defined by global distances (mean & standard deviation) can be expressed as an outlier. Hence, it can be removed from the dataset as suggested by Rusu and Cousins (2011).

Figure 2.3 illustrates the results of the sparse outlier analysis and removal process. The initial dataset is on the left, and the processed result in on the right. Furthermore, the graph on the right illustrates the mean k-nearest neighbour distances in a point neighbourhood, before and after the filtering process (Rusu and Cousins, 2011).

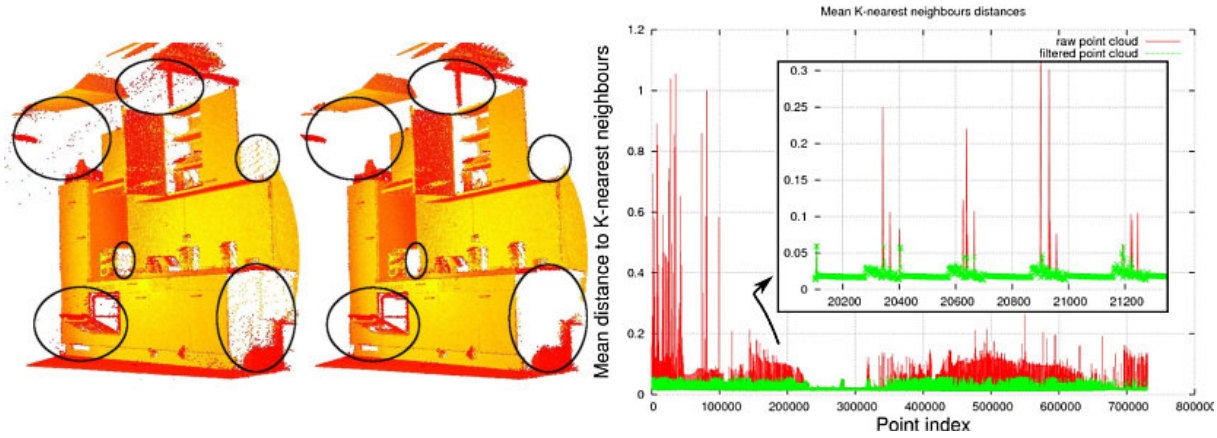


Figure 2.3: Removing outliers using a StatisticalOutlierRemoval filter (Rusu and Cousins, 2011)

2.3.2 Pass Through Filter

This component of the PCL library has embedded two main components of the framework. The **PassThrough** components use the base **Filter** class methods to pass through all the data that satisfies certain constraints.

PassThrough passes points in a point cloud based on constrains for one specific field of the point type (Rusu and Cousins, 2011). It iterates through the entire input pointcloud once, and automatically filters non-finite points, including the points outside the interval specfied by *setFilterLimits()*, that only applies uniquely to the configured field *setFilterFieldName()*. The component *setFilterLimits()* sets the numerical limits for the field for data filtering, whereas *setFilterFieldName()* configures the name of the field that will be used for data filtering.

Therefore, this component performs a filter along a given dimension, removing the values outside or inside the configured range. The image 2.4 illustrates the process. The point cloud has five points after filtering. The green points represent the remaining result, and the red points are the points that were removed by the filter.

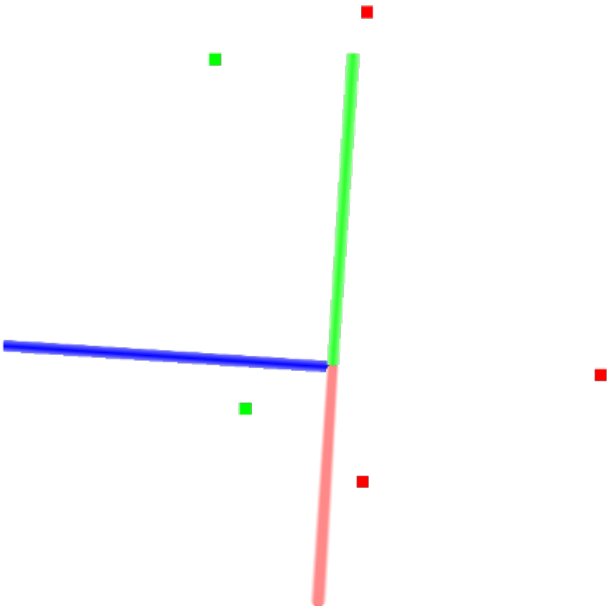


Figure 2.4: Filtering a PointCloud using a PassThrough filter
(Rusu and Cousins, 2011)

2.3.3 Voxel Grid Filtering

A voxel Grid illustrates a value on a regular grid in 3D space. A Voxel is an image of a 3D space region which is limited by defined sizes, which has its corresponding nodal point coordinates in an accepted system, own form, own state parameter that demonstrates it belongs to some modelled object and its associated properties of the modelled region (Shchurova, 2015).

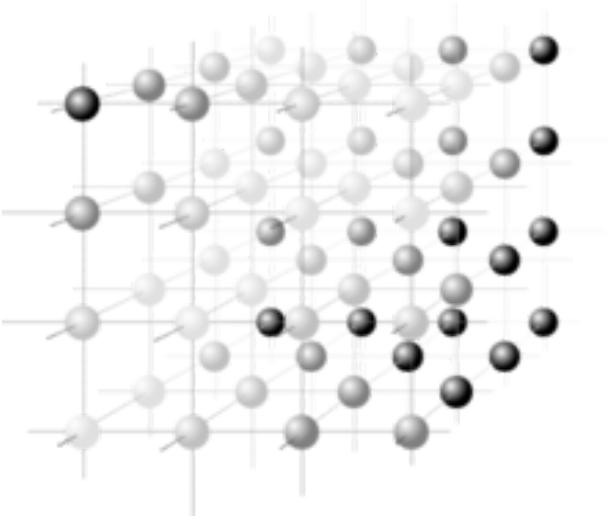


Figure 2.5: Illustration of a voxel grid containing color values
(Shchurova, 2015)

This component of the PCL framework aims to downsample (*Reduce the Number of Points*) of a point cloud dataset using a voxelized grid method. The **VoxelGrid** component creates a 3D Voxel grid, which in ordinary terms refers to a set of small boxes in 3D space over the input point cloud dataset. Once it successfully converted into a voxel grid, then in each voxel(*3D box*), all the existing points will be approximated (*Downsampled*) with their centroid as suggested by Rusu and Cousins (2011). This approach represents the underlying surface of the point cloud dataset with high accuracy. The main component is to set a defined and voxel leaf Size, which allows setting the Voxel Size and the Number of Voxel in the Voxel Grid. Therefore, directly influencing the downsampling process and the resulting processed point cloud.

2.3.4 Plane Model Segmentation

This component of the PCL library aims to perform a plane segmentation of a given set of points, which finds all the points within a point cloud data set that support a plane model. One of the foundations that this component uses is *pcl::ExtractIndices*, which aims to extract the indices from a point cloud. It is a filter that extracts a subset of points from a point cloud dataset related to the indices output of a segmentation algorithm.

Similarly, it uses *pcl::SACSegmentation*, which represents the Nodelet segmentation class for Sample Consensus methods and model, in a way that creates a Nodelet wrapper that can be used for generic-purpose SAC-based Segmentation (Rusu and Cousins, 2011). The *pcl::SACSegmentation*, creates an object and sets the model & method type. Furthermore, it specifies the "distance threshold" used to determine the distance for a point in the model to be considered an inlier. The used method in this project uses the **RANSAC** method. The main advantage of RANSAC is its simplicity and robustness.

The estimated plane parameters are estimated with equation 2.2.1, where **a,b,c,d** are the model coefficient values.

$$ax + by + cz + d = 0 \tag{2.1}$$

Figure 2.6, illustrates the process below. The red points are the outliers, whereas the green points are the inliers of the found plane model.

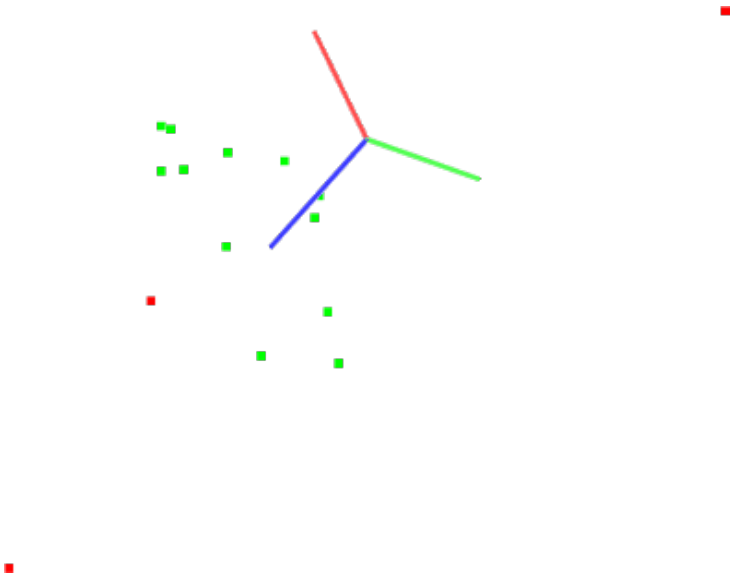
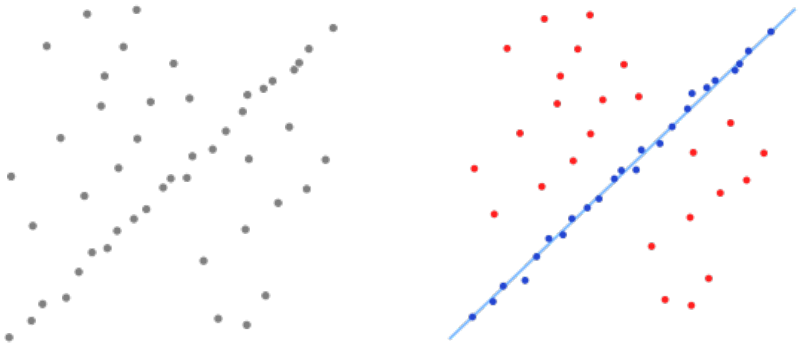


Figure 2.6: Illustration of Plane model segmentation
(Shchurova, 2015)

2.3.4.1 RANSAC

Random sample consensus (RANSAC) is an iterative method that is used to estimate parameters of a mathematical model from a set of observed data that contains outliers (Strutz, 2016). Hence, it can be interpreted as an outlier detection method. It is a non-deterministic algorithm that generates an adequate result with a specific probability, and the probability increases as more iteration are performed.



(a) A data set with many outliers for which a line has to be fitted

(b) Fitted line with RANSAC; outliers have no influence on the result

Figure 2.7: Dataset Before & After RANSAC algorithm
(Strutz, 2016)

2.3.5 KdTree

In this project Kdtrees are used to find the K nearest neighbour of certain points or locations and how to find these within an specified radius.

A K-D tree (*k-dimensional tree*) is a data structure that is commonly used to organise a certain number of points in a space with k dimensions as suggested by Rusu and Cousins (2011). It is a binary search tree with other constraints imposed on it and are helpful for searches for range and closest neighbours. As this project uses point clouds, the Kdtrees would have three dimensions. Each level of a Kdtree splits all children along a specific dimension, utilising a hyperplane that is perpendicular to the corresponding axis. At the root of the KdTree, all children will be divided based on the first dimension, and each level down in the Kdtree divides on the next dimension; finally, it returns to the first dimension once all others have been exhausted (Rusu and Cousins, 2011).

KdTrees are used extensively in many components of the project framework pipeline.

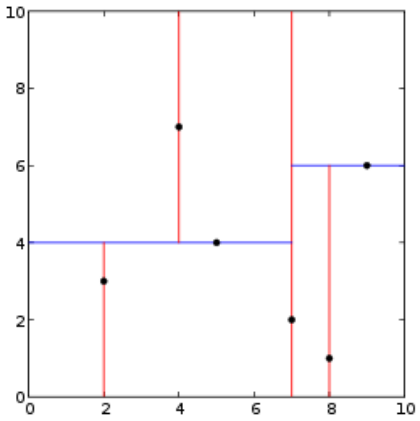


Figure 2.8: Example of a 2-dimensional k-d tree
(Rusu and Cousins, 2011)

2.3.6 Euclidean Cluster Extraction

The clustering method allows dividing a non-organized point cloud dataset P into smaller components to reduce the overall processing time for P. Point Cloud Data Clustering can be approached in a Euclidean manner and be implemented using 3D grid subdivisions of the space with fixed-width boxes or an octree data structure. In this project, Kdtrees are used to find the nearest neighbours and implements a clustering technique that is similar to a flood fill algorithm (Rusu and Cousins, 2011).

The following Steps illustrates the clustering algorithm in this project with a Kdtree:

1. create a Kd-tree representation for the input point cloud dataset P;
2. set up an empty list of *clusters* C, and a queue of the points that need to be checked Q;
3. then for every point $p_i \in P$, perform the following steps:
   - add p_i to the current queue Q;
   - for every point $p_i \in Q$ perform:
     - Search for the set P_i^k of point neighbors of p_i in a sphere with radius $r < d_{th}$
     - for every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q
   - when the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list
4. the algorithm terminates when all points $p_i^k \in P$ have been processed and are now part of the list of point clusters C

Figure 2.9, illustrates a cluustering output of a dataset, where different components were divided into multiple clusters

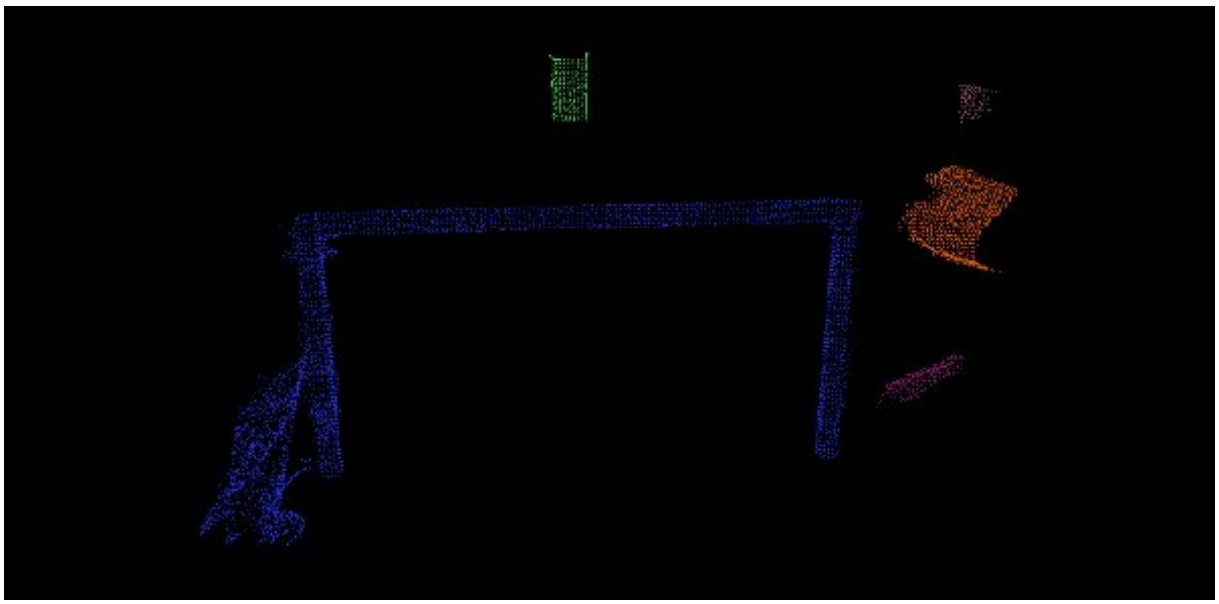


Figure 2.9: Example of point clouud cluster
(Rusu and Cousins, 2011)

2.3.7 Principal Component Analysis

The Principal Component Analysis (PCA) is the process of computing the principal components and use them to change the basis of data, using only the first few principal components and discarding the rest. The principal components are a sequence of p unit vectors of a collection of points in a real coordinate space (Artac et al., 2002).

The principal components are extracted using a singular value decomposition method applied on the covariant matrix of the central input point cloud dataset. Once the PCA analysis is performed using the PCL library, it is possible to calculate the following components:

- Mean of input data
- Eigen Vectors: Ordered set of vectors represents the final principal components and the Eigen cartesian space.
- Eigen Values: These are the correspondent loading of the Eigen Vectors in a descending order.

The Principal Component Analysis is used to determine the scale of the two point clouds used in this project and properly adjust the final mesh.

2.3.8 Moving Least Squares

Moving Least Squares (MLS) is a method of reconstructing continuous functions from a set of unorganized point samples through the computation of weighted least squares, biased towards the region around the point at which the reconstructed value is requested (Levin, 1998).

MLS can be modelled considering function f: $\mathbb{R}^n \rightarrow \mathbb{R}$ and a set of sample points $S = \{(x_i; f_i)|f(x_i) = f_i\}$. Then, it MLS approximation of degree m at the point x is $\tilde{p}(x)$ where $\tilde{p}$ reduces the weighted least square error $\sum_{i \in I}(p(x_i) - f_i)^2\theta(\|x - x_i\|)$ over all polynomials p of degree m in $\mathbb{R}^n$ as mentioned by Levin (1998).

The main objective of Moving Least Squares (MLS) is to smooth and resample noisy data in a surface reconstruction object. Some point clouds have certain data irregularities, which can be caused by minor errors in the distances measurements, and are complex to remove using conventional statistical analysis. It is essential to account for complex surfaces and occlusions to create a complete model. Resampling algorithms such as MLS can be used to recreate the missing parts of the surface by a high order polynomial interpolation between the surroundings data points. Resampling allows for correcting minor errors and smoothing the surface of a point cloud dataset.

