Scanned surface to CAD design: Matching, Alignment and Difference Evaluation

Student: Marc Prat I Masó

Thesis supervisor: Cristina Cadevall Artigues

Tutor: Carlos Andujar Gran

Master in Innovation and Research in Informatics (MIRI)
Facultad de Informática de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC)

Abstract

This work proposes a tool to compare a high-precision surface scan with its original CAD model. The project is divided into three steps: how to process the CAD files, compute and optimize the registration, and develop tools for visualization.

Because CAD files can contain multiple representations, we can't work directly with them. Normally this is approached by triangulating the described components and simplifying the mesh for a fast rendering, but this doesn't work for our high-density scans. Instead, we need to process the CAD to obtain a point cloud with a parameterized distance between points —this will give a good starting point for the registration—.

Next, the registration can be divided into two parts, coarse and fine. For the coarse register, we adapt the Initial Alignment Sample Consensus algorithm (IA RanSac) to automate the configuration settings and optimize the time for our input size. While in the fine register we will use the classic Iterative Closest Point (ICP).

Due to the approach being a random consensus and the input being two big points cloud, reducing the number of points to a feasible number (statistically and computationally) will be essential to find a solution. For this, we developed a local optimizer that combines a set of LOD to find a global solution.

Finally, to analyze the result, we have developed a color visualization interface with a set of modifier tools (colormaps, transparencies, range modifiers, etc.). This allows us to detect discrepancies between the two models that can be caused by wear or manufacturing imperfections.

Acknowledgments

I would like to thank Jordi Mariné and Guillem Carles Santacana, who also contributed to this project during the weekly review. And my thesis supervisor and tutor, Cristina Cadevall Artigues and Carlos Andujar Gran, especially for helping me to overcome the thesis writing.

Also, I want to thank my couple, Ana, for supporting me and putting up with my empty talk. And my dad for encouraging me and giving me the confidence I needed.

Contents

1. Introduction

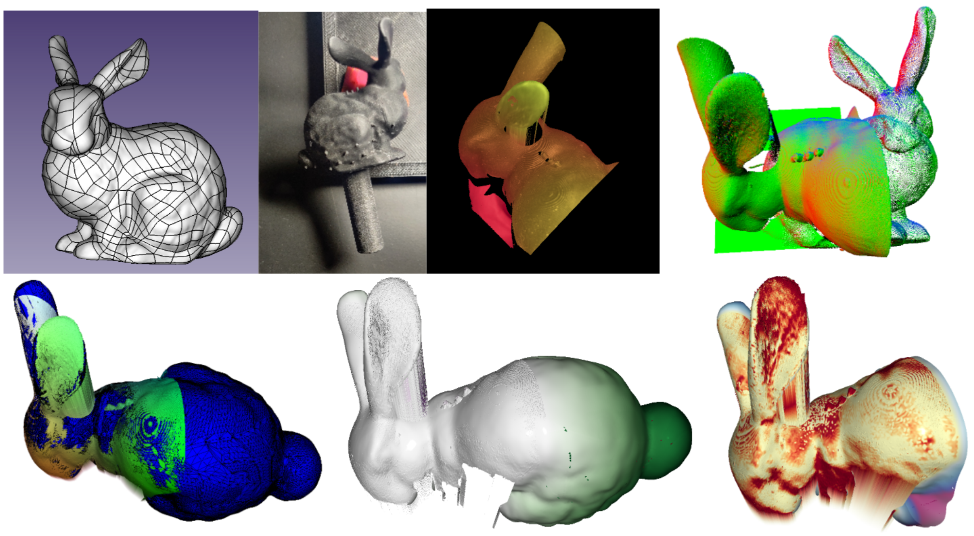


Figure 1.1. In the top row, the input: CAD model design, a 3D printed version sample, its 3D scan, and the final point clouds inputs in the program. In the bottom row: The result after registration, a difference evaluation, and better visualization for the detail.

This project has been done under an agreement with Sensofar Tech. SL, where I'm currently working. Sensofar is a leading-edge technology company within the field of non-contact surface metrology.

One major use for the Sensofar scanners is to study the wear in pieces. This is known as a tribology study, an iterative process to evaluate the wear and durability: first, the sample is used multiple times, then it's scanned, and finally compared with the reference model. For the moment the clients use other companies' software, but these tools aren't prepared for our scanner's quality, becoming slow and costly. For this reason, we needed a better solution to compare the CAD design with its scanned surface.

Besides tribology, there are other uses for these scanners and software combinations. For example, in closed-loop manufacturing: each step (or some key steps) includes the scanners to evaluate and check if the quality is correct along the production line and detect fabrication errors and malfunction. Another use could be reverse engineering, for example trying to replicate animal leather, creating a prototype design, and analyzing the similarity to the original on a micro-scale.

The main problem in this work will be solving the registration —the process to recognize and align two similar objects—. In this business sector, most solutions are oriented toward macroscopic views. With this kind of scanner, you can have excellent results —one of the state-of-the-art in stereo photometry is around 350.000 points and $5\mu m$ z-resolution[24]— but the common number of points is much lower than our techniques, and the current solutions for the registration are more focused on these macroscopic scanners. For example, the Kinect camera has a resolution of 307.200 pixels but uses infrared light, which limits its depth resolution to 1cm. In our case, a typical scan has more than 500.000 points and a z-resolution sub-micrometric.

To address this inconvenient the registering is done with a random sample consensus (RANSAC), in each iteration, a smart-random process will assign to the source points —the data we want to move— a correspondence to the target mesh —where we want to align—. This method becomes unfeasible with brute force —because the time in some parts is cubic and the sampling combinations rise to a point it's even impossible to calculate—. For these, the only solution is to increase the probability to find a better register, without discarding combinations.

In our case our RANSAC will follow this concept with some adaptation. The first step will be better sampling with an intelligent filter to reduce the size of the object. This filter has to delimit the number of points to a viable quantity, but at the same time, these points have to cover all the mesh as much as possible. Next, we will modify the correlation assignment, the problem here is, that there is no information that could connect the two objects. So we need a similarity index that uses the neighbors' points' information to describe what represents the point and another data structure to know where is this point in the mesh —this data has to be independent of the mesh rotation and position—.

1.1 Goals

The main goals for this project are:

CAD Processing
For how the scanner technology works all the points in the scan are in a XY plane with an equal distance —i.e. scans are height fields defined on a regular grid—. Following our intuition, a good starting point will be having the target point cloud with the same requirements without losing any information that could be helpful.