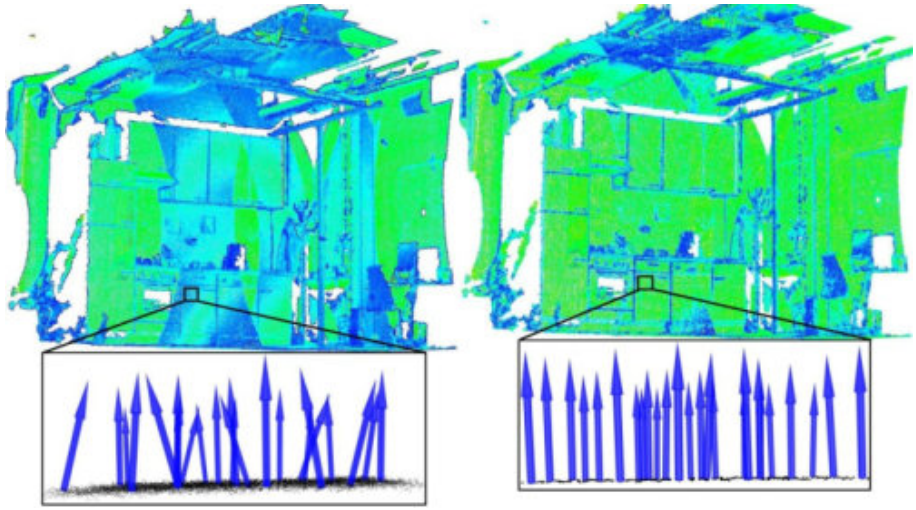


Figure 2.10: Moving Least Square Smoothing
(Rusu and Cousins, 2011)

In figure 2.10, on the left side, is illustrated the effect of estimating the normals of a point cloud; however, due to alignment errors, the normals are noisy. On the right side of figure 2.10, the effects of the MLS algorithm is demonstrated as it smoothes the surface of the point cloud.

To approximate the surface illustrated by the local neighbours of points $p1, p2, \rightarrow pk$ at a point q, it uses a bivariate polynomial function, which is defined on a robust modelled reference plane. Figure 2.11, shows the curvatures at each point with the eigenvalue relation before and after the resampling method.

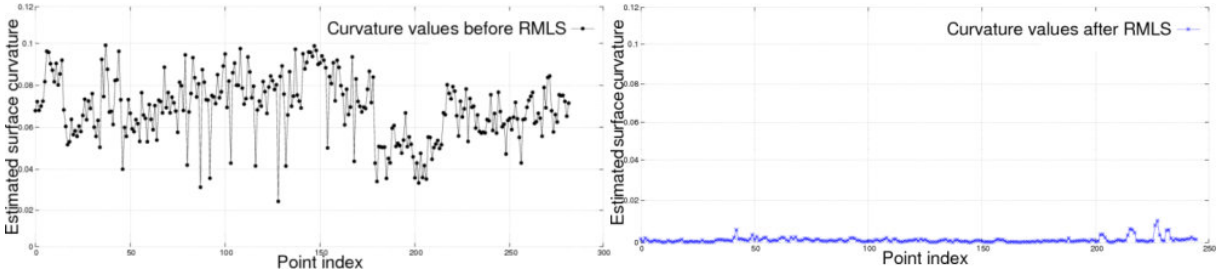


Figure 2.11: Curvatures of MLS before & after
(Rusu and Cousins, 2011)

Furthermore, the Moving Least Squares Component of the PCL library also allows for different upsample methods. These include:

DISTINCT CLOUD Project the points of the distinct cloud to the MLS surface(Rusu and Cousins, 2011).

SAMPLE LOCAL PLANE The local plane of each input point will be sampled in a circular fashion using the $UpsamplingRadius$ and the $UpsamplingStep$ parameters(Rusu and Cousins, 2011).

RANDOM UNIFORM DENSITY The local plane of each input point will be sampled using a uniform random distribution such that the density of points is constant throughout the cloud - given by the $DesiredNumPointsinRadiusParameter$(Rusu and Cousins, 2011).

VOXEL GRID DILATION The input cloud will be inserted into a voxel grid with voxels of size $VoxelSize$.This voxel grid will be dilated ($DilationIterationNum$) N times, and the resulting points will be projected to the MLS surface of the closest point in the input cloud; the result is a point cloud with filled holes and a constant point density (Rusu and Cousins, 2011).

For this project, the **RANDOM UNIFORM DENSITY** method is used for upsampling the MLS processed point cloud. This method takes the parameters for a desired point cloud density within a fixed radius neighbourhood as suggested by Ichim (2012). For each point, based on the neighbours' density, it will add more points on the local plane using a random number generator. The random number generator will have a uniform distribution, and it will stop once the desired density is achieved. Then it will replay the MLS algorithm to smooth the surface again.

Figure 2.12 illustrates on the left the implementation of MLS with **UNIFORM DENSITY** upsampling method on a point cloud, whereas on the right if the raw point cloud.

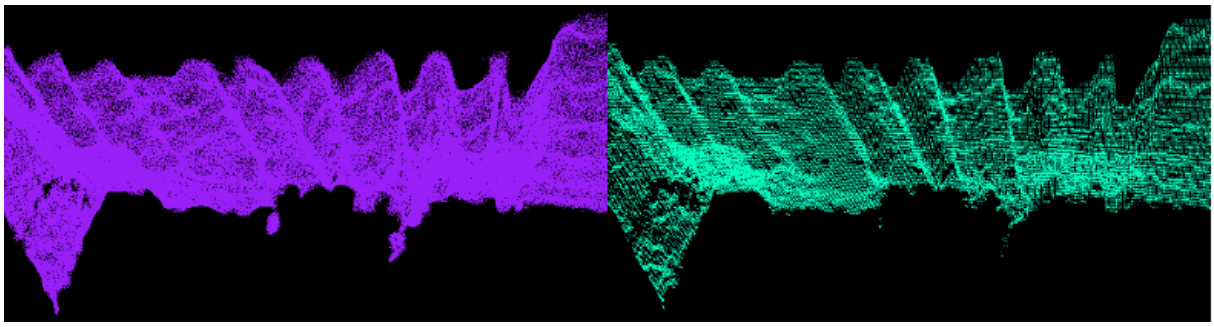


Figure 2.12: Uniform Density Upsample Method After & original (Ichim, 2012)

2.3.9 Poisson Surface Reconstruction

In this project the Poisson Reconstruction algorithm is implemented , to recontruct the processed point cloud. The Poisson Reconstruction algorithm demonstrates that the surface reconstruction from a set of oriented points can be associates with a spatial Poisson problem. This formulation utilizes all the points at once, with no need of using heuristic spatial partitioning or blending, hence, making it resilient to noise as mentioned by Kazhdan et al. (2006).

The main objective of this algorithms to reconstruct a smooth surface, which is based on a large number of points p_i from a point cloud, where each point possesses an estimate of the local surface normal n_i. It aims to create an implicit function f, whose value is zero at the points p_i and the gradient at points p_i is equal to the normal vector n_i (Kazhdan et al., 2006). The set of (p_i, n_i) is modelled as a continuous vector field V and the implicit function is found by integrating the vector field V. In complex calculations, it is possible to perform a *least-squares fit* to minimize the difference between the gradient of f and V (Kazhdan et al., 2006).

Figure 2.13: Poisson Recontruction Example.
On the left it it input pointcloud dataset, whereas on the right is output of the Poisson Surface Reconstruction Algorithm(Kazhdan et al., 2006)

2.4 Open3D

Open3D is an open-source library that supports fast development of software that handles 3D data. The Open3D frontend exposes outputs a set of carefully chosen data structures and algorithms in both Python and C++ (Zhou et al., 2018). The backend of the framework is highly optimized, and it configured for parallelization.

Open3D is compatible with Linux, macOS and Windows, and it can be installed via source code or packages.

The core features include:

- Simple installation via conda & pip
- 3D data structures
- 3D data processing algorithms
- Scene reconstruction
- Surface alignment
- PBR rendering
- 3D visualization
- Python binding

In this project, Open3D was used as an auxiliary library used as a contingency for Poisson Reconstruction. As in many circumstances, the normals of a point cloud might not be oriented appropriately *orient normals consistent tangent plane* allows to propagate the normal orientation with a minimum spanning-tree as suggested by Zhou et al. (2018). After the normals are aligned, it will excute the Poisson Surface Reconstruction algorithm (Kazhdan et al., 2006) and solves a regularized optimization problem to obtain a smooth surface mesh.

Figure 2.14: Open3D Poisson Recontruction Example
(Zhou et al., 2018)

Methodology

As mentioned before, this project is associated with the software reconstruction of a 3D scanner device. The key components are divided into three processes that include:

- Data collection
- Photogrammetry Process
- Data Processing Framework

All the components mentioned above will work in a Pipeline manner and as the critical process for a 3D scanner includes Data collection. Once the Data is collected, a photogrammetric process will run to reconstruct the object based on the input data. Finally, the resultant reconstructed object will be processed using a Custom framework pipeline based on the PCL (*Point Cloud Library*) and proceed with a processed mesh. Once the mesh has finished processing, it can be used in different applications such as virtual clothing fitting.

3.1 Data Collection

The Data Collection process is performed with a 3D scanning RIG (device). The process will be performed with a series of twenty-eight cameras and a single Lidar. The cameras will be located and assembled in a series of eight tall poles and four short poles. Similarly, the Lidar will be mounted to a single Pole.

All the cameras will be controlled using Raspberry Pis. There will be a total of three Raspberry Pis that control the execution process to acquire data from all the cameras that are attached to them, with a total number of twenty-eight images. Similarly, an Nvidia Jetson Xavier is used to acquire data from the Lidar. The Nvidia Jetson Xavier will obtain a single image and a point cloud of the scanned subject.

Cameras

The cameras used for data acquisition have the following specifications.

Model No.	HBV-1825
Model Size	62mm × 9mm × 5.68mm±0.2mm
Active Array Size	2592 x 1944
Pixel Size	1.4μm x1.4μm
shutter	rolling shutter / frame exposure
Field of View	65°

Table 3.1 Cameras Specification

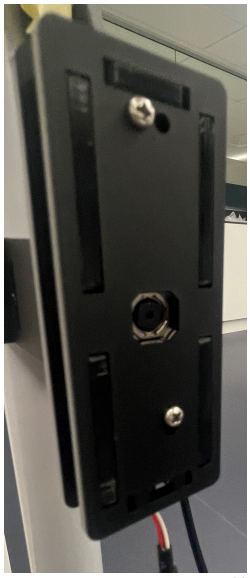


Figure 3.1: Tall Pole Cameras Location With Housing.

As observed in figure 3.1, the cameras will have a Plastic Housing. This Housing device allows to place the cameras safely in the poles as well as protecting the electronics and connection from external factors. The cameras will be connected to a series of USB hubs directly connected to the Raspberry Pis to control the data acquisition process.

Lidar

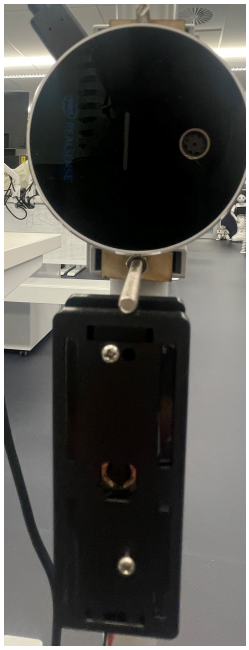


Figure 3.2: Lidar Mounted in Pole

The Lidar used for the data acquisition process in an Intel RealSense L515. The Intel RealSense will be located in a tall pole, and it will be attached in the middle section to maximise the field of view. It will be locked above the central camera of a tall pole. As the Lidar has both an RGB sensor and a Depth Camera, it will be used to capture an RGB image and a Pointcloud with the Depth and RGB Sensor. The Lidar will be connected and controlled with an Nvidia Jetson Xavier. It will use the RealSense SDK to capture both RGB image and Point.

Table 3.2 states the specification of the Intel RealSense L515.

Model No.	Intel RealSense L515
Depth Technology	Lidar
Ideal Range	0.25 m to 9 m
Depth Field of View (FOV):	70° × 55° (±3°)
Depth output resolution:	Up to 1024 × 768
Depth Accuracy:	∼5 mm to ∼14 mm thru 9 m
RGB frame resolution	1920 × 1080
RGB sensor technology	Rolling Shutter
RGB sensor FOV (H × V)	70° × 43° (±3°)
RGB sensor resolution	2 MP

Table 3.2: Intel RealSense L515 Specification

3.1.1 Layout & Configuration

As mentioned before, Lidar will be controlled to an Nvidia Jetson Xavier, which will make it agnostic of the Cameras acquisition process. It was previously mentioned that the Cameras would be controlled with a Series of multiple Raspberry Pis, and those will be managed via an SSH connection with Putty. Due to the limited number of USB connectors, a series of USB hubs will control more cameras with Single Raspberry Pis. Figure 3.3, illustrates how the twenty-eight cameras will be connected with the Different Raspberry Pis.

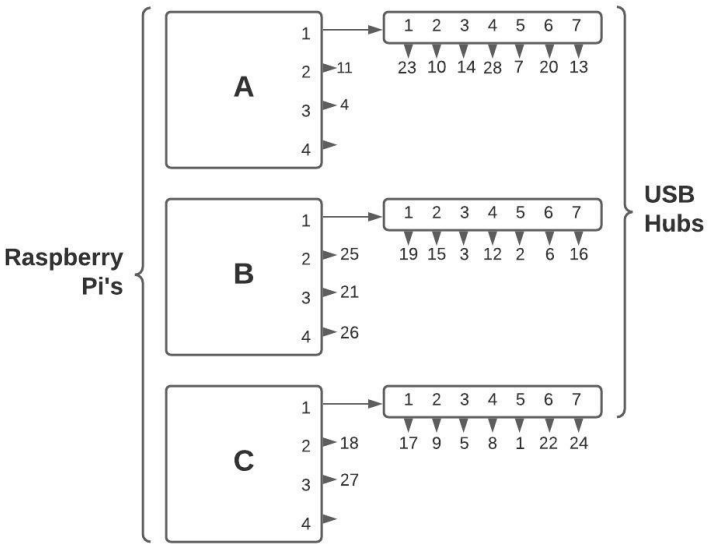


Figure 3.3: Cameras Lay-out connection

All the cameras will be mounted in the RIG in a series of poles. There will be a total number of eight tall and four short poles. The Tall poles will be used to attached three cameras to the Frame. The First Camera will be located on top of the pole and oriented toward the subject to have the best overlap with the Field of View. The Second Camera is located in the middle of the pole and parallel oriented to the scanned subject to enhance the field of view. The third Camera is located at the bottom of the pole, and it will be oriented upwards towards the scanned subject in order to have comprehensive coverage. This configuration can be observed on figure 3.4.

On the other side, there will be four short poles that each will hold a single camera. These poles aim to gain close data as well as acquiring images from regions of interest around the body in order to increases the accuracy of the final scan.

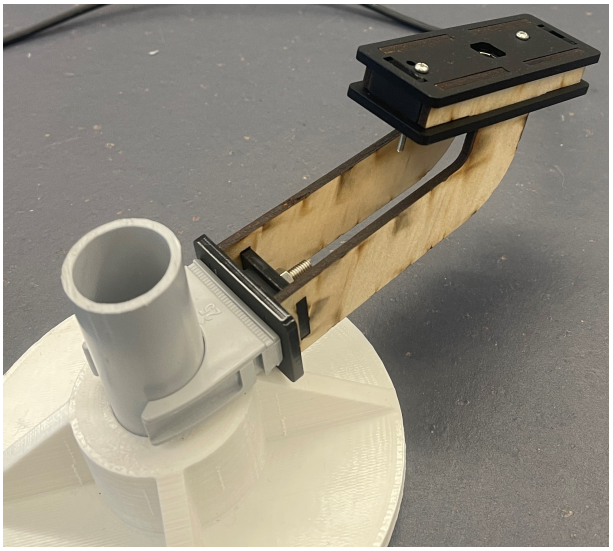


Figure 3.5 Open3D Poisson Recontruction Example

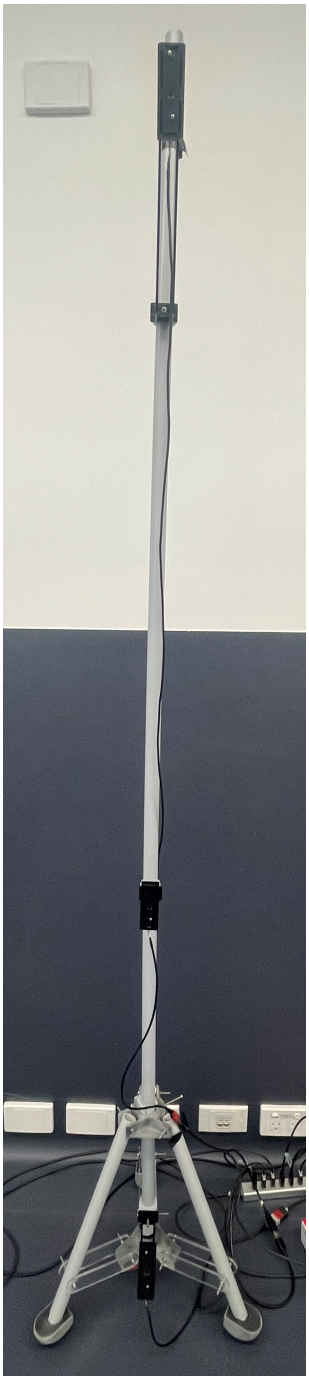


Figure 3.4: Tall Pole Cameras Configuration.

All the tall poles will be placed in a circular shape to capture as much data as possible. The subject (person) that will use the RIG will be located in the circle's origin, and all the tall poles will be placed 1.5 m away from the person to have the best overlap between all cameras. The four short poles will be located closeup from the person that will be scanned. The four poles will be placed 60 cm away from the person (circle origin) to capture close up data for complex regions and improve the overall reconstruction. The architecture of this RIG is observed on figures 3.6 and 3.7.

(a) RIG From View **(b)** RIG lateral View

Figure 3.6: RIG scanning With Poles Locations

Figure 3.7: CAD of RIG and Poles Placement

3.1.2 Cameras Calibration

As mentioned before, there will be a set of twenty-eight cameras in the RIG. It is crucial to be able to calibrate these cameras an object the intrinsic parameters in order to input these parameters into the Photogrammetry Pipeline to enhance the reconstruction process. The intrinsic parameters are illustrated in the camera matrix on equation 2.2.1.

All the cameras were calibrated by taking a series of fifty photos of a checkerboard. The checker board is in different positions and orientation in the images, as this enhances the calibration process. Once all the images have been captured, these were processed using

OpenCV and Matlab to perform the calibration process and obtain several parameters such as the camera intrinsics, distortion, etc.

Table 3.3 contains the result of the camera Calibration process for each camera.

Photo ID	**Camera ID**	**Focal Lenght**	**Principal Point X**	**Principal Point y**	**Radial 1**	**Radial 2**	**FoV**	**ID**
Lidar	Lidar	1348.9	987.1289	552.4585	0.002083	0.004345	70	1
A1	23	2069.79669	1062.276	825.4553	0.043834	-0.16655	65	2
A2	10	2023.979147	1055.148	740.4276	0.047799	-0.18798	65	3
A3	14	2073.0703	1064.465	772.3883	0.037661	-0.16921	65	4
A4	28	1685.622704	1044.281	772.533	0.056978	-0.06307	65	5
A5	7	2077.408129	1054.39	849.7223	0.037899	-0.19003	65	6
A6	20	2080.992114	1030.656	743.8521	0.043046	-0.20511	65	7
A7	13	2035.145838	1070.591	764.946	0.025991	-0.16304	65	8
A8	11	2057.82487	1025.147	799.6319	0.032882	-0.14583	65	9
A9	4	2045.525031	1017.459	781.6747	0.052045	-0.22293	65	10
B1	19	2007.968395	1053.849	793.203	0.040332	-0.1859	65	11
B2	15	2031.574945	1060.078	775.8948	0.033055	-0.17165	65	12
B3	3	2111.254725	1037.889	799.7706	0.032561	-0.17198	65	13
B4	12	2058.9386	1022.306	742.7845	0.046652	-0.20622	65	14
B5	2	2078.153107	980.6483	781.8416	0.060616	-0.21788	65	15
B6	6	2099.600814	1000.535	731.1288	0.035295	-0.20673	65	16
B7	16	2066.488622	1035.458	756.2719	0.045434	-0.20259	65	17
B8	25	2076.530213	1060.421	815.187	0.06066	-0.20289	65	18
B9	21	2031.68852	1038.411	743.4008	0.032986	-0.19021	65	19
B10	26	2048.856969	1047.928	747.5051	0.037514	-0.18331	65	20
C1	17	2072.818087	1037.388	805.2937	0.041437	-0.17553	65	21
C2	9	2023.270233	1066.445	761.1484	0.039684	-0.19295	65	22
C3	5	2038.942135	1052.651	734.7338	0.0562	-0.2034	65	23
C4	8	2024.024741	1064.304	760.5158	0.037415	-0.18791	65	24
C5	1	2063.919749	976.3176	761.5408	0.022828	-0.11913	65	25
C6	22	2061.102003	1071.594	788.9326	0.053629	-0.20935	65	26
C7	24	2060.07463	994.6496	772.853	0.053323	-0.1899	65	27
C8	18	2124.144197	1060.398	822.0559	0.05472	-0.25224	65	28
C9	27	2062.300912	995.6599	763.2463	0.045703	-0.21291	65	29

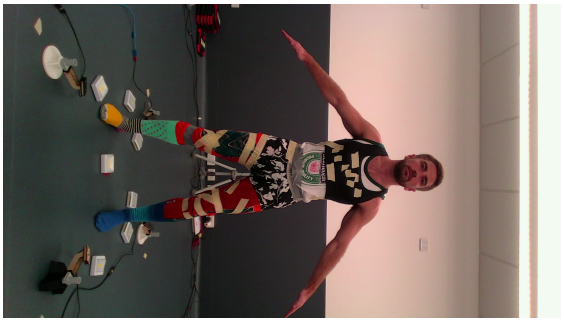
Table 3.3: Cameras Calibration Parameters

Execution

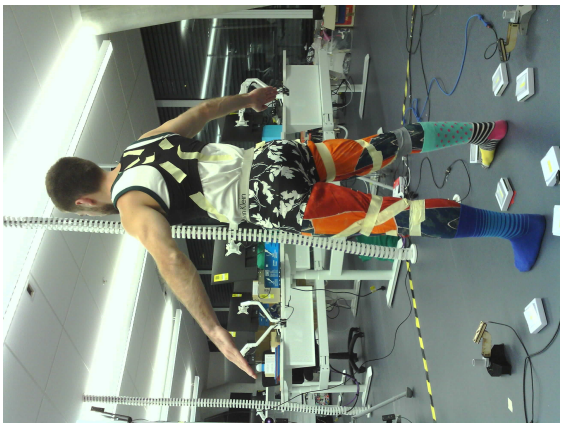
The execution process starts by connecting the user computer to the Raspberry PIs with an SSH connection via Putty. Once a connection is established, a trial run will be performed to ensure the cameras work as intended. After the trial run is successful, the RIG is fully operational.

Before a person enters the scan, the user is required to start the Nvidia Jetson Xavier and execute the Realsense SDK. Once the mentioned above are performed, a person can enter the RIG to get scanned. In parallel, the administrator of the RIG will execute the script to capture data using Putty. Once that script is running, the administrator will take a Photo and PointCloud using the RealSense SDK.

The entire process takes approximately ten seconds. Once the data is captured, it can be used in the photogrammetry pipeline for reconstruction.



(a) Front View Lidar



(b) Camera Lateral View

(c) Camera Side View

(d) Camera Lateral View

Figure 3.8: Example from Captured Dataset

3.2 Photogrammetry Reconstruction

As mentioned before, Meshroom will be used to process the images and reconstruct the scanned person once the data acquisition process has finalised. Meshroom will have a series of nodes as noted in section 2.2. It can be used in either windows or Linux. In this project, both Meshroom versions (2020 & 2021) were tested and fully worked. The chosen platform was Ubuntu 18.04. The host computer had a CUDA enabled GPU to speed up the execution speed in specific nodes.

After a series of multiple iterations & experimentation, the best parameters were selected, and it will be explained in section 3.2.

As Meshroom has a series of nodes that work in a Pipeline manner, all of the corresponding settings for the best results are detailed below.

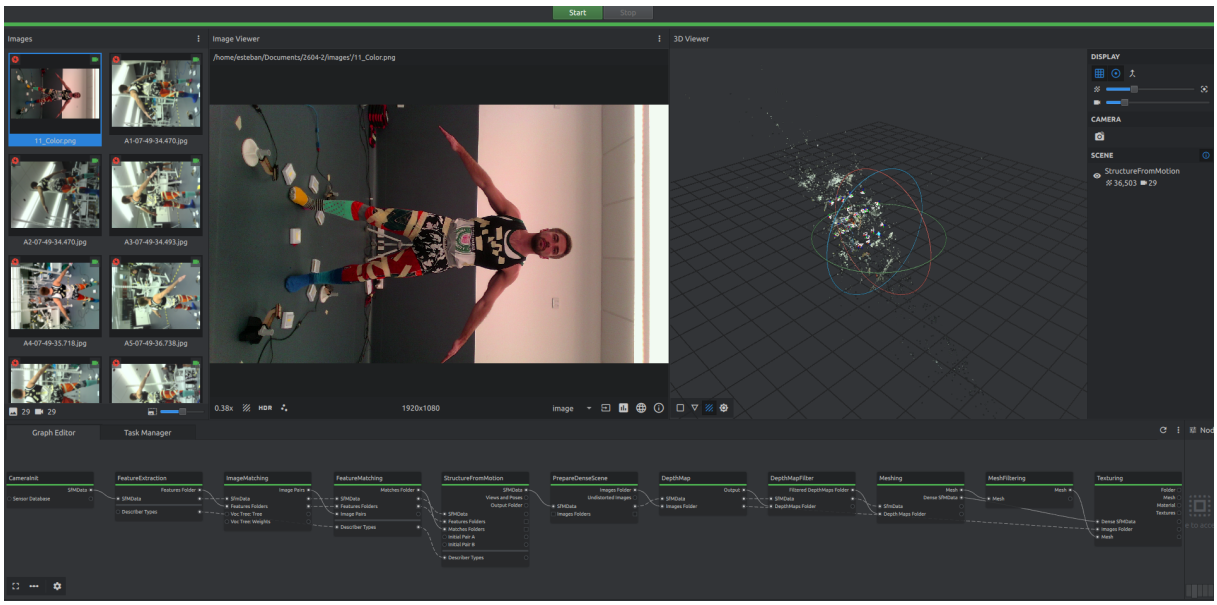


Figure 3.9: Successful Meshroom Pipeline (2021 version)

CameraInit Node

This is the Initial node of Meshroom Pipeline Reconstruction. In this node, all the captured data (images) will be imported. Once all the images have been imported into the initial node. All the available fields are illustrated on table 2.1. Nevertheless, in this section, the mentioned fields were configured, whereas non-mentioned settings remaining as default and are referenced on figure 3.10. These fields are described below

ViewPoints : In this field is necessary to edit the **Id, Pose Id, Intrinsic**.

For each image (viewpoint), the *Id, Pose Id, Intrinsic* have to be unique in order

to properly assign the camera calibration parameters from table 3.3. All of these fields will be directly mapped with the *Intrinsics Field*. This process will have to be performed for all twenty-nine images for this project.

Intrinsics : This Field is where all the camera calibration parameters from table 3.3 will be associated to all the configure ViewPoints (Images) from the previous field.
In section, it is crucial to map the *ViewPoint Id* from the previous section to the correct *Intrisics Id* for this field. Similarly, it is necessary to select the *Camera Type* as **Radial3** as this would allow inputting the Focal Lenght, Principal Point and Distorsion Parameters. Furthermore, the Width & Heigh of the Image is required to be imported. All these parameters are located on table 3.3.
On the other hand, the *Initialization Mode* has to be configured as **calibrated**, and the *Locked Box* requires to be checked as the Cameras (viewpoints) have been previously fully calibrated. This process will have to be performed for all twenty-nine images for this project.

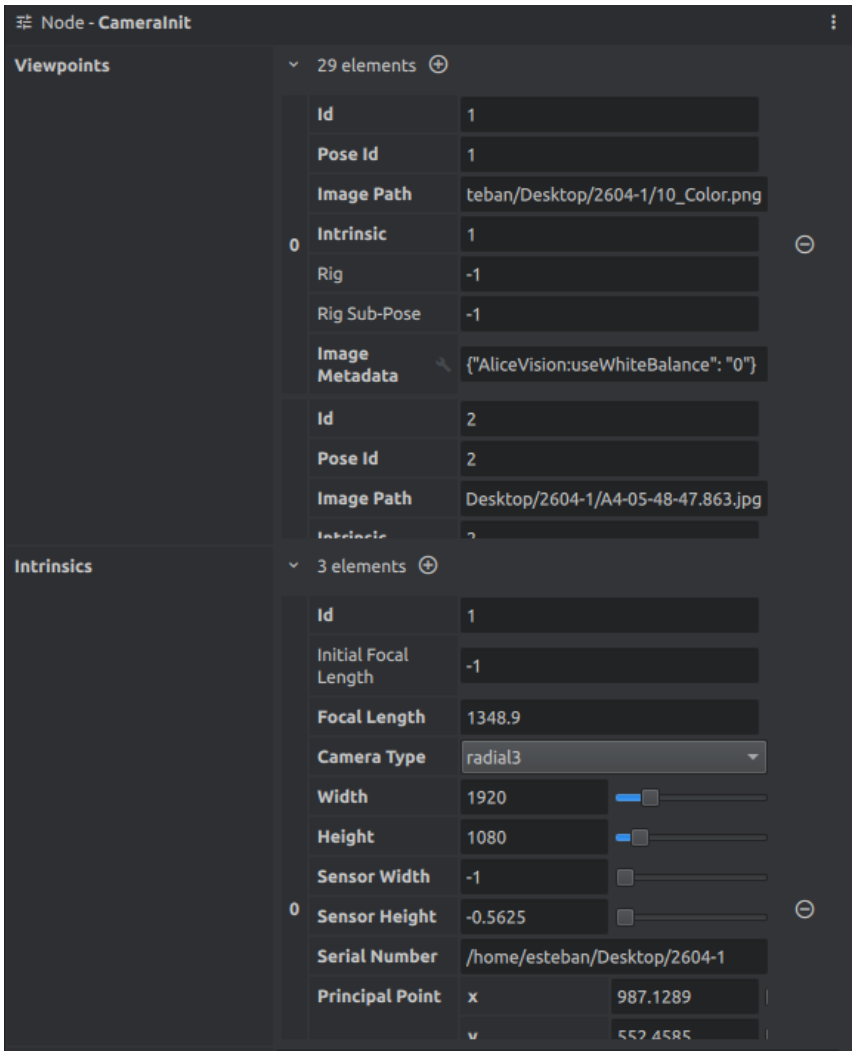


Figure 3.10: CameraInit Node Settings

FeatureExtraction Node

This will be the Second Node of Reconstruction Pipeline. All the field and settings of this node are located on table 2.2. For best results all the settings were configured as below:

Describer Types : The used describers are **SIFT** and **AKAZE**. After extensive testing it was found that a combination of *Sift & Akaze* will extract the most feature and enhance the quality of the reconstruction.

Describer Density : This field was configured as **Ultra** due to the small dataset (twenty-nine images)

Describer Quality : This was configured to **Ultra**, in order to retrieve the most amount of features from the images.

Contrast Filtering : Configured as **GridSort**

Grid Filtering : Checked field.

Force CPU Extraction : Unchecked Field. The reason behind this was to use CUDA in order to use the GPU to extract all the features to inprove the time of processing.

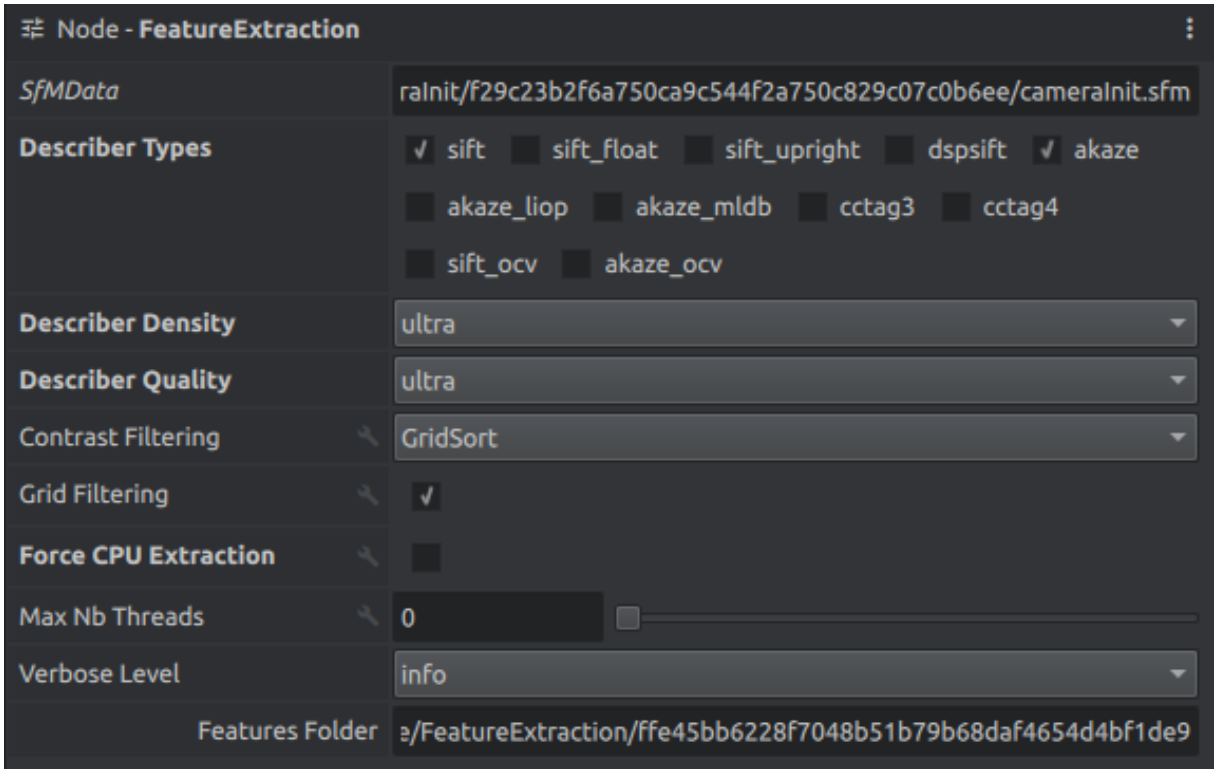


Figure 3.11: FeatureExtraction Node Settings

ImageMatching Node

This is the third node of the Pipeline Reconstruction. All the fields & settings for this node can be seen on table 2.3. The configured fields for best results are found below:

Method : The selected method for best results is **Exhaustive**. This method will export all image pairs for image matching process.