Coarse and Fine Registration
The common approach to this problem without extra information is starting with an initial matching, and then doing a final alignment with more accuracy. Also, to ease the problem you can make use of shape correspondence, key features, or other filters.

In our case, the problem is the same. But keeping in mind that our scans are bigger, 2.5D —an XY plane with a $f(X, Y) = Z$ as a function—, and may include scanner noise and/or the sample may be worn out. Furthermore, we want to automatize this process, so we need no user input.

Error Calculation

Here we are working with a small scale, so the error has to be precise and we need to know the precision for analytical use.

Visualization

The final result will have a macroscopic alignment but there will be a lot of details with different ranges. Therefore we need tools to keep the encoding and visualization as clear and understandable as possible.

1.2 Sections Organization

In State of the art as its name suggests we will start talking about how the different registering fields have evolved. We will cite some relevant writings here to make it easy for new people starting in this field and some basic concepts. As an opening, we will show Sensofar's scanners and scanning techniques. Then we will comment on how 3D registration is defined, how has evolved, and the main application. Also, we will mention how other software solves this from a user perspective. In this section, we won't talk about CAD processing.

Next in Our Method we will start discussing the tasks needed to reach our goal. Followed by what examples we've used and why. Then the main text in this chapter, where we describe how the main goals have been solved. And finally, some early tests and ideas we have commented on but discarded or failed, and some subjective conclusions.

The final chapter Results presents all the results and their data. Followed by, Conclusions, what we have concluded from it and what are the following steps we identified to improve the time and stability.

2. State of the art

2.1 State of the Art at Sensofar Tech.

Currently Sensofar has many products for different applications, but the two we have used for this project are the S Wide and the S Neox[29]. Also, the S Neox has a second model, the S Neox Five Axis, which has 5 degrees of freedom to move and obtain full 3D scans.

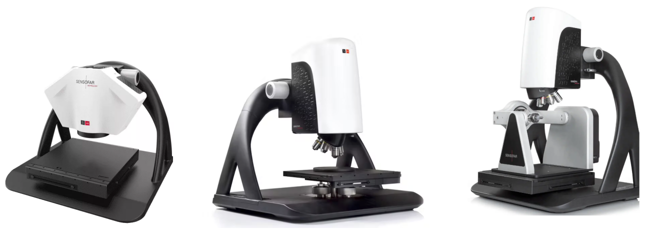


Figure 2.1. S Wide, S Neox, and S Neox 5 axis.

The S Wide is a special case because it only uses one technique called fringe projection[10], where it uses two projectors to project multiple known patterns in two different angles over the sample. Meanwhile, a camera captures each pattern and computes the pattern curvatures to reconstruct the depth information.

On the other hand, the S Neox has more techniques and they have better precision.

Starting with the focus variation[3] and the confocal[1] techniques they share a similar concept. They know the exact height position where the camera is and detect if the pixel is in focus. The difference is in how is detected the focus. While focus variation considers, from all the Z photos, a point in focus as the one with higher contrast respect to its neighbors. The confocal project light patterns to isolate the pixel and the in-focus point is the one with the highest light —because the light beam converges at a single point–.

Finally the interferometry[1], here the optics divide the light beam in two, one goes to the sample, and the other will go to the reference. Then the computer receives the wave phase difference and computes the distance, by determining the number of wave cycles —only knowing the wavelength, Z position, and the wave phase—.

Technique	Objective & Camera	Number of Points	XY resolution	Theoretical Z resolution
Fringe Projection	- & 5MP	$2448 \times 2048px$	$14,23\mu m/pixel$	$500nm$
Focus Variation	EPI 10x v35 & HD	$1224 \times 1024px$	$1,38\mu m/pixel$	$90nm$
Confocal	EPI 50x & 5MP	$2448 \times 2048px$	$0.14\mu m/pixel$	$12nm$
Interferometry	DI 10x & 5MP	$2448 \times 2048px$	$0,69\mu m/pixel$	$0.01nm$

Table 2.1. Characteristics of the scan surface density and resolution depending on the scan technique, settings, and the sample we choose.

2.2 3D Registration and Retrieval

3D registration, it's been a common problem since the classic ICP algorithm[2], published in 1987, and it has evolved in complexity since then. To start we need to differentiate rigid and non-rigid registration. Where in one case we work with only affine transformation, while non-rigid has more degrees of freedom due to it can be deformed. Though this definition has been popularized there are many ad-hoc problems depending on the degrees of freedom like: time, predefined joins or skeleton constraints, isometric or conformal deformations, how the deformation is defined, etc.

Registration is normally defined as a non-linear optimization problem. Given two models (S and T), any extra information (I), and defining an error ($\mathcal{E}$) and a transformation ($\mathcal{T}$). We can describe the registration solution space as $f_{N-DoF}(S,T,I) = \{\mathcal{T}, \mathcal{E}\}$ where the global minimum error is the best registration. For this reason, the two most common methods are optimization methods, and learning-based methods —especially in non-rigid—.

Then we have two approaches intrinsic and extrinsic alignment, depending on the information used for our algorithm. For the extrinsic methods, the knowledge comes from both models in the same Euclidean space. On the other hand, intrinsic methods consider each model to be in a free-standing space and the information used is from internal properties. So in intrinsic methods, the data is transformed to another domain, aligned, and replayed in the original domain.

Note that any extra data can be useful: model global position, color, if the data is defined by the viewpoint, any known geometrical or point references...

One common critical step in most methods is computing the correspondence between the models. Here there is a whole field of shape correspondence. Where in this 2010 decade survey [28] the main approach has been by defining compact map representations —like in functional maps[23]—. Also, learning-based methods have proven to be efficient, and in some cases, they are fully unsupervised[11].

The knowledge evolution has grown rapidly and diversified but here is an overview of the timeline. The registration problem has been studied for at least 40 years. As we can see in the IEEE 2013 survey [30] during this time most approaches consist of adaptations to the sub-problem, defining custom constraints for the correspondence, and trying different optimization strategies.

Meanwhile, the computer vision field published SIFT[21] (an image feature detector), and from this idea in 2010, the SHOT[33] algorithm was presented. Two years before the slow version of FPFH was also published PFH[27]. These two feature descriptors settle a generic solution using histograms. From here there have been developed variants like B-SHOT[25], and also some feature detector comparisons[12].

Recently a new survey on non-rigid registration[9] was published. The number of register methods has grown and diversified significantly. This includes learning-based methods where there is a need for more data but it proved to be more resistant to noise. At the same time, we are seen new scanning tools (reconstructions from monocular videos or real-time video with gyroscopic information) with more noise, but they are able to run on mobile phones.

In the essays we found several real field applications for this problem. One important field is medical image analysis to compare different tests (PET, MRI, etc.), currently, there are numerous software applications to do this [14]. Another use is MOCAP where you may want to apply an avatar to a captured motion and you don't have the bones information[7]. Finally, another common field is simultaneous localization and mapping (SLAM), where they constantly acquire 3D data and map its estimated position from their sensors and scanning[22].

2.3 Commercial Software

To see how is the commercial situation of this registration problem we tested some software. We experimented with the Geomagic X Control and Geomagic Warp, Gom Inspect Pro, and the Optoinspect demo. For each program, we spent about 1/2h to testing and learning how the initial alignment and best fit registration works, though they also include other alignment algorithms that require some geometrical input (curvature, point, plane...). Also, the workstation for this was an Intel 11th Gen i7-11700 with 32GB of Ram and NVIDIA GeForce RTX 3070.

After importing the files, we tested its *Initial Alignment* (which can be configured to fast or precise) and it take less than a minute to get a good initial approach to position and orientation, though it was almost there. Next, we tried the *Best Fit Alignment*, the configuration here is more elaborated and can affect both time and precision, but in general, it seems it is a point filtering with an ICP. This operator deviates the registration to the center of the mesh.

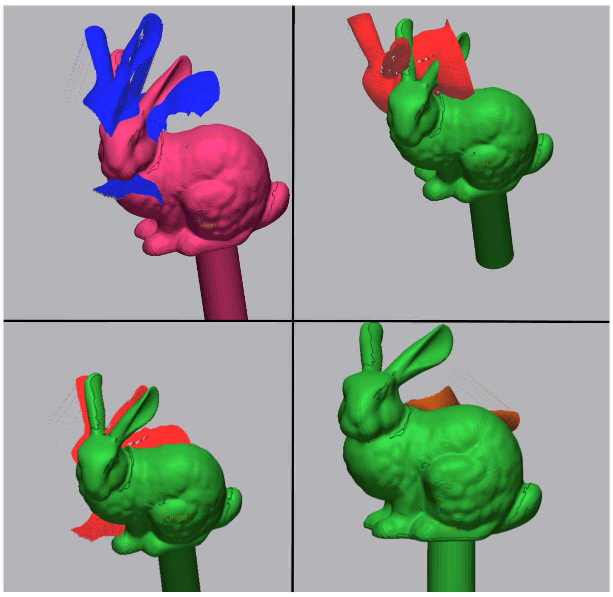


Figure 2.2. Registration using Geomagic Design X. Left: Using Initial Alignment. Right: Best Fit Alignment.

Then the Gom Inspect 2016 includes an initial alignment called pre-alignment algorithm than requires no geometrical input but it couldn't compute our input. All the other options require selecting some geometry input, so we ended the test here.

Lastly the Optoinspect3D Inline Demo has only one algorithm and its parameters are similar to the ones for an ICP and the behavior as expected from an ICP. In the end, register was not possible.

In conclusion most software focuses on other algorithms using geometric references values. Although it is quite interesting and remarkable how the Geomagic has a really fast and almost correct initial alignment —which may have a different strategy than ours—.

3. Our Method

3.1 Overview

Task

The main goal is to create a tool to compare a high-precision scan surface with its original design. To do so, our first step will be to transform both inputs into point clouds with the same properties so we can do operations on both of them. The next step is to find the 6-DOF (3 Degrees Of Freedom for translation and 3 for rotation alignment), at least a good initial guess, that will ensure the preconditions for the following operators and avoid local-solutions that in fact are false-positives. Finally, we have to compute the precision and represent it in an intuitive visual interface.

CAD processing

CAD (Computer-Aided-Design) representation has many file extensions. In our case, we will work with STEP files, a data exchange format from the ISO 10303-21[13]. This format supports both tessellated and boundary representations. In this form, we can have one or multiple shapes. And each shape is a set of faces[32] (or features) and this face describes a surface —i.e.: p-curve, plane, b-spline, etc., also the face can include boolean modifiers to remove or include more surface parts—.

To transform this into a 3D triangle model, we need to tessellate these faces. And next, reconstruct the mesh topology between faces —without losing what point belongs to each face—. In order to do that, we have to store the points describing the border and the interior of the face in multiple sets, where the border can be repeated.

Coarse and Fine Registration

Here there are multiple problems. As mentioned previously, we need to reduce the number of points. This can be approached by filtering the model but this can be approached with different ideas like: randomly, averaging, or ranking the points.

Next step will be to study how these three filters can be combined to give the best result.

In addition there is another limitation, our scan is a 2.5D mesh and may or may not represent the whole CAD model —represent only a sub-model—, increasing the number of outliers. For this, the data used to assign correspondences have to consider not only the point itself and its neighborhood's local information but also where this point is with respect to the whole mesh neighborhood. To do so, we can tweak a little bit how the correspondence between points is done, and add this mesh space information.

Finally for the fine registration, the task will start with the ICP. If it's not working well enough we will use the previous filters to optimize the time and result.

Error Calculation

For the alignment error, the ICP can provide a global error value. But for each point, we need an accurate point-to-cloud distance calculator, because we have a lot of resolution in the scan, this can be approximated with a point-to-disk distance.

Visualization

Here the work will consist in encoding the point error and representing it using a color map. Furthermore, to help visualize the differences we tried some transparency policies, and last, a sectional cut to focus on the zone of interest.

3.2 Test Dataset

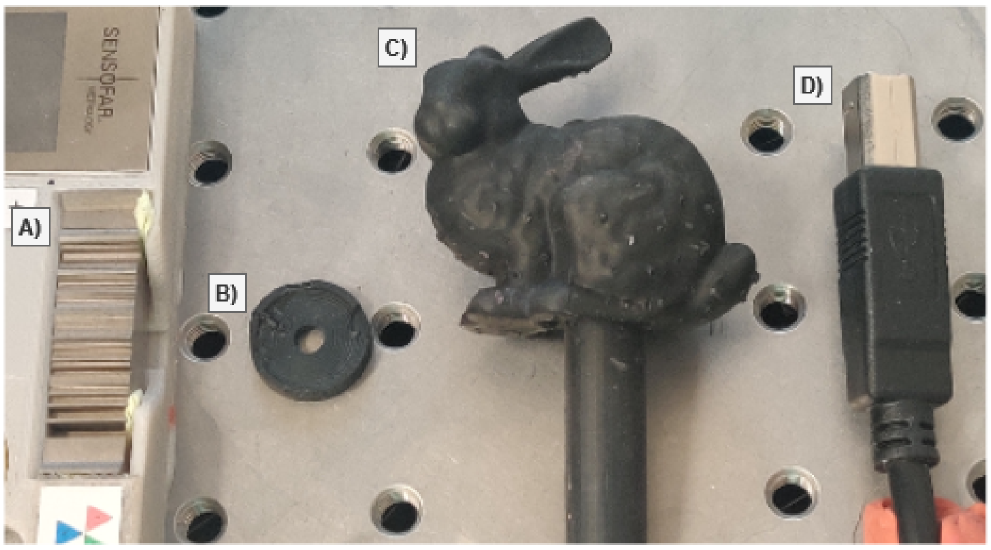


Figure 3.1. A) Calibration B) Diaphragm C) Stanford Bunny D) USB-B.