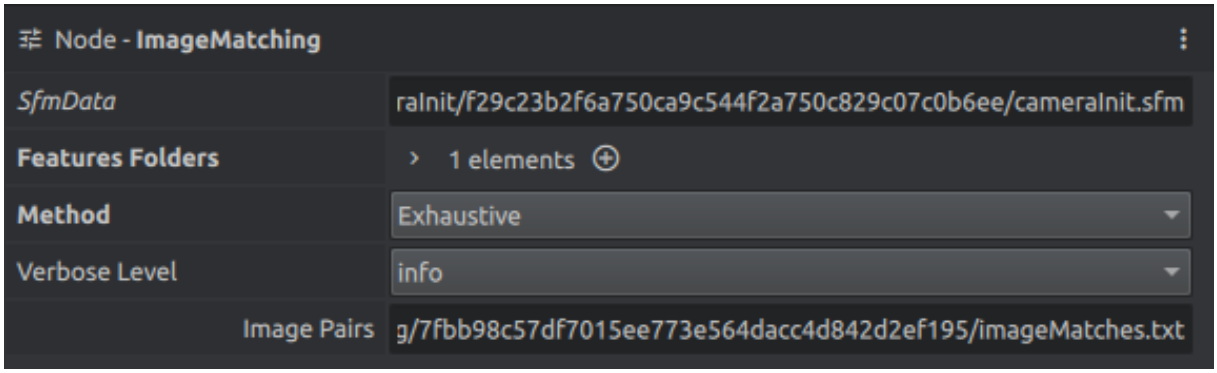


Figure 3.12: ImageMatchingNode Node Settings

FeatureMatching Node

This is the fourth node of the Reconstruction Pipeline. All the settings for this node can be referred to in table 2.4. The configured settings for best results are stated below.

Describer Types : The Selected Desciber have to match the selected on node the *FeatureExtraction Node*. Therefore the selected describers will **SIFT & AKAZE**

Photometric Matching Method : The selected method is **ANN_L2** *(Approximate Nearest Neighbor Matching)*.

Geometric Estimator : The selected Estimator **acransac** *A-Contrario RANSAC*.

Geometric Filter Type : The selected validation method to filter feature matches is **fundamental_matrix**.

Distance Ratio : The selected distance ratio to discard non-meaningful matches is 0.8.

Max Iteration : The selectd number of iteration for RANSAC step is 5000.

Known Poses Geometric Error : The maximum error specified for feature matching guided by information for cameras poses was selected as 5

All the other settings were configured as default and are referenced on figure 3.13.

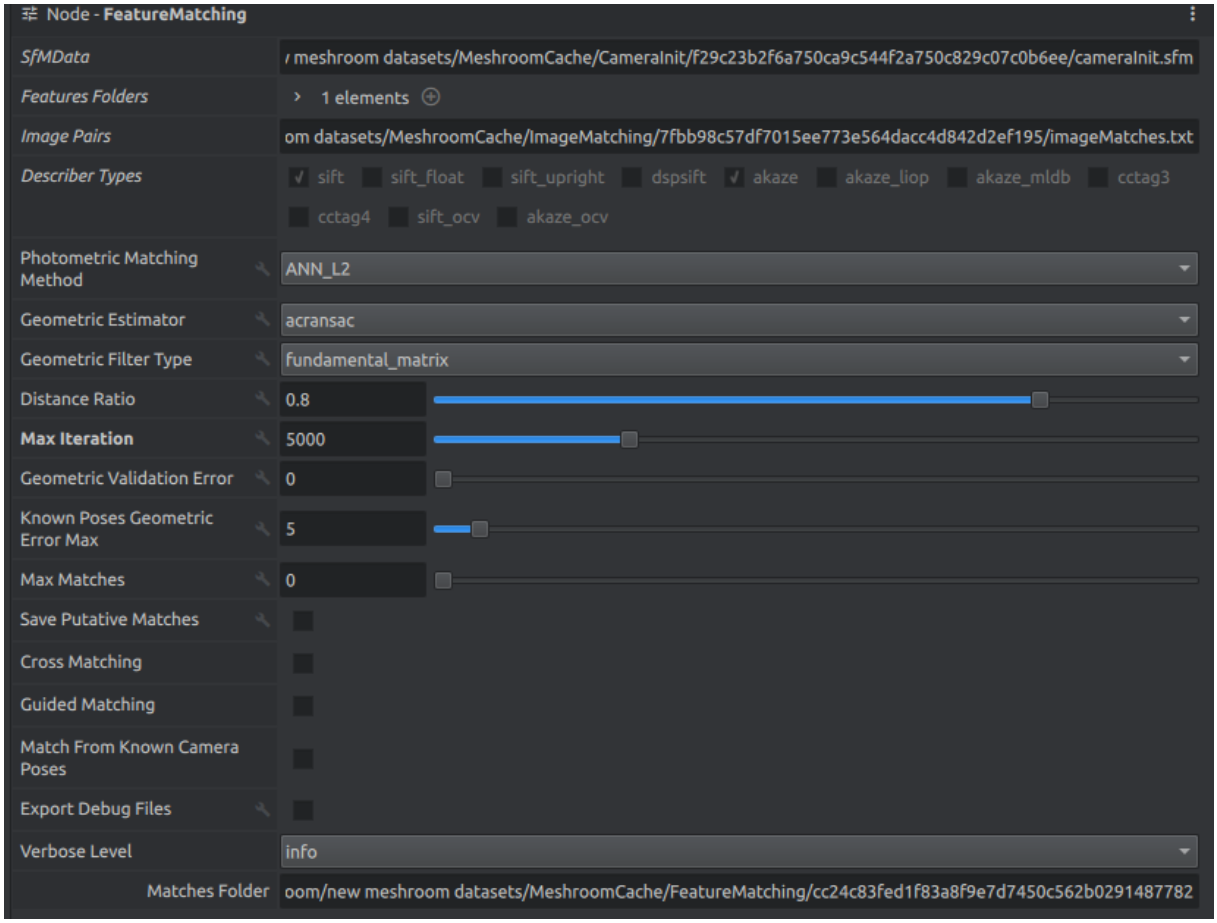


Figure 3.13: FeatureMatching Node Settings

StructureFromMotion Node

The Structure for Motion is the fifth node of the Reconstruction Pipeline. All the settings for this node can be referred to table 2.5. In this Step, it is possible to visualise the Structure along with the valid Viewpoints (images) and the number of feature matches. Furthermore, it is possible to observe the number of cameras that match the valid viewpoints and the corresponding Pose. As observed in figure 3.14, there are a total of 27883 feature matches and 29 valid viewpoints. Therefore, all the input data was valid, and the initial reconstruction process started.

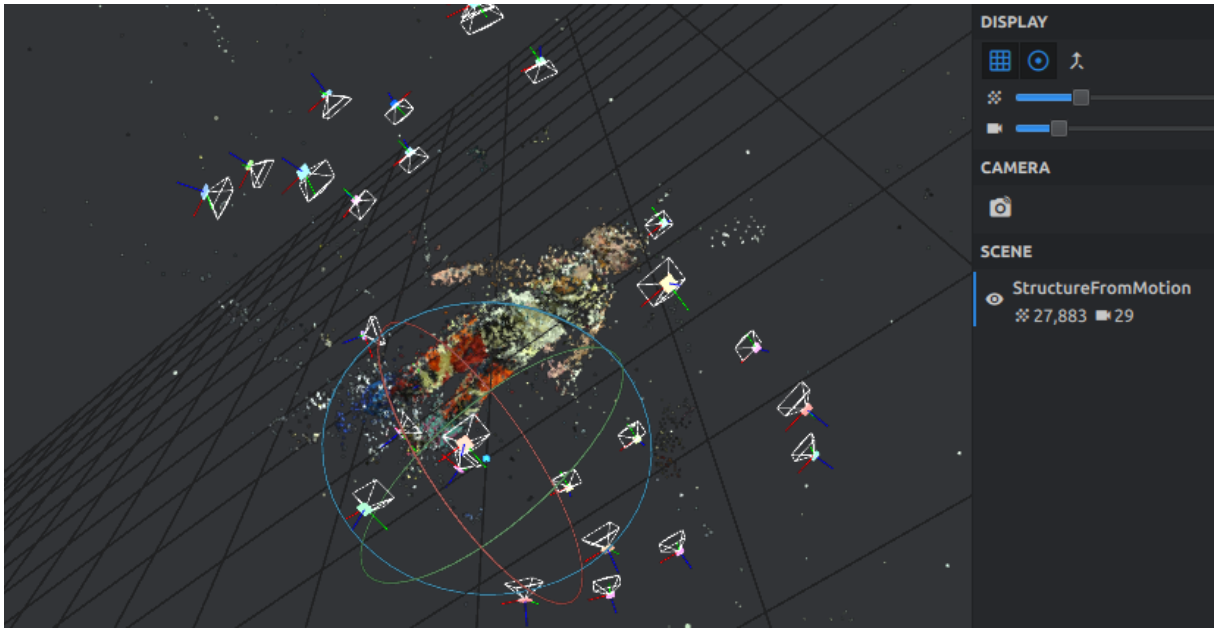


Figure 3.14: Structure From Motion Output

The configured settings for best results are stated below.

Describer Type : The Describer Type has to the match the ones used in previous steps. Hence, **Sift & Akaze** were selected.

Localizer Estimation : The estimator selected to localise the cameras poses is **acransac**.

Observation Constraint : The Selected Method for Observation Constrain Optimization is **Basic** .

Localizer Max Ransac Itereations : For an Optimal Results the selected number of iterations is 15000

LocalBA Graph Distance : The selected Graph-Distance Limit is 1

Min Input Track Lenght : Configured as 2

Min Observation For Triangulation : Configured as 2

Min Angle For Triangulation : Configured as 1

Min Angle For LandMark : Configured as 1

Max Reprojection Error : Configured as 4

Min Angle Initial Pair : Configured as 1

Min Angle Initial Pair : Configured as 40

Use Rig Constant : This field is selected.

Foce Lock of All Instrinsic Camera Parameters : This field is selected in order to force all the cameras intrinsics, as the cameras have been fully calibrated.

All the other fields were configured as default and are referenced on figure 3.15.

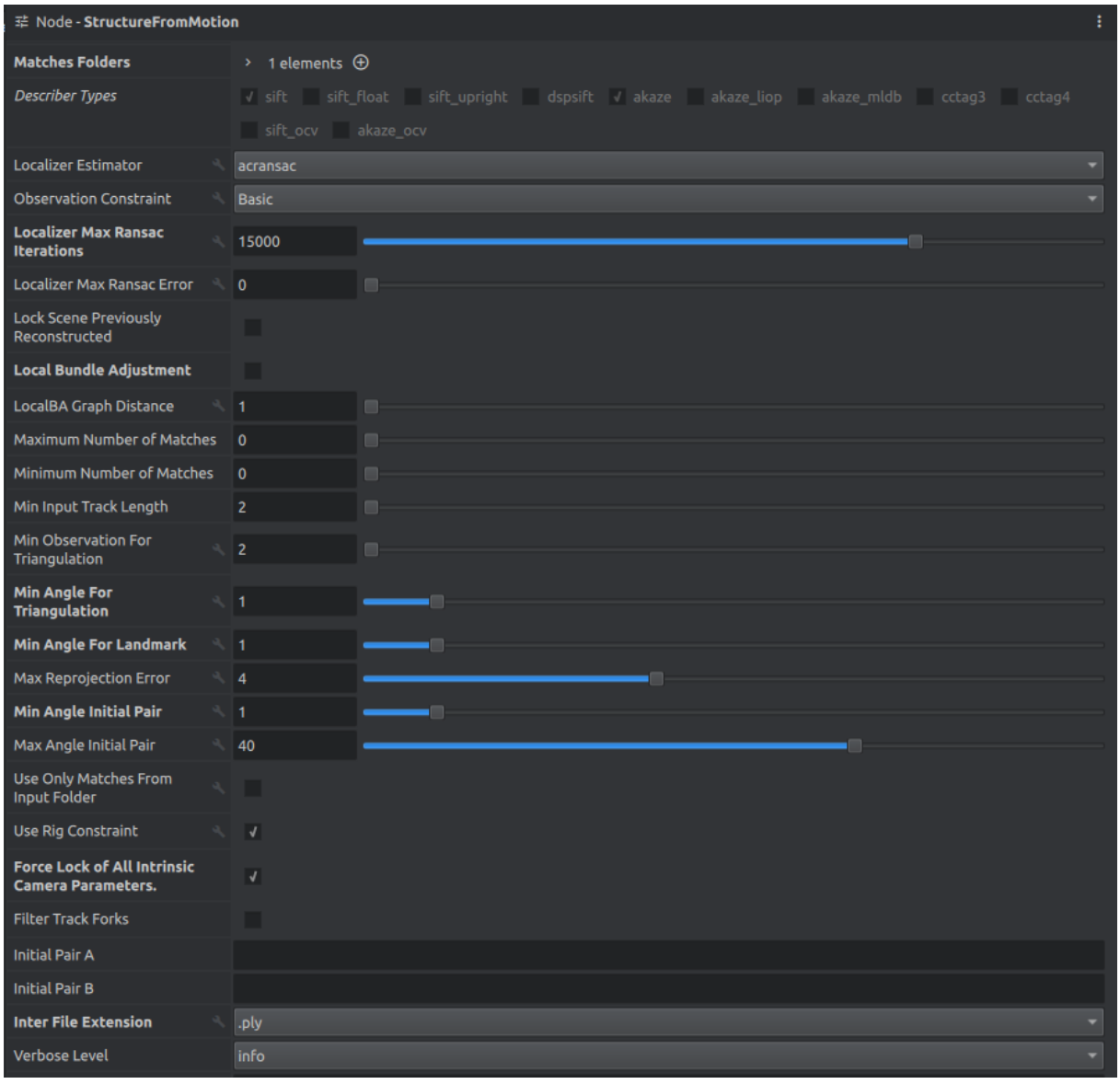


Figure 3.15: Structure From Motion Settings

PrepareDenseScene Node

The PrepareDenseScene is the sixth node of the Reconstruction. All the settings for this node can be refered to table 2.6. The configured settings for best results are stated below.

Output File Type : The choosen File Type for undistorted images is **PNG**.

All the remaining paramaters stayed as default and are referenced on figure 3.16.

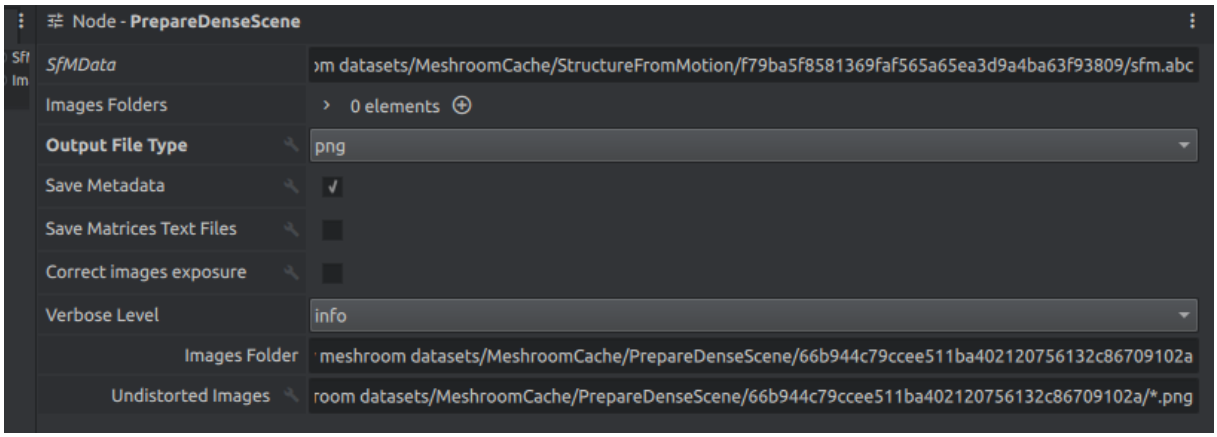


Figure 3.16: PrepareDenseScene Settings

DepthMap Node

DepthMap Node is the Seventh Node of Meshroom Reconstruction Pipeline. All the settings for this node can be referred to on table 2.7. The configured settings for best results are stated below.

DownScale : The Downscale Image factor is set to 1 for the best results.

SGM: Nb Neighbour Cameras : The Semi Global Matching Number of Neighbour Cameras selected is 10

Refine: Nb Neighbour Cameras : The selected Refine Number of Neighbour Cameras is 6.

All the remaining parameters stayed as default and are referenced on figure 3.17.

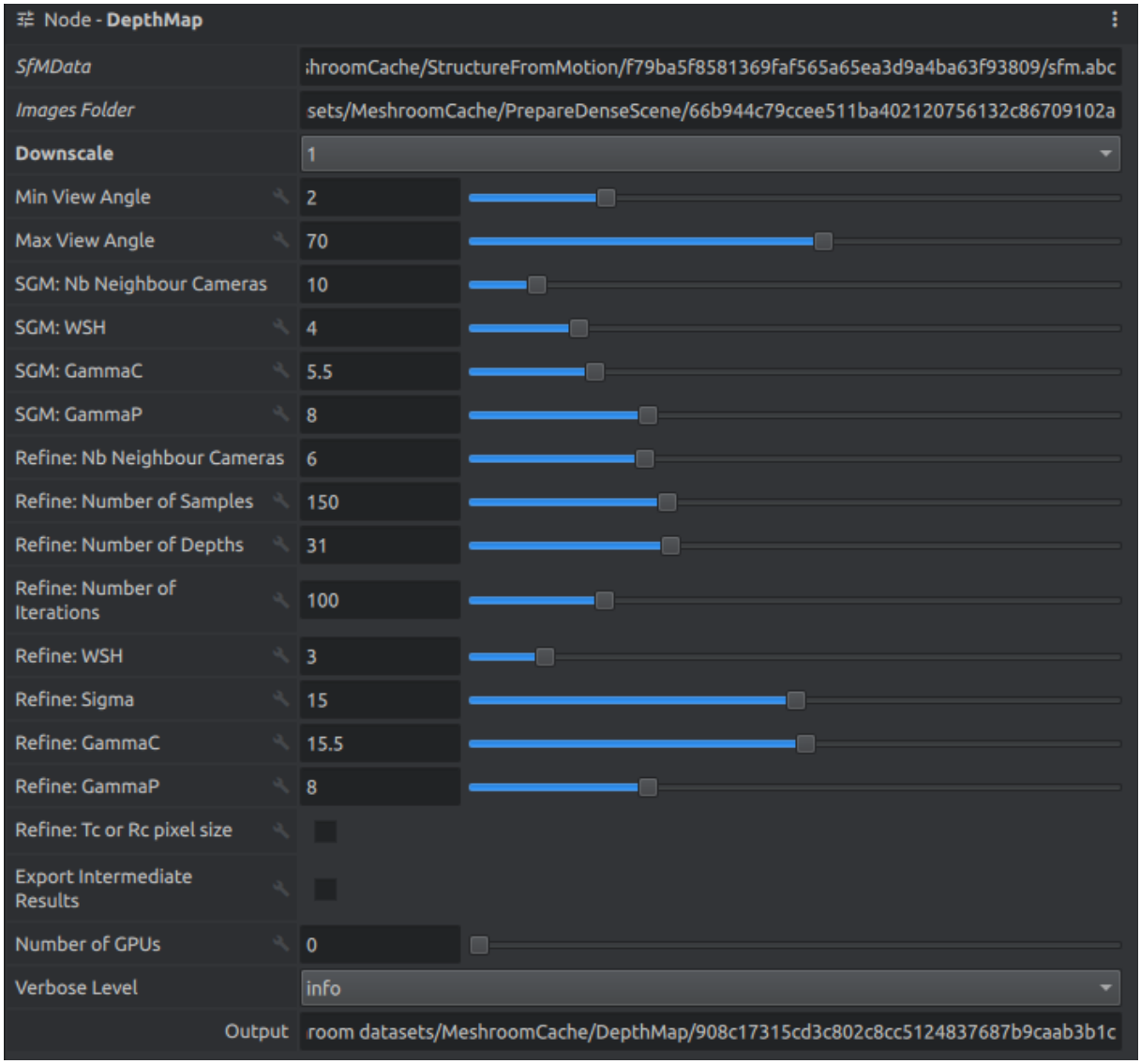


Figure 3.17: DepthMap Node Settings

DepthMapFilter Node

DepthMap Node is the Eighth Node of the Meshroom Reconstruction Pipeline. All the settings for this node can be referred to on table 2.8. The configured settings for the best results are stated below.

Min View Angle : The selected Minimun angle between two views is 2.

Max View Angle : The selected Maximum angle between two views is 70.

Number of Nearest Cameras : The selected Number of nearest cameras used for filtering is 10.

Min Consistent Cameras : The selected Number of minimum constent cameras is 3.

Min Consistent Cameras Bad Similarity : The Selected minum Number of consistent cameras with weak similarity value is 4

All the remaining settings are configured as default and are referenced on figure 3.18.

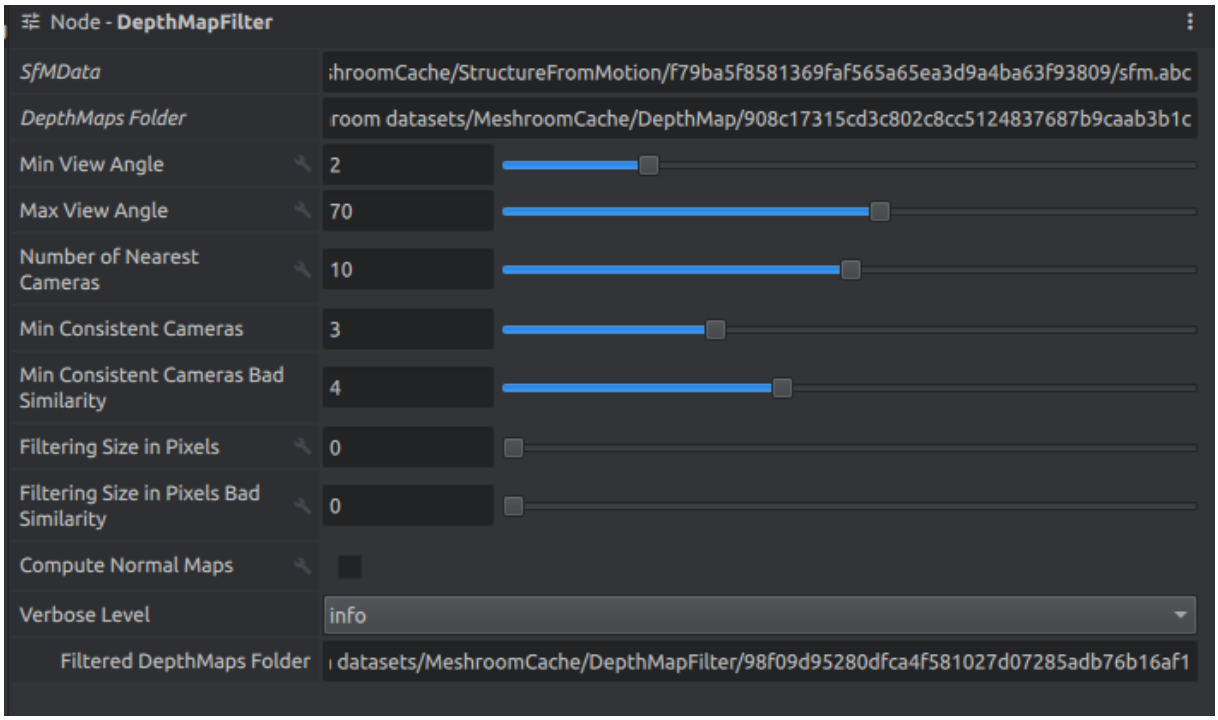


Figure 3.18: DepthMapFilter Node Settings

Meshing Node

The Meshing is the ninth step of the Meshroom Pipeline Reconstruction. This node will take all the results from the previous steps and it will produce a mesh file (*.obj file*) that can be later processed on the PCL framework. All the settings for this node can be referenced to table 2.9.

The configured settings for best results are configured below.

Estimate Space From SFM : This setting was selected to estimate the space from the results of the SFM node.

Min Observation For SFM Space Estimation : The minimum observation selected value for estimating SFM is 1.

Min Observation Angle For SFM Space Estimation : The selected value for an angle for SFM space estimation is 45. This angle should be modified based on the desired result.

Min Step : The selected step used to load the depth values from depth maps is 1

Partitioning : **singleBlock** was selected for best results.

Repartition : **MultiResolution** was selected for best results.

Refine Fuse : This setting was selected to refine the depth map fusion with new pixel size defined by the angle and similarity scores.

Weakly Supported Surface Support : This setting was selected to improve support of weakly supported surfaces.

Add Landmarks to the Dense Point Cloud : This field was selected in order to add Structure From Motion (SFM) Landmarks to the dense point cloud and improve the resulting mesh.

Colorize Output : This field was selected to generate a colourized output point cloud and mesh.

Add Mask Helper Points : This field was selected in order to add helper points on the outline of the depth maps masks.

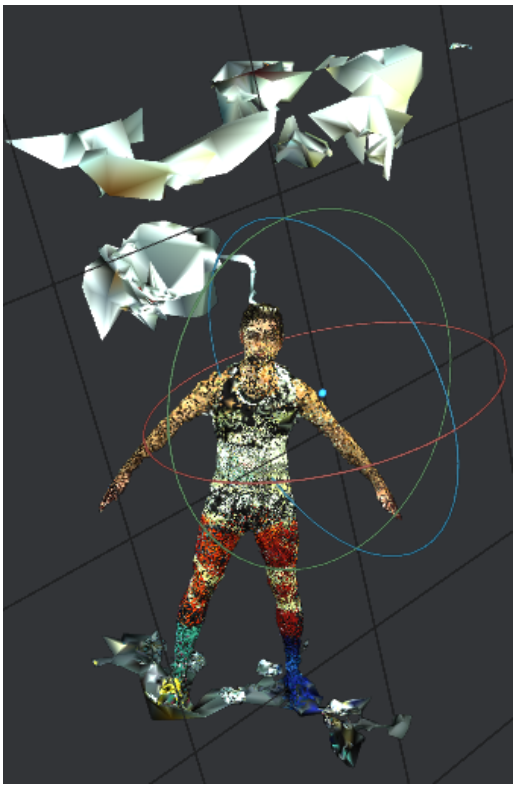
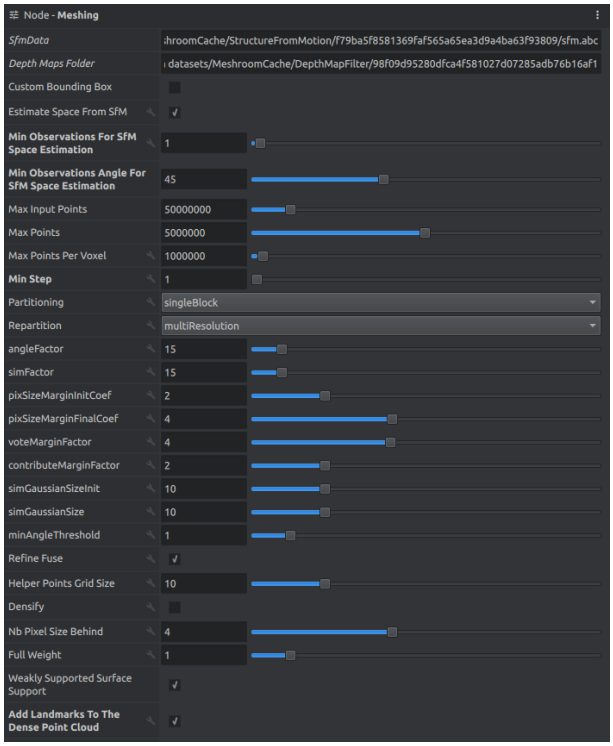
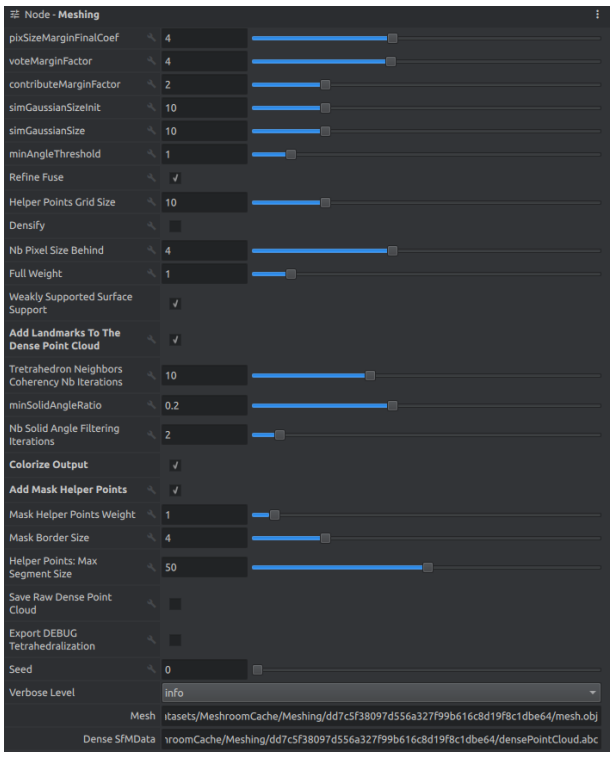


Figure 3.19: Meshing Node Output



(a) Meshing Node Settings



(b) Meshing Node Settings

Figure 3.20: Entire Meshing Node Settings

All the remaining settings are configured as default and are referenced on figures 3.20a and 3.20b. Full scale images in Appendix (A.2 & A.3)

MeshFiltering Node

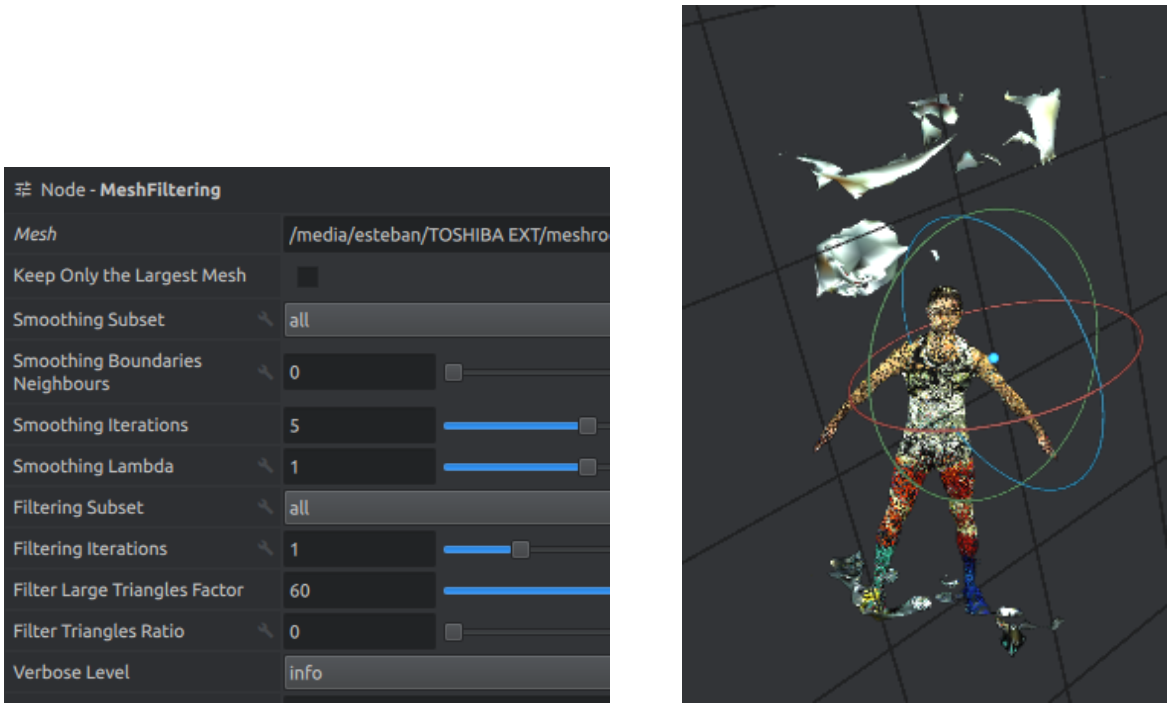
The MeshFiltering Node is the tenth step of the reconstruction pipeline. It smooths the resulting point cloud and mesh. The output of this step will be used in the Data processing framework based on the PCL library. All the settings for this node can be referenced to table 2.10.

The configured settings for the best results are configured below.

Smoothing Iterations : The selected number of smoothing iterations for best results is 5.

Filter Large Triangles Factors : The selected factor to remove all large triangles is 60.

All the remaining settings are configured as default and are referenced on figure 3.21a.



(a) MeshFiltering Node Settings **(b)** MeshFiltering Node Output

Figure 3.21: MeshFiltering Node Settings & Output

Texturing Node

The Texturing Node is the Final Step of the Meshroom Reconstruction Pipeline. It generates a textures mesh. All the settings for this node can be referenced to table 2.11.

The configured settings for best results are configured below.

Texture Side : The selected output texture size is 8192.

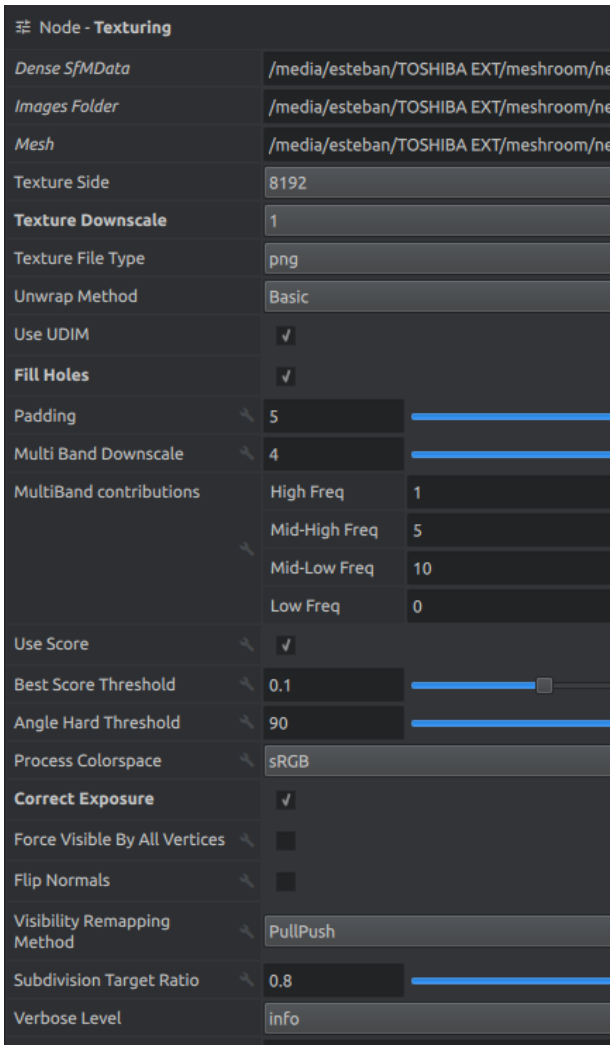
Texture Downscale : The selected texture downscale factor is 1.

Texture File Type : The selected texture file type is **PNG**.