As we will see our algorithm is sensitive to points that don't have correspondents. This is a major inconvenience since forces us to have an accurate CAD and its scan.

To evaluate the software we have four samples, each one more complex than the previous:

Calibration The first one is a set of steps used in the S Neox scanner to validate its accuracy. The scan was done using focus variation and has 5.706.848 points while its CAD design is composed of 91 faces. This is by far the biggest and most accurate sample where all steps are unique.

Diaphragm Next is the diaphragm. Here we have the original design model and a physically damaged scan. This has 1.005.886 points with 124 features. This set of circumscribed circles with two notches will test how well it works with almost symmetric objects and some small noise.

Bunny Next we have the Stanford bunny, which was 3D printed —from the CAD version— so we could have an accurate scan. The input is 2.454.548 points and 700 features. Here the differences are from the printer error. The bunny has a similar curvature between the ears, crown, and body, this creates many local-solutions that in fact are false-positives.

USB The last one is a USB-B and its 3D model from the internet. It's composed of 1.557175 points and 2.232 features. The 3D model was originally a high-detail triangle mesh, but in order to work, we simplified the details and convert it to STEP file. Though the input is a simple geometric mesh, the details won't match since they are completely different.

3.3 CAD processing

As we have commented in the task introduction, we will start by remeshing the surfaces. For this we will read each CAD faces using UV coordinates surface[31]. While tessellating, we will push the points in order — forming a $|V| \times |U|$ matrix—. The tessellation will be proportional to the scanner XY resolution leading to a theoretical regular distance between points that we will call theoretical resolution length (RL_T). The final vector will be a concatenation of matrices of different sizes. At the same time, we store the number of points for U and V dimensions, so we can retrieve where the face starts and ends, which points are interior, border, or edges. In some cases the design is done using boolean modifiers —if there are holes or complex figures— that can be solved using a point-to-plane check — with tolerance— and replacing the point with padding in the matrix (e.g. using nan values).

Now the points are in a sorted matrix. We can reconstruct the inner triangles trivially and mark the outer face points, as edges and corners. The next will be to solve three possible problems. For this we took inspiration from the common mending sewn problems:

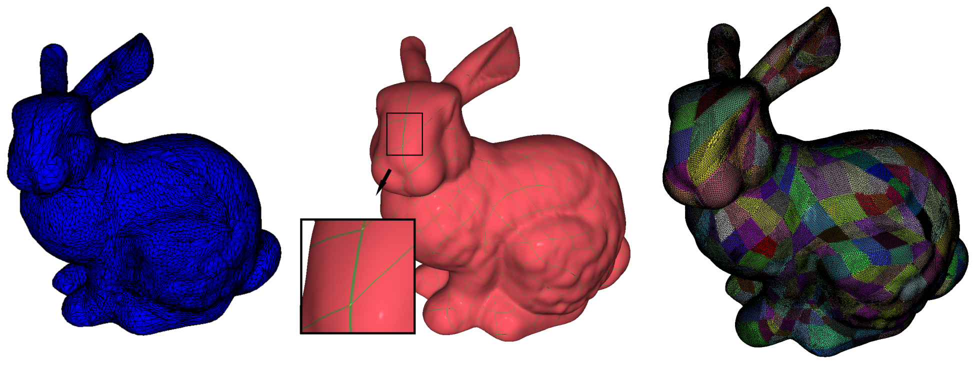


Figure 3.2. Left: Common CAD processing. Center: UV tessellation without sewn, Red: interior face; green: Edge; Yellow: Corner. Right: Tessellated mesh preserving features ID.

Pin holes There are many points very close and they may be unconnected. This can be detected with a k-nn (k-nearest neighbors), the input will be all the points and a portion of the theoretical resolution length as a radius (in our experience 0.5% is good enough). Once detected, choose one to keep in the mesh, delete all the others, and update the classifier list —interior, border, edge—.

Tears There are two faces sharing one edge, because there is a constant resolution, they can be joined in a one-to-one association using k-nn. The input will be the edge points and k equal to one.

Unstitched Edge This is like the Tears problem but the edges may have a one-to-many relation. Now that the set of points —borders and edges— is smaller (approximately 3 times the number of corners) the k-nn can work faster so we can increase the radius to the whole length resolution.

This process has two drawbacks. First, you have to re-mesh the whole object at the end of the “sewing” to update the triangle list. And second, the length resolution has to be a multiple of the UV feature length.

The first problem sets a time and memory limit that in some cases may affect our inputs since at a critical point we duplicate the memory and we have a complexity of $\mathcal{O}(n \log n)$, creating a bottleneck.

And the second problem can be adapted to our situation. The artifacts increase when the length resolution is big and the mesh has many sharp curves —we are losing information—, luckily this isn’t our case since we use a small value. Furthermore, if the length is a bad multiple from the U or V original face size, some edges points will not be the same as the original, causing problems when we apply the Tears mending —merging to the interior producing holes—. This can be patched by forcing that U and V always include the last point. By doing this, the problem is reduced.

For our case, this can be accepted, as our objective is to obtain a point cloud (with a regular distance, and good points and normals).

3.4 Coarse and Fine Registration

The process to fully align the two inputs will start with a rough matching (coarse registration), for this, we will modify the IA RanSac, an intrinsic method, to work better with our inputs. And for the accurate alignment (fine registration) we will use the ICP[2].

Our work on this section consists of two parts. One is to optimize the source and target points cloud to maximize the fitting by reducing or increasing the number of points, and how it’s done. The second part consists of adapting the IA RanSac to work better with the 2.5D scan, and/or with a limited scan part.