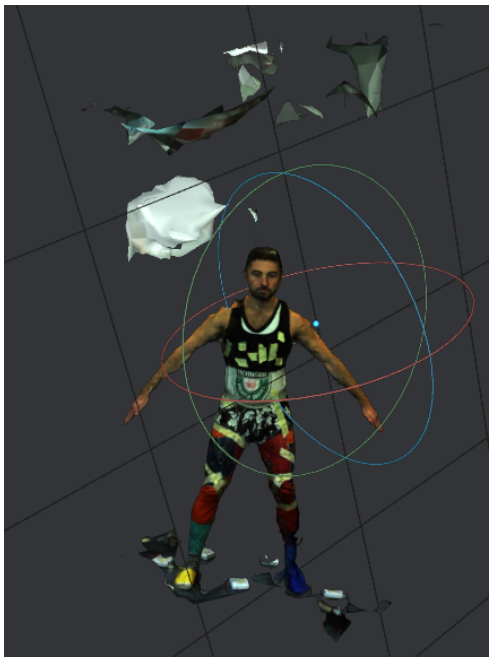
Fill Holes : This setting was selected in order to fill texture holes with plausible values.

Correct Exposure : This setting was selected in order to uniformize the exposure value from the images.

All the remaining settings are configured as default and are referenced on figure 3.22a.



(a) Texturing Node Settings



(b) Texturing Node Output

Figure 3.22: Texturing Node Settings & Output

3.3 Data Processing Framework

The Data Processing Framework is a custom developed wrapper that allows processing. This framework was developed using the PCL framework and was built using modern C++ on a Linux Platform (Ubuntu 18.04.5 LTS).

To deploy this framework, refer to the following GitHub Repository. It includes all the dependencies for installation with the corresponding steps. **3D Reconstruction Framework.**

This custom-developed framework used the concepts from the PCL library and its corresponding algorithms, which are detailed in section 2.3. On the other hand, a different framework (ScaleRatioICP) was attached and incorporated as a wrapper to add extra features and functionalities. Nevertheless, this wrapper was not implemented in the final executable.

The framework consists of two wrappers *(classes)* and one executable file *(Main)* which will be explained below.

3.3.1 Wrappers (Classes)

As mentioned before, the framework will have two specific classes that were developed and adapted. The main class is the custom-developed class that is based on the PCL library and uses the algorithms mentioned in section 2.3. This class is called **Reconstruction**.

This class can input all the output data from the photogrammetric reconstruction process (*Meshroom*) and the captured lidar point cloud. All the detailed functionalities (Variables & APIs) can be found in the Github Repository mentioned above. Nevertheless, the general overview will be explained below.

On the other hand, the other wrapper is a developed class that was adapted from the following GitHub Repository **ScaleRatioICP**. However, after substantial testing, this wrapper was not implemented in the final executable of the developed framework.

3.3.1.1 ScaleRatioICP

This developed framework was developed by Lin et al. (2013) and it is a method to scale 3D point clouds. It estimates the ratio of scales of two given point clouds by performing PCA of spin Images over different scales of two point clouds; then, it generates two sets of cumulative contribution rate curves, which can be considered to illustrate the scale of the given point clouds.

This wrapper processes the input point clouds and sends the result of the cumulative contribution rate curves to MATLAB via the APIs to obtain the final scale ratio between two input point clouds.

However, after substantial testing, it was discovered that the cumulative contribution rate curves process was highly time-consuming. Furthermore, in this particular project, the final scaling result was inconsistent. As the lidar point cloud, which is used to estimate the scale, only has the front view of the person, it has some errors in the calculation process. Therefore, the implementation of this class in the final executable was not appropriate.

3.3.1.2 Reconstruction

The Reconstruction Class is based on the PCL library and the algorithms used in section 2.3. This section will provide an insight into the class APIs and the overall functionality. For a detailed look refer to the GitHub Repository **ScaleRatioICP**.

The description of the developed APIs (*Class Public Methods*) will be explained below.

setInputCloud : This member class function will be used to input a given Mesh or PointCloud. It designed to work with common **.ply & .obj** formats.

normalsDirectionAdjustment : This method is designed to input a given Pointcloud and output a different pointcloud with Adjusted Normals. It will get the vertices of each face and recompute the normals using the *"right hand rule"*. Furthermore, it will normalize the normal vector and stack up the new normals in the output point cloud.

pointCloudOutliersFilter : This API takes an reference cloud and outputs a processed point cloud with filtered data outliers. This method is based on the Statistical OutlierRemoval Filter algoritm referenced on section 2.3.1. It requires to have configured the following fields.

- ***setMeanK*** : It configures the number of points (K) to use for mean distance estimation.

- ***setStddevMulThresh*** : It Sets ths standard deviation multipler threshold. It considers all the points outside $\mu \pm \sigma * std\_mul\_threshold$ as outliers where μ is the estimated mean and σ is the standard deviation

pointCloudPlaneSegmentOperation : This API is degined to input a given Point Cloud and output a plane segmented point cloud. The logic behing this algorithm is based on section 2.3.4. Similarly it uses a Kdtree to search through the pointcloud as referenced on section 2.3.5. Also it uses RANSAC as the segmentation method and

Model Type. For details about RANSAC refer to section 2.3.4.1. For this method it is required to configure the following fields.

- ***setMaxIterations*** : This parameters set the number of iterations for RANSAC analysis.

- ***setDistanceThreshold*** : This paramater sets the distance threshold for the plane model.

In a nutshell, this API aims to segment(remove) the plane model for any point cloud that has been used as input.

pointCloudClustering : This API is designed to Input a specific point cloud and run an Euclidean Cluster Extraction (refer to section 2.3.6). The Clusterized point cloud will output the biggest point cloud cluster as the valid target point cloud. In order to define a cluster, it is crucial to determine the *Minimum Cluster Size and the Cluster Tolerance.* Once all the clusters are generated, these will be stacked in a vector. In this vector, an algorithm will iterate throughout all elements and compare their cluster size. Based on this process, it will output the largest cluster as the valid output point cloud.

pointCloudNormaliseUpscale : This API aims to smooth the surface of the Point Cloud via MLS (*MovingLeastSquares*). Furthermore, it also upsamples the point cloud to a defined parameter using *RANDOM_UNIFORM_DENSITY* as the prefered method (refer to section 2.3.8). For this specific API uses a Kdtree (refer to section 2.3.5) for data searching and it is required to specify the *Search Radius* for determining the k-nearest neighbor for best fitting. This method required an input point cloud, and it will output a smoothed and upsampled point cloud.

poissonReconstruction : This API will require an input pointcloud and it will generate a reconstructed Polymesh based on the Poisson Reconstruction Algorithm (refer to section 2.3.9). For this API is required to configured the following parameters:

- ***setDepth*** : It sets the maximum depth of the tree(kdtree) that is used for surface reconstruction.

- ***setOutputPolygons*** : Enables to use an polygon mesh (polymesh) as output.

- ***setSamplesPerNode*** : Sets the minimum names of sample points that fall within tree.

- ***setSolverDivide*** : Sets the depth at which block Gauss-Seidel Solved is used to solve the Laplacian Equation.

- ***setIsoDivide*** : Sets the depth at which a block extracts the iso-surface.

- ***setScale*** : Sets the ratio between the diameter of the cube used for reconstruction as well as the diamater of the bouding cube samples.

downsamplePointCloudVoxelGrid : This API aims to downsample the input point-cloud through a voxel grid filter(refer to section 2.3.3). It requires the user to input the *LeafSize in x,y,z* of the Voxel Grid. Based on the given LeafSizes, it will downsample all the data.

pointCloudScaleAdjustment : This API uses a custom-developed algorithm to scale on the point cloud. It required to input the target point cloud and the reference point cloud. This developed algorithm is based on PCA (refer to section 2.3.7). It calculated both centroids of the input & reference point clouds as well as the corresponding eigenvectors. With his values, it will estimate the transform for each point cloud. Based on these results, it will calculate the Minimum and Maximum Points in **X, Y, Z** axis. Based on this it will obtain the scale in axis based on the following equation, where axis applies to the *X,Y,Z axis*

$$Scale_{axis} = \frac{cloudRefMaximumPoint_{axis} - cloudRefMinimumPoint_{axis}}{cloudInputMaximumPoint_{axis} - cloudInputMinimumPoint_{axis}} \tag{3.1}$$

With the Obtained values for the Scale in X, Y and Z from 3.1, it is possible to fully scale the input cloud based on the obtained results. The final results will be stored in a vector for later processing.

pointCloudCentroidAlignmentAdjusment : This API aims to align two given point cloud to its corresponding centroid. It is based on PCA (refer to section 2.3.7). Once it is properly aligned, the user can rotate in any axis (x,y,z) the output point cloud with respect to its centroid. This aims to correct the centroid orientation and increase the accuracy of the scaling process.

transformVerification : This API is used to Verify the error Transform error between the two point clouds. It is used to determine the transformation error between the scaled point cloud and the referenced point cloud (lidar). It uses PCA (refer to section 2.3.7). to determine the transforms (T) of the two point clouds. Then it obtains the translation and rotational error between the two transforms.

The Translation error is defined as:

$$translation\_error = \left\| \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} \right\| \tag{3.2}$$

The Rotation error is defined as where $R(\hat{n}, \theta)$ is the rotation componet of the transform

$$Rotation\_error = \left\| \log(R_1(\hat{n}, \theta) * R_2(\hat{n}, \theta)^{-1}) \right\| \tag{3.3}$$

With both translation and rotation error, it is possible to compare to a user-defined threshold and ensure that the error is within acceptable margins.

passThroughFilter : This API aims to filter a specific point cloud based on the specific axis limit, which is known as *passThroughFilter* (refer to section 2.3.2). It requires the user to input an input point cloud, a string with the axis to filter, and the maximum and minimin limits of the constrained axis. This is designed to aid with the plane segmentation process and ensure appropriate limits for the scaling process.

In addition to the mentioned above API's there area several other such as:

- getScale
- cloudToPolyMesh
- convertPLYtoPCL
- convertPolyMeshtoPLY
- convertPCLtoPLY
- importPLYtoPolymesh
- pointCloudOutliersFilterManual
- normalsReAdjustement
- normalsTesting
- pointCloudPlaneSegmentOperationManual
- transformPointCloud

All the extra APIs are used to add extra features when importing & data. Furthermore, it allows an extra level of customization to the main APIs.

3.3.1.3 Executable

The Executable is a program that was developed using the developed framework. For extended details on how what are the dependencies and how to install this executable, refer to the GitHub Repository(**3D Reconstruction Framework.**)

In order to run this program is necessary to input two point cloud (the target Point cloud

for processing and the reference point clo[ud for scaling). Furthermore, it is necessary to specify if it is required to have output files or only be used for visualization. An example is as follows.

```
./3D_reconstruction_pipeline [Path_to_referenceFile] [Path_to_SCaling_file] [yes/no](Generate output files)
```

Example:

```
./3D_reconstruction_pipeline ../Data/mesh_ply.ply ../Data/10.ply no
```

There will be a series of multiple predefined point clouds in the main file that will be used for reconstruction. These will be used for both processing of input reference file as well as Input Scaling reference files. Similarly, there will be three objects derived from the class (*Reconstruction*) that are used to handle the reconstruction processes(input Reference File), Lidar and a helper object that Handles the optional generation of output files.

Figure 3.23, shows the output of the executable with a valid result.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

2021-05-31.01:25:56     Target  Pointclouds Scaled and Aligned and stored

2021-05-31.01:25:56     SCALING FINISHED


2021-05-31.01:25:56     Starting SCALING

2021-05-31.01:25:56     scale: 1.06258

2021-05-31.01:25:56     Target  Pointclouds Scaled and Aligned and stored

2021-05-31.01:25:56     SCALING FINISHED


2021-05-31.01:25:56     NORMALIZING & UPSCALING
2021-05-31.01:25:57     Smoothed and Upsampled 204218 data points from Smoothed file with the following fields:

2021-05-31.01:25:57     ANALYSING.............

2021-05-31.01:25:57     Transform 1
  0.999069  0.0136684 -0.0409293  -0.015201
-0.0148426   0.999483 -0.0285226 0.00831712
 0.0405183  0.0291035   0.998755   -1.92959
         0          0          0          1

2021-05-31.01:25:57     Transform 2
  0.997306  0.0470585 -0.0562683 -0.0380475
-0.0482872   0.998619 -0.0206793  0.0183849
 0.0552175  0.0233406   0.998201   -1.92571
         0          0          0          1

2021-05-31.01:25:57     Translation Error in cm= 2.52653 %       Rotation Error in cm = 5.27637 %

               T1*T2` =
   0.999323  -0.0337463   0.0146295   0.0516133
  0.0338365     0.99941 -0.00596233  -0.0202513
 -0.0144196  0.00645333    0.999875  -0.0047816
          0           0           0           1

2021-05-31.01:25:57     SUCESSFULL.............


2021-05-31.01:25:57     Poission  Reconstruction STARTED

2021-05-31.01:25:58     Poission Valid

**************Not Saving Files.         Please Review Existing Ones********************


**************Starting ----->Point Cloud Viewer********************
```

Figure 3.23: Pipeline Execution Log

ReconstructionMain Object

The ReconstructionMain Object aims to manipulate and process the input point cloud data. This Object aims to input the given point cloud and do all the processing that includes inputting the starting point cloud and using a series of filters to remove noice &outliers. It aims to contain the scanner person as the region of interest. Similarly, this objects scales the point cloud based on the result and the processed lidar point cloud. Once the point cloud has been appropriately scaled, it will use the Poisson Reconstruction Algorithm to Produce a Polymesh of the Reconstruction point cloud. For a detailed look at the APIs used for this Object, refer to the Repository (**3D Reconstruction Framework.**)

Lidar Object

The Lidar Object is made to handle and process the input data from the lidar that will be used for scaling. This Object will process the captures point cloud in order to maintain only the region of interest of the point cloud that will be used in the Scaling process in the Reconstruction Object. It will use several filters and segmentation algorithms to keep the regions of interest as accurate as possible to ensure an efficient scaling. For a detailed look at the implemented APIs, refer to the Repository(**3D Reconstruction Framework.**)

Helper Object

The helper object is created with the option to save the pipeline of the point clouds. These auxiliary objects save all the steps of the point clouds if the last argument of the executable is configured to **YES**. For a detailed look at the implemented APIs, refer to the Repository(**3D Reconstruction Framework.**)

Viewer

Similarly, there is a viewer implementation that allows viewing the output of the pipeline. This Viewer allows the user to observe the different steps of the reconstruction and scaling pipeline. It allows the user to observe the steps and make changes in different components of the pipeline if needed in order to adjust to new datasets.

Figure 3.24, illustrates the *Viewer* with all the corresponding steps of the reconstruction pipeline.

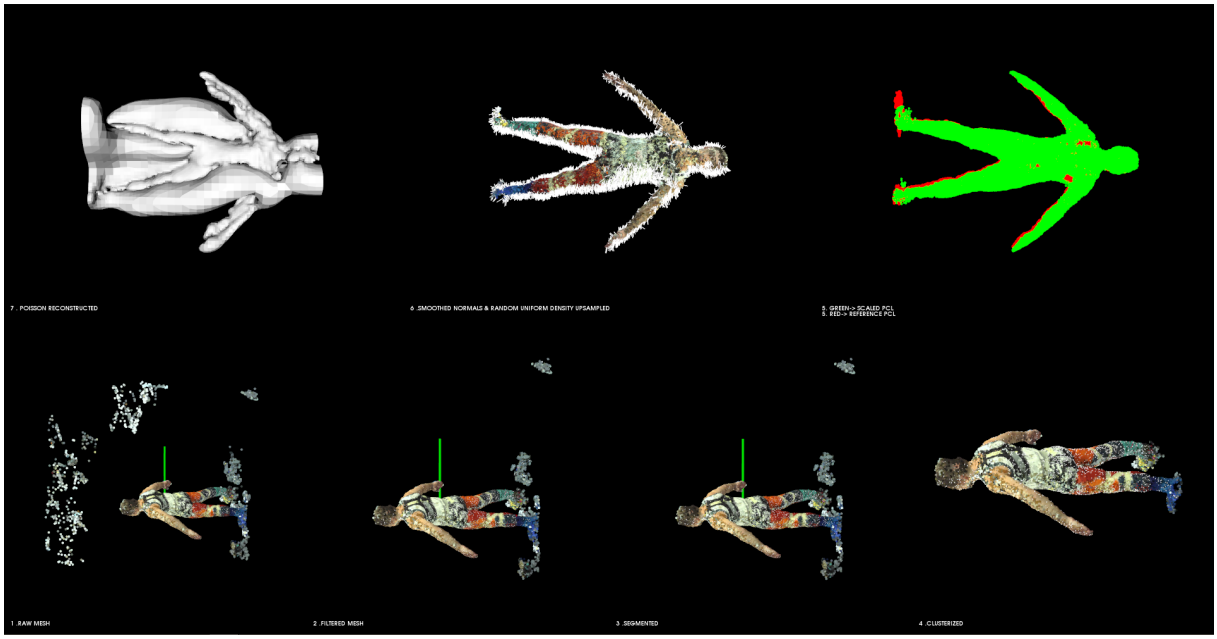


Figure 3.24: Pipeline Viewer Final Result

Open3D Poisson Reconstruction

In Certain occasions when the Poisson Reconstruction from the PCL pipeline fails. The created *poisson_open3d.py* script can be used to rerun the Poisson Reconstruction Algorithm and obtain a successfully reconstructed mesh. This script will take the outputs files from the PCL framework and generate a valid Poisson Reconstructed Mesh and ensure the robustness of the developed framework for this project. For a detailed look at the implemented script, refer to the Repository(**3D Reconstruction Framework.**)

Figure 3.25, illustrates all the steps of the Open3D Poisson Reconstruction.

(a) Imported Pipeline Genareted Mesh

(b) Initial Open3D Possion Reconstruction

(c) Analyse And Process Region of Interest

(d) Final Open3D Reconstructed Mesh

Figure 3.25: Open3D Poisson Reconstruction Script Outputs

3.3.1.4 Expected Behaviour

The expected behaviour is to input both files into the script via the terminal's arguments and enable to save the files for tracking. Once the input files have been fed to the framework, the mentioned objects above (ReconstructionMain & Lidar) will process in parallel both files and will remove outliers, plane segment and execute multiple filtering processes. Once Both files have been processed, the Initial Scaling process will start. After the Initial scaling process is finished, another filtering process will start in order to improve the region of interest of the point cloud. Afterwards, another scaling process will be executed as this will increase the accuracy of the final scale.

Moreover, the centroids of the point clouds will be aligned, and the processed Pointcloud will be transformed to the reference frame of the Lidar Point Cloud. Once the transformation and alignment process is completed, the resultant point cloud will be smoothed and upsampled. Furthermore, the pipeline will estimate the transformation error of the final point cloud with respect to the Lidar to ensure that the final pose and scale are within acceptable parameters.

Finally, the Poisson reconstruction process will generate the output point cloud.

Once all the processes finish, it will generate the output files if the option was selected. It will start the viewer to contextualize all the process steps with the corresponding expected output point clouds.

For a detailed look at the process flow of the program, refer to the appendix on figure A.1.

Results

After several iterations and different datasets, the results vary due to the differences in the captured data (images). These variations in the datasets cause inconsistencies in the photogrammetric reconstruction process. Therefore, this produced some meshes that had partial holes in certain regions. This is explained in section 4.1 and include the strategy to solve these issues in order to have a successful reconstructed photogrammetric mesh output from Meshroom.

On the other hand, inconsistencies in the given caused some issues in the developed reconstruction framework. Similarly, it is required to analyse if the scaling and transform of the input point cloud were successful in comparison to the reference point cloud captured with the Lidar. The obtained results are explained here, and the stipulated configured threshold to ensure a successful process. Furthermore, it explains the issues with the Poisson Reconstruction Algorithms and the implemented solution of developing an Open3D script to tackle these issues and ensure a successful Poisson reconstructed Mesh that can be used for external applications.

4.1 Meshroom

As mentioned on section 2.2,Meshroom is a Photommetric Software with a series of multiple nodes. All nodes are connected in a pipeline manner and follow a specific order as mentioned on section 2.2.

Several datasets were processed during the experimentation process with a series of different settings in all the different nodes. Many of the challenges in this process were that not all viewpoints (images) were successfully accepted in the StructureFromMotion Node, which causes inaccuracies in the reconstructing process and did not produce completed meshes (scanned persons). On many occasions, the outputs had missing limbs or a deformed structure where non-accepted viewpoints are located.

After substantial investigation and experimentation with different settings in multiple nodes, such as testing different descriptors, methods, iterations, among others, the best results for this project were identified. Among the modified settings and parameters, it was found that map the Cameras Intrinsics to the corresponding images in order to avoid the intrinsic estimation guess. This allowed a ten per cent better result than the obtained results that did not include the actual camera intrinsics mapped to each image (viewpoint). Similarly, another critical setting was to use both *Sift & Akaze* as the descriptors. The inclusion of these allowed to detect of more features in each image. Therefore improving the feature matching and enhancing the SFM process, and increasing the number of valid viewpoints.

It is essential to ensure that the SFM process accepts all the images as valid viewpoints. This will allow obtaining an accurately reconstructed mesh with an even structure. All the configured parameters for all the nodes are mentioned in section 3.2.

However, there are still some considerations that need to be addressed to have an accurate and complete reconstruction. These considerations what sort of features are in the images. On many occasions, the clothing patterns and features of the scanned person played a key role when reconstructing the scanned person. In a nutshell, the more distinctive features in the scanned object, the better the overall final result will be.

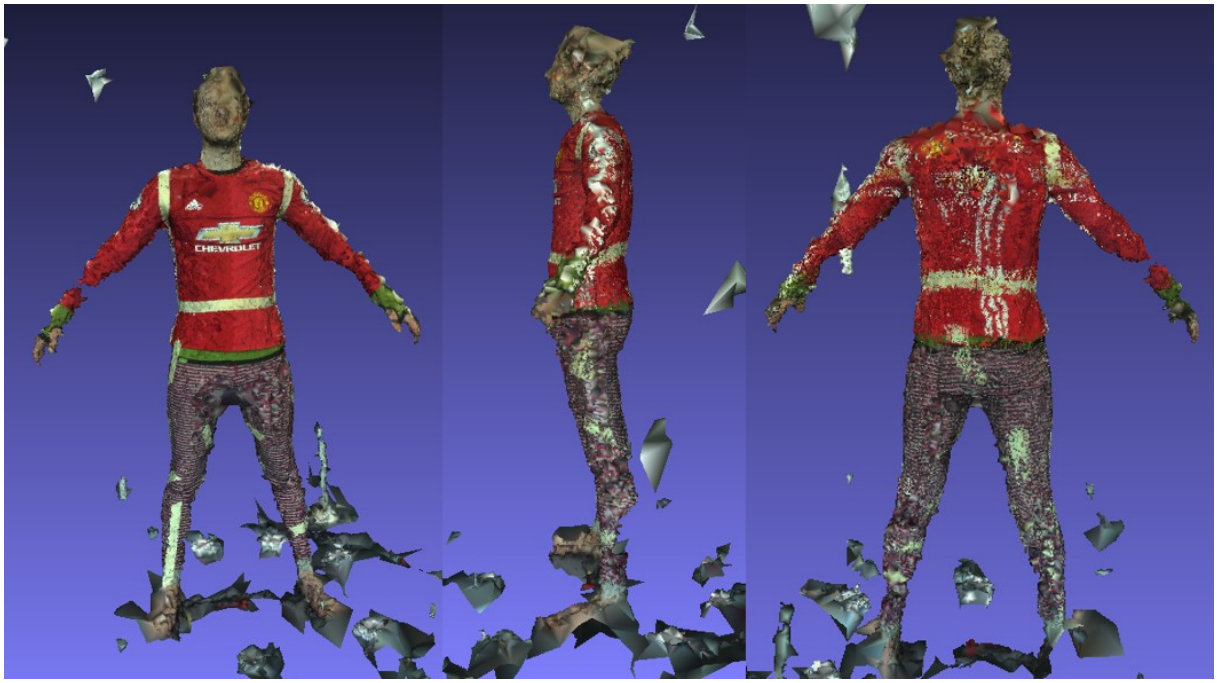


Figure 4.1: First Successful Scand

After several iterations of testing with different data, it was clear that the clothing features had to be distinctive to have an appropriate structure for the final mesh. The first successful scan result is observed in figure 4.1. In this Scan, the subject has the correct structure for the body, and the overall shape was accurate. However, after a detailed inspection, some holes were found in the Legs and Arms. The reason behind these holes that not enough features were detected in these areas. In the images, the subject used some pants that had repetitive patterns(strips). This pattern caused some issues as not enough feature matches were found due to this pattern. Similarly, there were some holes in the arms due to the lack of features in these areas. The reason is that not enough features were detected as it was a solid colour pattern, hence causing issues in the Reconstruction of these areas.

As the initial Scan had some issues in certain areas, therefore it was necessary to change the subject's clothing to generate new data and rerun the photogrammetric process (Meshroom).

With a modification to the clothing of the scanned subject, the result improved. Some extra patterns were manually added to the upper limbs (arms). These added patterns allow to detect and extra more features in these areas that caused issues in the first Scan (figure 4.1). Furthermore, the pants changed from a repetitive pattern structure to a solid pattern with manually added features.

The final result of this Scan can be observed in figure 4.2.

Figure 4.2: Second Successful Scan

As it can be seen, the issues mentioned around the arms were fixed as the manually added patterns allowed for better feature detection and matching. Nevertheless, in the lower area (around the legs), there are still holes. These holes were present in areas with only solid pattern, whereas the areas with the added patterns were fully reconstructed.

The main issue was that in the leg areas where it was mainly a solid pattern, no features were detected. Therefore, causing a lack of features matches, which produced inconsistencies in the reconstructing process. The result of the second can be observed in figure 4.2.

Finally, another custom scanning suit was created. This scanning suit had a multitude of features, colours and patterns. The custom morphing suit has very distinctive colour in different areas of the body, allowing it to have a significant number of features and obtain more features matches than before.