Since this work is highly related to IA RanSac we will do an overview of how it works. This algorithm is a random iterative process. In each iteration, a kernel calculates a fitness score, and at the end, it returns the best solution —there is no connection between iterations, it’s a random brute-force approach—. The kernel has four steps:

1. The original algorithm samples a random number of points, defined by the programmer as *samples*. At the same time, these samples are constrained by a *minimum distance* they have to have between them, this variable is defined by the programmer. Currently in the PCL library this is done with an asymptotic time of $\mathcal{O}(samples * |points|)$[16].

2. The kernel searches for the correspondence between these source samples and the target point cloud. This is done using k nearest neighbors (k-nn) over a histogram that represents the features in the point cloud. This histogram is computed using a Point Feature Histograms (PFH or Fast-PFH alias FPFH)[27]. In the correspondence, to avoid a false-positives, instead of assigning the nearest feature neighbor, the programmer sets a quantity of *correspondences*, and the algorithm chooses a random correspondence in the top *correspondences* candidates.

   - The key concept in this step is the FPFH. This feature evaluator has the property that it's not subject to the mesh rotation or position. The process to compute the features is: Compute a k-nn for each point (the programmer will set a *k* or *radius* for it). Now for each set using the original point as a center, compute a Darboux frame, and finally compute in which chunk resides each neighbor point respecting the frame planes. The percentage distribution will be the final histogram.[15]

3. An estimated rigid transformation will calculate the matrix transform. Now that we have the source samples and the target correspondences, we can estimate the affine transformation using an SVD estimator (which has a $\mathcal{O}(samples^3)$ asymptotic time).

   - The SVD estimator consists of: computing the centroid corrected version of the point sets (centroid source = $\hat{S}$, centroid target $\hat{T}$). Prepare the covariance matrix $S = \hat{S} * \hat{T}^T$ and extract the SVD $S = U \sum V^T$ to obtain the optimal rotation matrix $R = VU^T$. Finally you can construct the transformation matrix computing the translation with $t = \hat{S} - R\hat{T}$. [17]

4. Once the source cloud is transformed, we can evaluate how accurate it is with a fitness score. One option for this heuristic is the summation of all the point-to-point distances. Because the programmer may have a tolerance value, *maximum distance*, it could be used to normalize the distance and clamp the value between zero and one —if it's bigger than the maximum—.

3.4.1 Part-to-Model

After studying how IA RanSac works we see that its major flaw is that it's able to understand how the points are distributed (edges, corners, concave, convex, flat surface, etc... using the FPFHs the k-nn settings in a precise way), but this local information is the only concept being used for the correspondence.

The problem with this is that the algorithm is ignoring the global information between features. And it may attempt to connect two pairs of features that are correct in the FPFH but looking at the whole mesh it's impossible —e.g. imagine that the source and the target both have two curves very similar. But if you connect them, in the target the distance between them is 3 units, and in source 10, it's impossible to connect in a 6-DOF, but the current correlation policy will consider this a target candidate—.

A solution that Chu-Song Chen proposes at [5] is a constraint on the target correspondence to be inside a radius described from the sample points information.

Following this idea combined with the IA RanSac, the next step is to apply this between-features information as a weight in the correspondence assignment (instead of a binary filter). To do so, we try to match the distance in the source samples with the distance in the target samples.

Given N source samples ($SS = \{0, ..., N\}$) over M target points (TP), we can get the top K correspondences and sort them. The classic IA RanSac correspondence will assign for each source point K target candidates (TC_i).

$$\forall s \in SS, TC_s = \{TP[f_c(s, 0)], ..., TP[f_c(s, k)]\} \tag{3.1}$$

Now the classic algorithm will select the final correspondence from the target candidates randomly, forming a best target correspondence (BTC). Following what we have mentioned, a better option will be to minimize the distance difference between source points and target candidates.

$$\begin{aligned} \text{classic: } & BTC = \{rand(TC_0), ..., rand(TC_N)\} \\ \text{ours: } & BTC = \{\forall s \in SS | \arg\min_{c \in TC_s}(\sum_{n=0}^{N-1} |s - SS[n]| - |c - BTC[n]|)\} \end{aligned} \tag{3.2}$$

This theoretical solution differs from the random approach in the IA RanSac, and can't be applied directly. Because it's a recursive definition and it lacks a base case or we may compute all the combinations and that will be too expensive. To solve this, BTC_0 will be selected randomly. Now the definition is complete, but a deterministic solution is subject to local-solutions, so we can form a final correspondence selector with a weighted random using the distance.

$$\begin{aligned} \text{final: } & BTC = \{rand(TC_0, [1, ..., 1]), ...rand(TC_N, W_N)\} \\ & D_N = \forall c \in TC_s | \sum_{n=0}^{s-1} |SS[s] - SS[n]| - |c - BTC[n]| \\ & W_N = \{\forall c \in TC_s | max(D_N) - D_N[c]\} \end{aligned} \tag{3.3}$$

This will make that the points with the smallest difference will be the most probable to be selected.

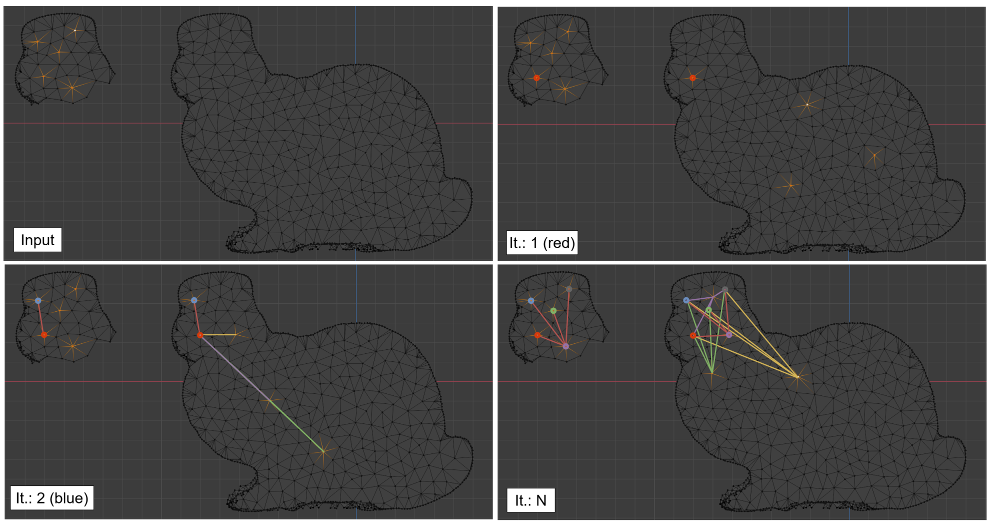


Figure 3.3. Input: On the left is the source with the random samples and on the right is the target.
Iteration: 1, On the left with a yellow circle, the sample is selected for the target correspondence, and on the right, the random target is selected (for these examples the random is always the correct one).
Iteration: 2, sample selected with its reference distance (left) and correspondences with its distance to the previously selected correspondence.
Iteration: N, the last iteration testing all possible target correspondence distances against the source.

3.4.2 Sampling methods

Along the project we needed filtering operators. For that reason, we developed a set of filters with different kernel ideas. These filters are developed for general input, but some are more focused on one input than another or may have a mandatory prerequisite.

Poisson Disk Sampling

Since we have the points dispersed in a 2D plane, we can adapt the Poisson disk sampling radius (r) parameter to:

$$r \approx \frac{\sqrt{|points|}}{\sqrt{k}} RL_e \qquad (3.4)$$

Where k is the number of Poisson points and RL_e is the empirical resolution length calculated from averaging the distance from all the points to the nearest neighbor. Although this is designed for 2D and 2.5D inputs, this filter can also work with 3D input if it has a high point density and $k << |points|$.

FPFH Unique points

In the second paper of FPFH[26] they mention an approach to detect unique points. This is an overview of how they did it. First, the FPFH histogram[1] will be interpreted as points in N dimensions, to compute the euclidean distance. They consider that the histogram of these distances should follow a Gauss distribution. From this, the distribution tails could be considered unique —In their paper, they consider that the top less common 5% are the unique points—. These points may be truly unique or just outliers, for this reason, they do a double-check step. What they propose is to calculate two FPFH unique point sets with incremental k-nn (P_{fi} and P_{fi+1}) and validate the points with their intersection. Then if we want a bigger or smaller set of points we compute n multiple FPFH sets and join them as:

$$P_f = \bigcap_{i=0}^{n-1} [P_{fi} \cup P_{fi+1}] \tag{3.5}$$

For our case we know we have a regular distance between points. Knowing this prerequisite, the radius we used to calculate the different sets is $r = i * RL_T$.

Voxel Grid

For this, we simplify the points with a Voxel Grid algorithm from the PCL library[19]. Here the points are hashed to a 3D grid where a parameter ($size$) defines each voxel size. Then the final point is the centroid of all the points in each grid. At the same time, the normal is recovered for each new point. This new normal results from averaging the old normals recovered using the new point and the voxel size.

$$\begin{aligned} P' &= \{\forall V \in Voxel(P, size) | \frac{\sum\limits_{p \in V} p}{|V|}\} \\ N' &= \{\forall p' \in P' | \frac{\sum\limits_{n \in k-nn(p',P,size)} n}{|k-nn(p', P, size)|}\} \end{aligned} \tag{3.6}$$

[1]The same histogram we talk about in the second step from Coarse and Fine Registration