The final result of this Scan can be observed in figure 4.3. In this Scan, it can be observed that there are no holes in the entire body of the scanned person. Therefore, ensuring a consistent and accurate result that can be used in the reconstruction framework

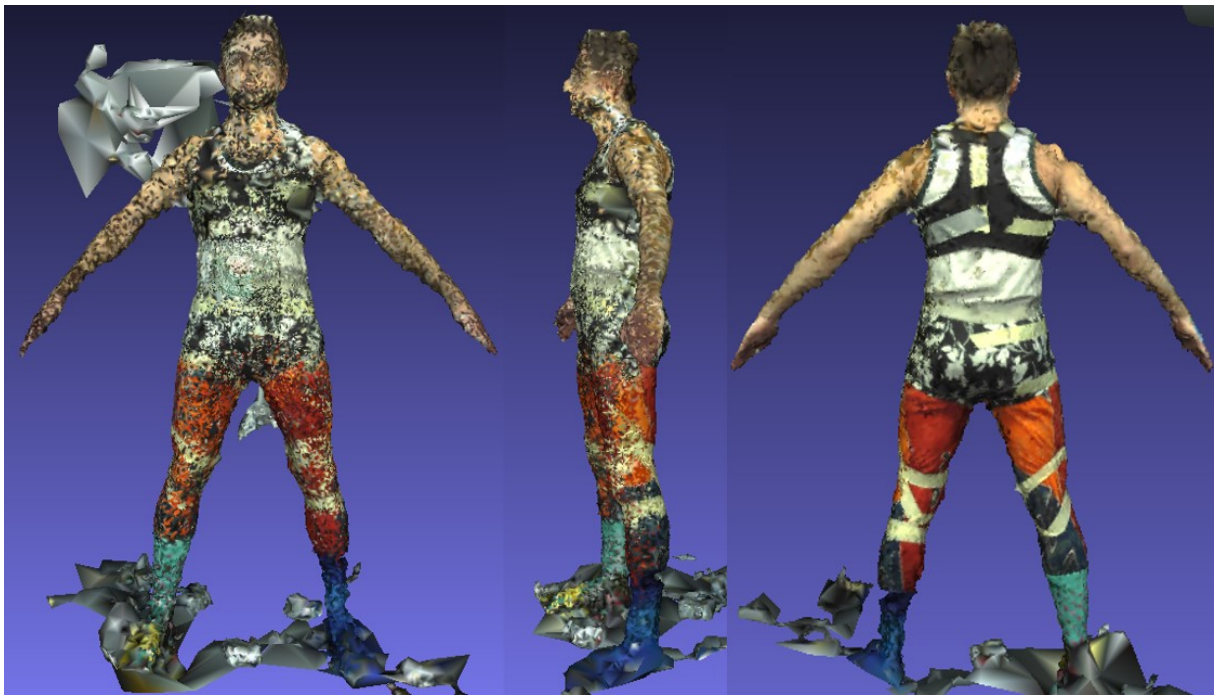


Figure 4.3: Final Sucessful Scan

4.2 Reconstruction Framework

The final results of the Reconstruction Framework were highly dependable on the results of the photogrammetric process output (section 4.1). The results from the reconstruction framework are consistent with different datasets. Minor adjustments have to be performed by the user in order to adjust to new data. However, the preconfigured values provide excellent coverage of most datasets. The expected result is the viewer with all the subsequent steps of the pipeline. All the steps of the reconstruction pipeline are mentioned on section 3.3.

On figure 4.4, the result of the pipeline reconstruction can be observed. The critical step is the Poisson Reconstruction steps, as this allows to reconstruct the final scaled pointcloud entirely. If the Poisson reconstruction step is successful, it should look similar to the top left section of the viewer output of the pipeline observed on figure 4.4.;

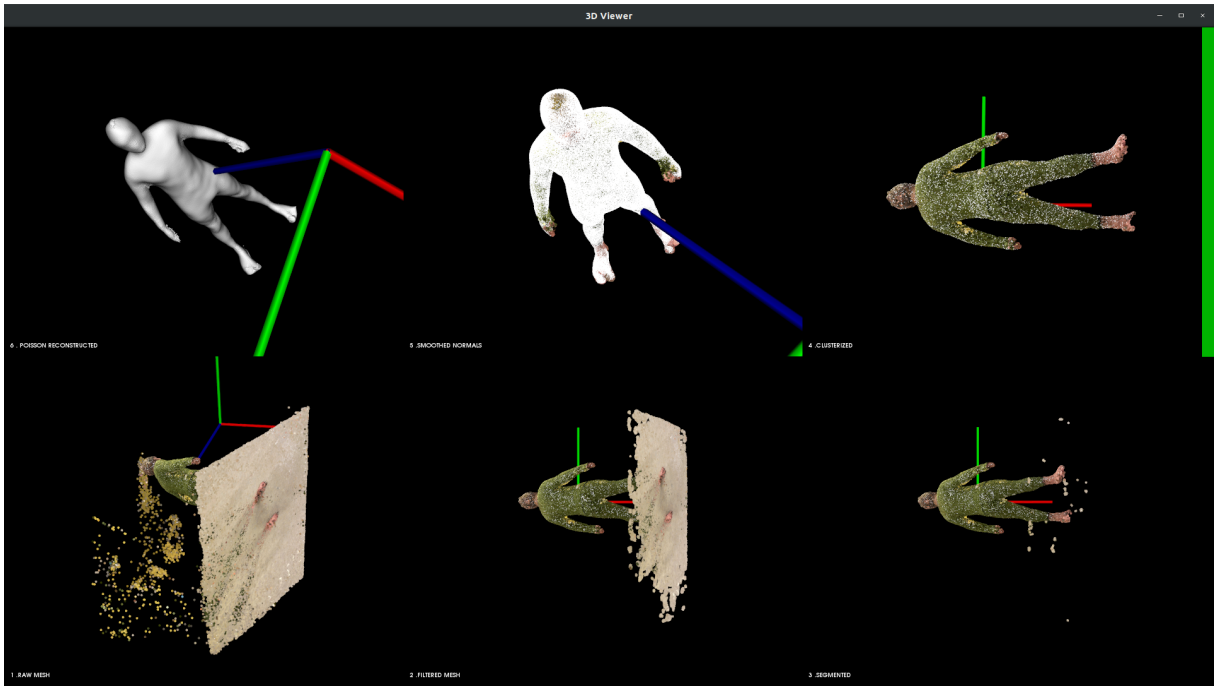


Figure 4.4: Reconstruction Framework Output

On certain occasions, the Poisson reconstruction will have some issues, as observed on the top left of figure 4.5. This, is due to an issues with the normals direction in certain datasets. Therefore it triggers issues and inconstencies in the Poisson Reconstruction Step.

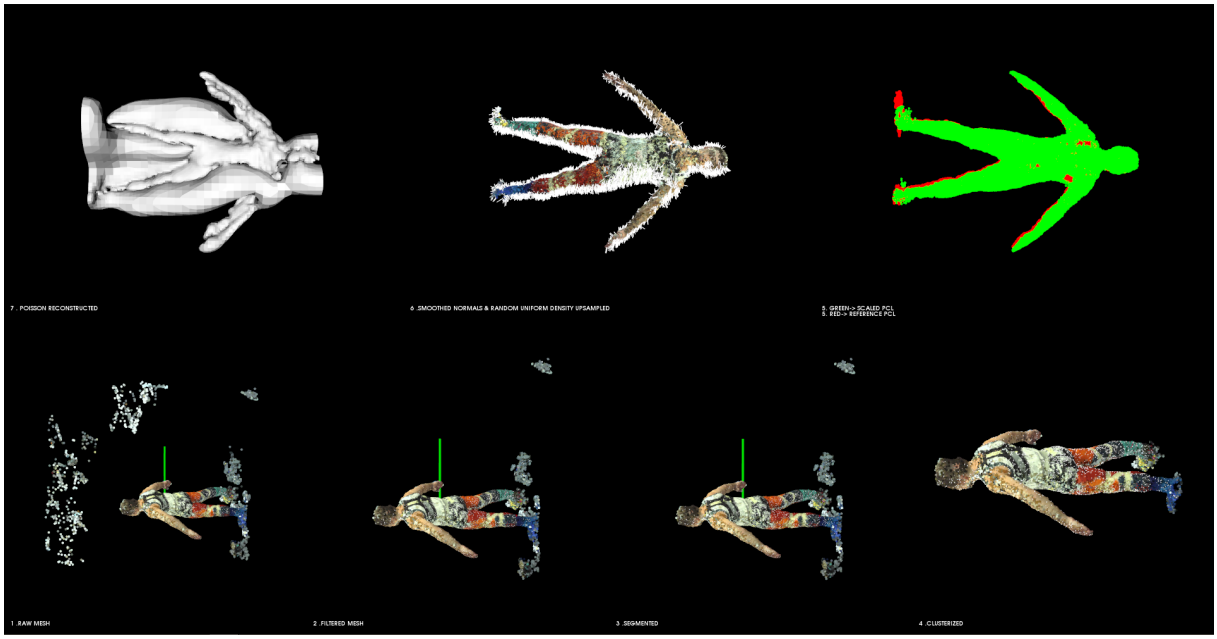


Figure 4.5: Reconstruction Framework Output Poisson Problem

In situations when this happens, it is necessary to run the *Open3D Poisson Reconstruction Script*. This script will adjust the normals, in order to have a consistent direction in the entire mesh and avoid the problem (refer to figure 4.6). Similarly, this script will adjust wrong inconsistencies in crucial areas of interest of the output mesh as seen in figure 4.6b. Afterwards, it removes the conflicting areas it will perform the Poisson Reconstruction again in order to have an accurate output as observed in figure 4.6c.

(a) Initial Open3D Possion Reconstruction

(b) Analyse And Process Regions of Interest

(c) Final Open3D Reconstructed Mesh

Figure 4.6: Open3D Poisson Reconstruction Script Adjustment

Conclusion

This project aimed at the development of a 3D scanner device that will produce accurate results. This report focused on the data processing and reconstruction of the acquired data from the scanned subject in order to generate a virtual scaled scanned representation of the person.

As previously mentioned, for this project, the RIG will capture a set of images used in a photogrammetric process using Meshroom to have a representation model of the person. The generated result will be processed in a reconstruction framework pipeline that will run several algorithms to improve the scan. Finally, the pipeline will scale the model accordingly and run the Poisson Surface Reconstruction in order to obtain the final Reconstructed model of the scanned user. The result will be subsequently usable for different applications such as clothing fitting.

Future development of this technology will allow more accurate scans and make this concept commercially viable for different industries such as online retailers, clothing manufacturing, among many others.

Recommendations

Based on the results obtained in this report and future development and improvements for this technology, the following recommendations are proposed.

- Create a Morphing suit with distinctive features and patterns that the user of the scan will use, which will allow having consistent features in every scan.
- Add more Cameras in order to have more viewpoints and improve the SFM process.
- Add better quality and resolution cameras in order to have more features in each image.

Bibliography

Aber, J. S., Marzolff, I., Ries, J. B., and Aber, S. E. (2019). Chapter 3 - principles of photogrammetry. In Aber, J. S., Marzolff, I., Ries, J. B., and Aber, S. E., editors, *Small-Format Aerial Photography and UAS Imagery (Second Edition)*, pages 19–38. Academic Press, second edition edition.

AliceVision (2021). Meshroom manual.

Allene, C., Pons, J.-P., and Keriven, R. (2009). Keriven r.: Seamless image-based texture atlases using multi-band blending. pages 1 – 4.

Artac, M., Jogan, M., and Leonardis, A. (2002). Incremental pca for on-line visual learning and recognition. In *Object recognition supported by user interaction for service robots*, volume 3, pages 781–784 vol.3.

Baumberg, A. (2003). Blending images for texturing 3d models. *Proceedings of the British Machine Vision Conference*.

Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of in-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26:1124–37.

Burt, P. and Adelson, E. (1983). A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2:217–236.

Cheng, J., Leng, C., Wu, J., Cui, H., and Lu, H. (2014). Fast and accurate image matching with cascade hashing for 3d reconstruction. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

Daanen, H. A. and Psikuta, A. (2018). 3d body scanning. *Automation in Garment Manufacturing*, pages 237–252.

Digumarti, S. T., Chaurasia, G., Taneja, A., Siegwart, R., Thomas, A., and Beardsley, P. (2016). Underwater 3d capture using a low-cost commercial depth camera. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9.

Fang, Z., Zhao, S., Wen, S., and Zhang, Y. (2018). A real-time 3d perception and reconstruction system based on a 2d laser scanner. *Journal of Sensors*, 2018:1–14.

Fernández Alcantarilla, P. (2013). Fast explicit diffusion for accelerated features in nonlinear scale spaces.

Fischler, M. and Bolles, R. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395.

Gao, W. and Tedrake, R. (2019). Surfelwarp: Efficient non-volumetric single view dynamic reconstruction. *CoRR*, abs/1904.13073.

Gautier, Q., Garrison, T., Rushton, F., Bouck, N., Lo, E., Tueller, P., Schurgers, C., and Kastner, R. (2020). Low-cost 3d scanning systems for cultural heritage documentation. *Journal of Cultural Heritage Management and Sustainable Development*, ahead-of-print.

Gonzalez-Rodriguez, A. (2020). Growing pains of ill-sized clothing for both consumers and brands.

Hirschmuller, H. (2005). Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 807–814 vol. 2.

Ichim, A. I. (2012).

Intel (2020a). Depth camera d435i intel realsense depth and tracking cameras.

Intel (2020b). Intel nuc mini pcs2020.

Jancosek, M. and Pajdla, T. (2011). Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR 2011*, pages 3121–3128.

Jancosek, M. and Pajdla, T. (2014). Exploiting visibility information in surface reconstruction to preserve weakly supported surfaces. *International Scholarly Research Notices*, 2014:1–20.

Jo, W., Kannan, S. S., Cha, G.-E., Lee, A., and Min, B.-C. (2020). Rosbag-based multimodal affective dataset for emotional and cognitive states.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, page 61–70, Goslar, DEU. Eurographics Association.

Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3d reconstruction in dynamic scenes using point-based fusion. page 8.

Kneip, L., Scaramuzza, D., and Siegwart, R. (2011). A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. pages 2969–2976.

Lepetit, V., Moreno-Noguer, F., and Fua, P. (2008). Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81:155–166.

Levin, D. (1998). The approximation power of moving least-squares. *Math. Comput.*, 67(224):1517–1531.

Levy, B., Petitjean, S., Ray, N., and Maillot, J. (2002). Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21:362–371.

Li, Y., Wang, S., Tian, Q., and Ding, X. (2015). A survey of recent advances in visual feature detection. *Neurocomputing*, 149:736–751.

Lin, B., Tamaki, T., Raytchev, B., Kaneda, K., and Ichii, K. (2013). Scale ratio icp for 3d point clouds with different scales.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–.

Magnenat-Thalmann, N., Seo, H., and Cordier, F. (2004). Automatic modeling of virtual humans and body clothing. *Journal of Computer Science and Technology*, 19(5):575–584.

Moulon, P., Monasse, P., and Marlet, R. (2012). Adaptive structure from motion with a contrario model estimation.

Muja, M. and Lowe, D. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. volume 1, pages 331–340.

Murcia, H., Monroy, M. F., and Mora, L. F. (2018). *3D Scene Reconstruction Based on a 2D Moving LiDAR.*

Newcombe, R. A., Fox, D., and Seitz, S. M. (2015). Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Nister, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770.

Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. volume 2, pages 2161 – 2168.

Otero, I. (2015). *Anatomy of the SIFT method*. PhD thesis.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Scharstein, D., Szeliski, R., and Zabih, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, pages 131–140.

Shah, R., Deshpande, A., and Narayanan, P. (2014). Multistage sfm: Revisiting incremental structure from motion. pages 417–424.

Shchurova, C. I. (2015). A methodology to design a 3d graphic editor for micro-modeling of fiber-reinforced composite parts. *Advances in Engineering Software*, 90:76–82.

Siena, F., Byrom, B., Watts, P., and Breedon, P. (2018a). Utilising the intel realsense camera for measuring health outcomes in clinical research. *Journal of Medical Systems*, 42.

Siena, F., Byrom, B., Watts, P., and Breedon, P. (2018b). Utilising the intel realsense camera for measuring health outcomes in clinical research. *Journal of Medical Systems*, 42.

Slavcheva, M., Baust, M., and Ilic, S. (2018). Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Spahiu, T., Shehi, E., and Piperi, E. (2014). *Extracting body dimensions from 3D body scanning.*

Strecha, C., Fransens, R., and Van Gool, L. (2006). Combined depth and outlier estimation in multi-view stereo. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2394–2401.

Strutz, T. (2016). *Data Fitting and Uncertainty (2nd edition)*.

Sturm, J., Bylow, E., Kahl, F., and Cremers, D. (2013). Copyme3d: Scanning and printing persons in 3d. *Lecture Notes in Computer Science*, pages 405–414.

Treleaven, P. and Wells, J. (2007). 3d body scanning and healthcare applications. *Computer*, 40(7):28–34.

Vedaldi, A. and Fulkerson, B. (2010). Vlfeat: An open and portable library of computer vision algorithms. volume 3, pages 1469–1472.

Yu, G. and Morel, J.-M. (2011). Asift: An algorithm for fully affine invariant comparison. *Image Processing On Line*, 1.

Zhou, Q.-Y., Park, J., and Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.

Appendix

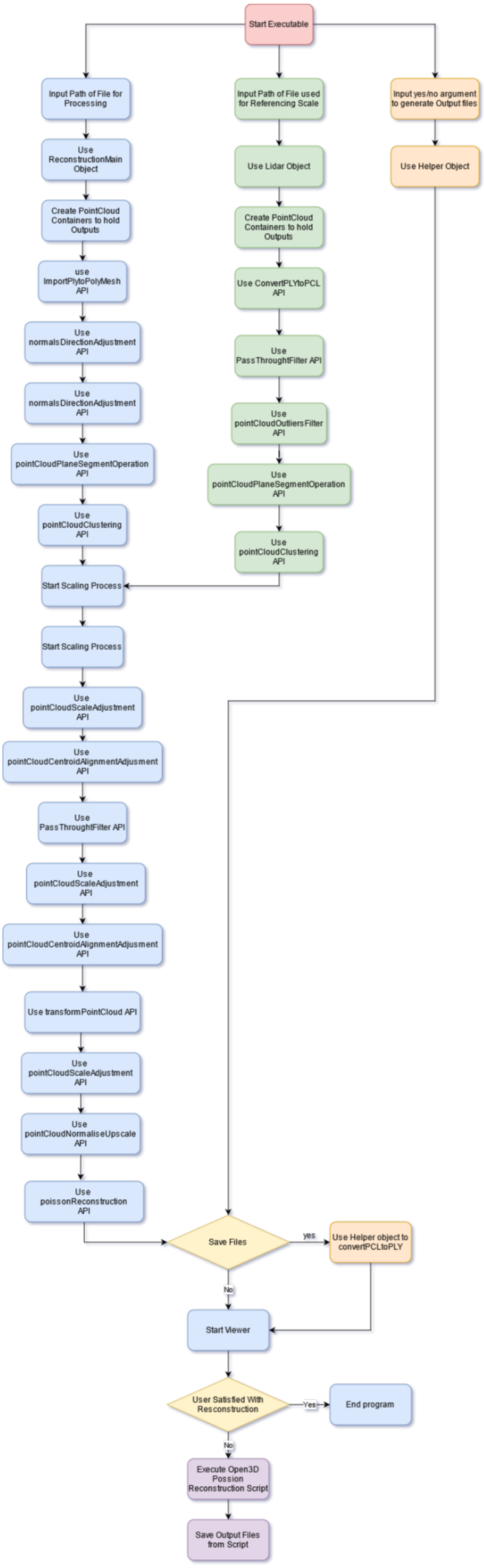


Figure A.1: Gantt Chart

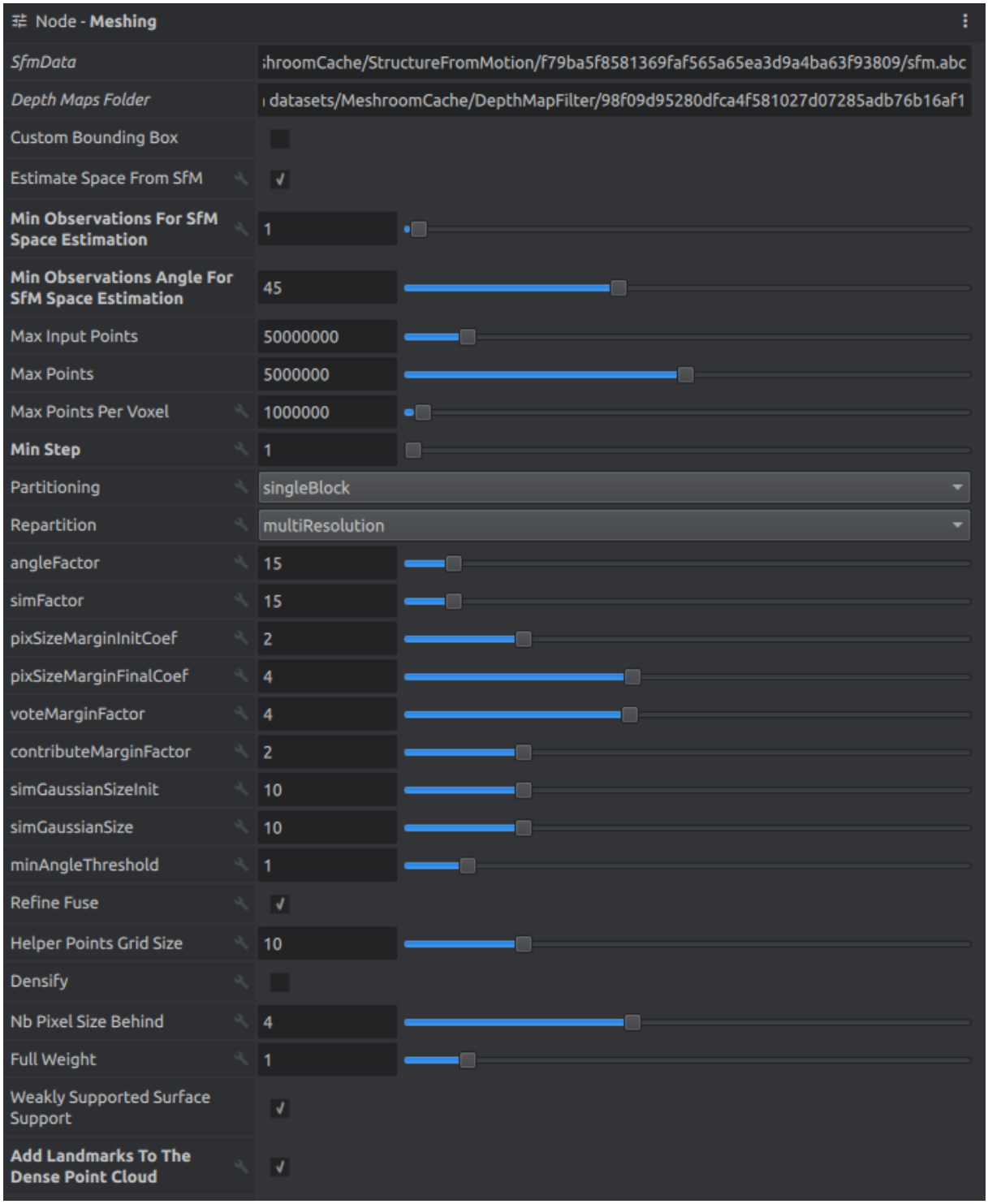


Figure A.2: Meshing Node Settings

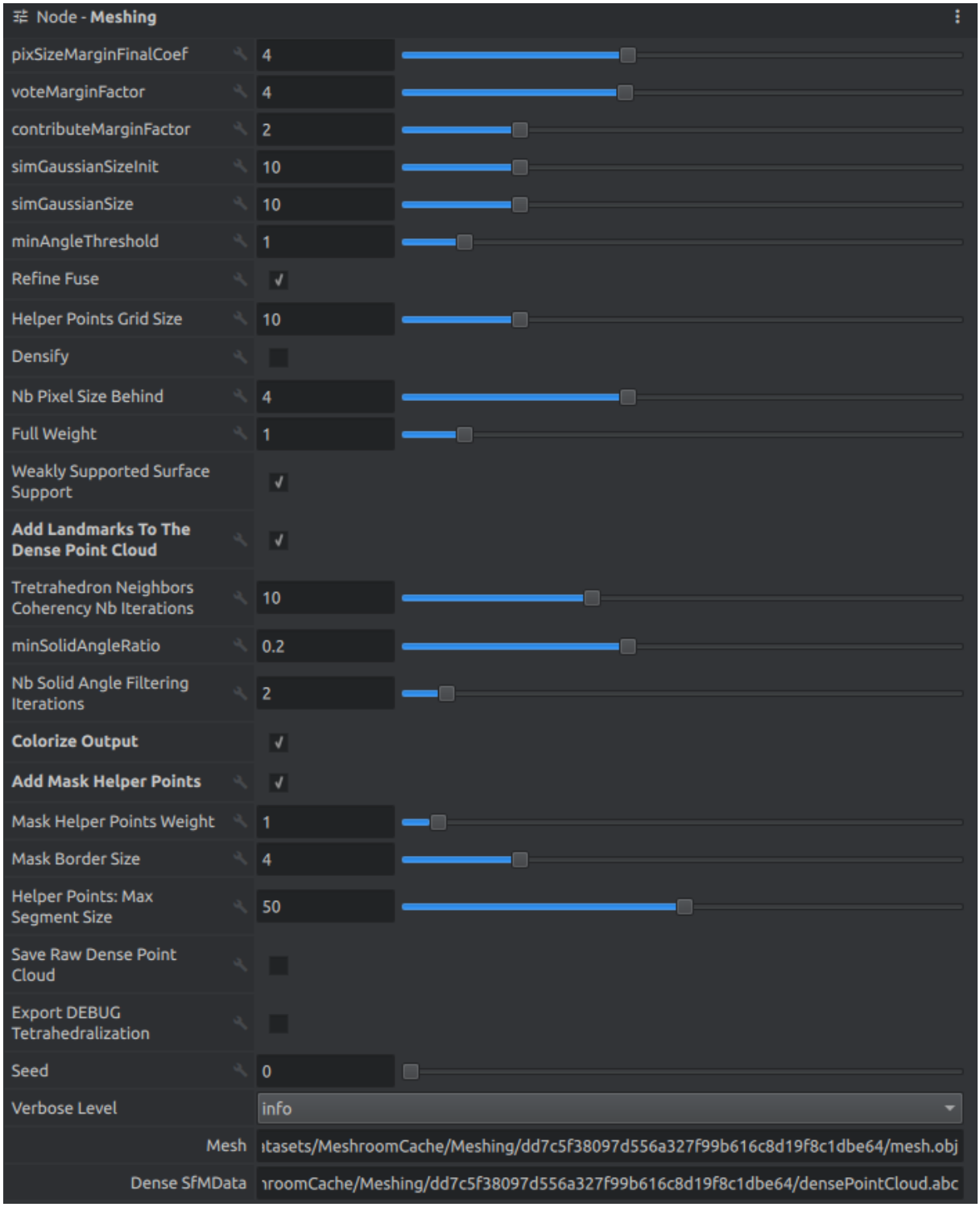


Figure A.3: Meshing Node Settings