3.4.3 Final Implementation

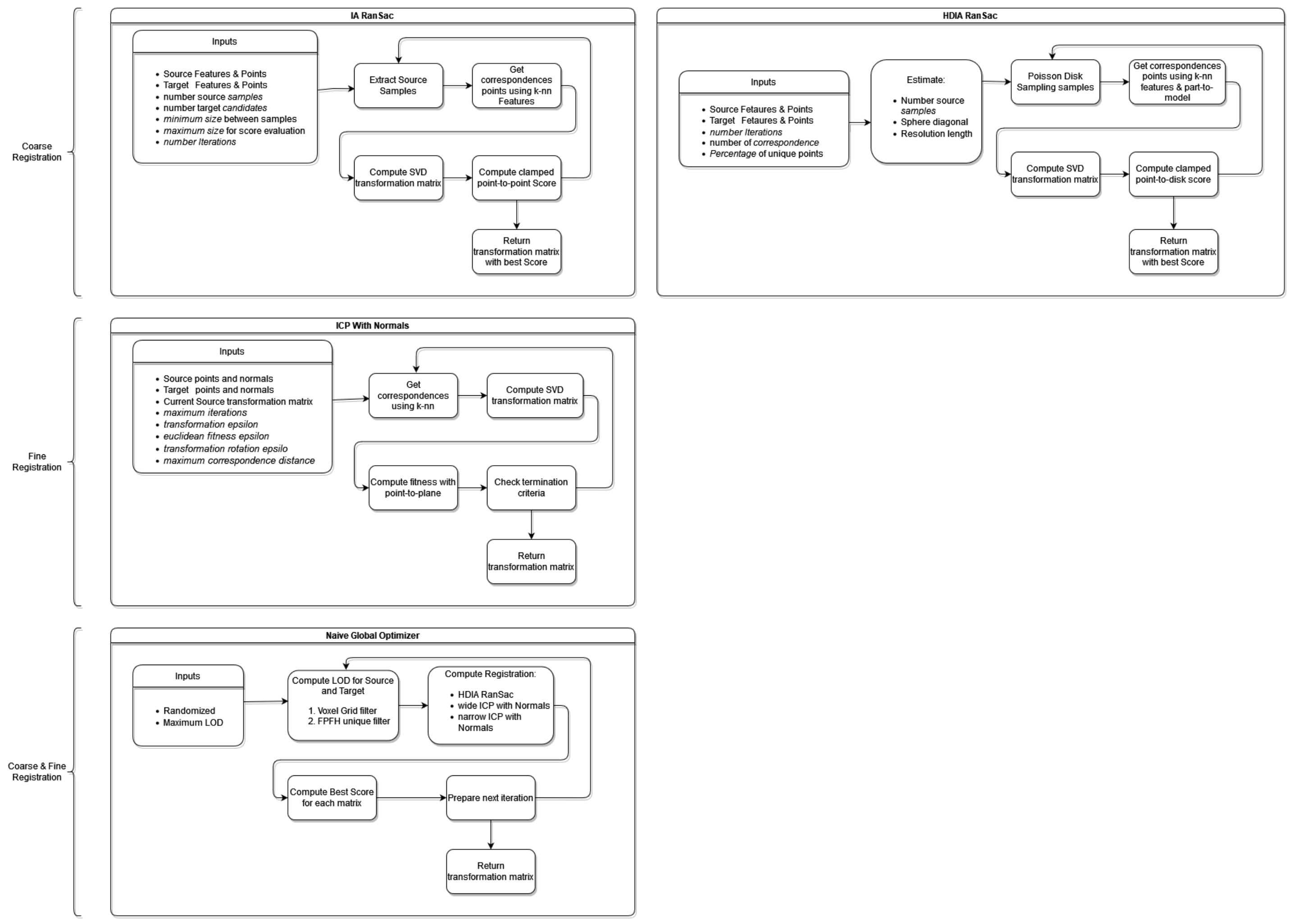

Coarse Registration
IA RanSac
Inputs
Source Features & Points
Target Features & Points
number source samples
number target candidates
minimum size between samples
maximum size for score evaluation
number Iterations
Extract Source Samples
Get correspondences points using k-nn Features
Compute SVD transformation matrix
Compute clamped point-to-point Score
Return transformation matrix with best Score
HDIA RanSac
Inputs
Source Fetaures & Points
Target Fetaures & Points
number Iterations
number of correspondence
Percentage of unique points
Estimate:
Number source samples
Sphere diagonal
Resolution length
Poisson Disk Sampling samples
Get correspondences points using k-nn features & part-to-model
Compute SVD transformation matrix
Compute clamped point-to-disk score
Return transformation matrix with best Score
Fine Registration
ICP With Normals
Inputs
Source points and normals
Target points and normals
Current Source transformation matrix
maximum iterations
transformation epsilon
euclidean fitness epsilon
transformation rotation epsilo
maximum correspondence distance
Get correspondences using k-nn
Compute SVD transformation matrix
Compute fitness with point-to-plane
Check termination criteria
Return transformation matrix
Coarse & Fine Registration
Naive Global Optimizer
Inputs
Randomized
Maximum LOD
Compute LOD for Source and Target
1. Voxel Grid filter
2. FPFH unique filter
Compute Registration:
HDIA RanSac
wide ICP with Normals
narrow ICP with Normals
Compute Best Score for each matrix
Prepare next iteration
Return transformation matrix

Figure 3.4. Flowchart of the different algorithms we will see and their input variables.

HDIA RanSac

To simplify the IA RanSac algorithm we've automated the user-defined input variables (*samples*, *minimum distance* for the sampling step, *correspondence* in the smart-random correspondence and *maximum distance* for the fitness score).

Starting with the source samples selector, we are replacing the brute force —double for loop with the variable *minimum distance*— check with a Poisson Disk Sampling. This way we no longer need the parameter and change the asymptotic time to $\mathcal{O}(k)$. At the same point, the number of *samples* is defined using the smallest set of unique points

$$samples = clamp(min_{i=0^{n-1}}(|P_{fi} \cup P_{fi+1}|) * percentage, 3, maximum) \quad (3.7)$$

This heuristic is based on the idea that the FPFH Unique points filter is the union of the set of unique points, which means each set is composed of the minimum points to define the model. Following this concept, if we want the first registration then we should include a minimum number of these particular points while keeping a good balance between outliers and interesting points. For that reason, we tested a *percentage*, and empirically this number should be around $(5\%, 25\%)$. The clamp is to ensure we have the minimum requisites for the SVD (3 points). And the *maximum* is for hardware and complexity limitations, we don't want the iterations to become a bottleneck.

Next parameter we configured is the *correspondence* in the correspondence step for our custom algorithm. The classic algorithm will work with an arbitrary ratio of 1/100 (correspondence candidates/source points), but with our part-to-model, a 5/100 ratio works better.

And finally, the fitness score is a custom fitness using point-to-disk distance, the *maximum distance* will be replaced to $1/8$ of the smallest bounding sphere diameter (source S or target T) and the value will be normalized with the number of points. This way our score will be a normalized value for all inputs making it easy to read without changing its continuity and with higher precision than the default point-to-point distance. To help to understand this $Score$ we also included the percentage of points inside this threshold as $QttyScore$ —this variable has no use except for better comprehension—.

$$\text{Score} = \frac{\sum_{p\in S} \min(\frac{|DistanceToDisk(p,T)|}{(D/8)}, 1)}{|P|} \tag{3.8}$$

$$\text{Qtty Score} = \frac{|\{p \in S | (D/8) > |DistanceToDisk(p,T)|\}|}{|P|} \tag{3.9}$$

Where T and S are the target and source clouds.

For our hardware limitations, even though we optimized the sampling process and limited the SVD size the k-nn search may become a bottleneck. To avoid this we developed a pre-process where we do a Poisson Disk Sampling filter to reduce the CAD and scan. This filtering is the maximum that our hardware can handle, 40.000 points for the target and 1.650 points for the source.

ICP

Since the coarse registration should lead to a close solution and we don't know for sure if we can have a perfect fit for each point, the ICP with normals is our best option —The major problem from this algorithm is that the correspondence points is selected from the shortest point-to-plane distance. This may provoke that depending on the starting point, the best solution is an edge with a normal that ends in a non-existing plane. But at the same time, in the case that the best position for a point may be in between two points it's possible, and the fitness error is lower than standard ICP—.

The other configuration we have done is the parameters for the termination criteria. In our case we want the best accurate transformation matrix, for that, we want a decent amount of *maximum iterations*. And the epsilons —differences between two iterations– will be small values, this way the criteria will stop with the smallest precision. For example, the *transformation epsilon* will be $1\%RL_e$, for the *transformation rotation epsilon* an angle of $1\circ$, and the other epsilons will be an arbitrary number like 1^{-10}.

Also, the parameter *max correspondence distance* will help to avoid the sliding problem and we already know that this distance at least is at $1/8$ of the bounding sphere diameter and for a more limited version we can use a scalar of the resolution length.

Naive Global Optimizer

Now using the simplification, we've developed a naive global optimizer. The idea is to try different simplifications —a similar concept to LOD (Level Of Details)— and get the best results for the registration. The concept behind this is that a global solution will be global for all LODs (where the cloud is big enough to still be considered a valid model). At the same time for some holistic reason, each LOD will have different probabilities for all possible solutions (global and local). So we can work with smaller samples to save time and achieve a better solution than the original inputs.

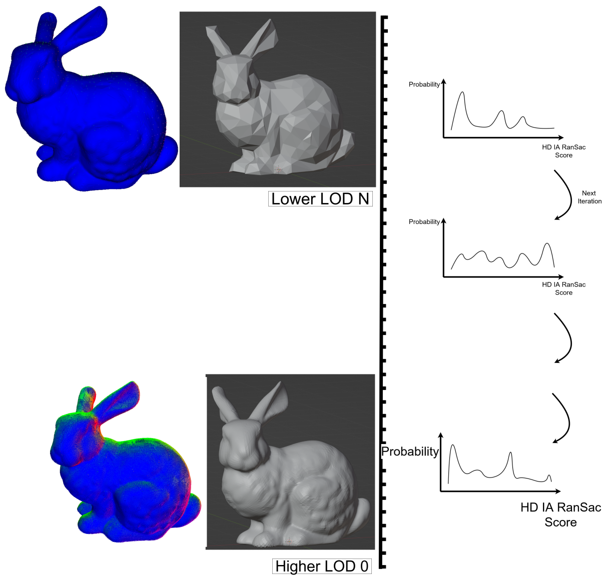


Figure 3.5. Visualization on how the input optimizer iterates through the selected LODs to obtain the best coarse score in each step and improve the fine registration with a better initial position.

Each LOD will consist in a Voxel Grid simplification followed by the FPFH Unique points unique points extractor. Then it tries to run the registration algorithms separately and stores the best result. Because we don't want to execute all LODs to save time, we will start from the smallest point cloud, and execute only some iterations increasing the cloud size.

Here we can develop many policies depending and the registration algorithms we have and how we combine them. We currently have three registration algorithms: HDIA RanSac, ICP with the RanSac threshold as the *max correspondence distance*, and the ICP with an even smaller *max correspondence distance* using the resolution length. Our combination approach will be a naive brute force. We will try the three algorithms, and since the fine alignment depends on the current position, we will execute the loop two times using the best know transformation matrix at the moment.

To select the LODs, we have two options, random and ranged based. In the random, we get a random value for the Voxel Grid length in each iteration. While the range will have a pre-process. Here the user will set the maximum resolution length to calculate the LODs and a process will iterate increasing the resolution until it finds a minimum, defining the minimum LOD as the one where the HDIA RanSac won't need its random filter to pre-process.

Also considering that the difference between two consecutive LODs may not be significant, to test as much possible LODs, the stride length between iterations will increase/decrease randomly. As mentioned previously we don't need to execute all the range, so we will terminate the input optimization after some iterations and if it isn't good enough the user will resume it.

3.5 Error Calculation

At this point we will consider that the registration has found the best solution and we will proceed to obtain the errors for the error analysis.

Computing The Error

For the error visualization we first need to obtain the values. This error will be a point-to-disk distance. It consists of two steps: for each point, p, get the closest point q with k-nn. Then using the q normal (n) compute the point-to-disk distance. Where we basically first project the point to the plane (p'), and next if it's outside the disk, we move q in the direction of p' projecting it to the disk boundary (p''). Also using the normal we can know on which side of the plane is p to determine the sign.

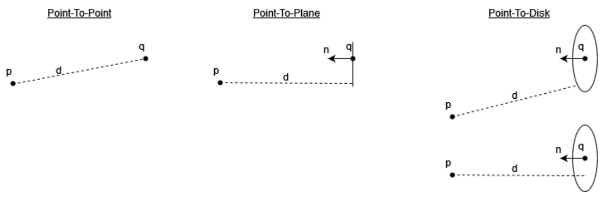


Figure 3.6. Visualization of point-to-point, point-to-plane, and point-to-disk distances.

$$\begin{aligned} p' &= p - n * (n \cdot (p - q)) \\ p'' &= \begin{cases} q + \dfrac{(p' - q)}{|p' - q|} * RL & \text{, if } |p' - q| > RL \\ p' & \text{, otherwise} \end{cases} \\ error &= \begin{cases} |p'' - o| & \text{, if } n \cdot (p - q) > 0 \\ -|p'' - o| & \text{, otherwise} \end{cases} \end{aligned} \tag{3.10}$$

Note that this method relays on having good normals and that the point is close enough to its equivalent in the other mesh. This is because the scanner may add a small noise in the cloud, and any small change can affect the normals significantly. This can be palliated by averaging the normals in some neighborhoods.

Precision

Now for a more analytical approach, we need to define a precision value so our tool is complete. We will define the precision using the RMSE using the original inputs.

$$Precision = \sqrt{\frac{\sum_{p \in Scan} DistanceToCAD(p, CAD)^2}{|Scan|}} \tag{3.11}$$

Note that the point-to-CAD distance algorithm is highly demanding and not feasible for the number of points we are working with. For that reason, we will do a fast random sampling from the scan. Because the points are in a XY matrix, we can define and decide to sample N points and compute a stride $s = \frac{|Scan|}{N}$, then the sample index will be the index by the step plus a random value from one to the stride ($index = iteration * stride + rand(1, stride)$).

3.6 Visualization

For the error visualization we start with a classic Phong's shading where the point error is already known and received as an input in the vertex shader.

Colormap Encoding

The encoding works with a $[-1, 1]$ range and can be set to the classic jet colormap or to a diverging color blind friendly colormaps (like PRGn, RdYlBu or PiYG)[8].

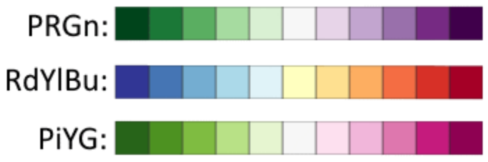


Figure 3.7. Divergent colormaps.

For the jet colormap the range is linear so we transform the range from zero to one where zero will be the negative minimum value and one the maximum. On the other side, the divergent mapping will be asymmetric starting with minus one as the smallest negative value, zero with no changes, and one as the maximum positive.

Another key point is that the user can configure the minimum, and maximum value to change the scale and increase the resolution for smaller values.

clipping Plane

The scans can have many details and concave zones, so a tool to discard the rendering in some parts can help the visualization. For this, I've just added a clipping plane check in the fragment shader that discards the outside fragments.

Transparency

Showing the two objects in the same position may be difficult to visually understand. To solve this problem, we have many options:

- Disable the rendering for one of them.
- Render the model with uniform color.
- Render the model with transparency.

For the transparency policy we propose two options, one is a low but constant alpha with the same color as the uniform option. And the other is defining alpha as equal to the error squared clamped to a minimum and a maximum value –avoiding full transparency or opacity—.

Interpolation

Though the data is calculated per-vertex the error could be sent to the fragment shader where the value will be interpolated linearly. Most users are experts and understand that the scan is a point cloud (represented as a mesh), so although the colors look sharper this interpolation may be interesting to disable (using the GLSL flat keyword).

3.7 Further Considerations

Even though the main concept for this project is an approach to 6-DOF registration with a high-density cloud, throughout the project with the help of my supervisor, tutors, and other contributors we found some attractive ideas that may be interesting to study, take into consideration or we couldn't explore them enough with the minimum rigor:

- A first thought was to simulate the real scanning camera and scan the whole CAD like the real instrument. The camera will follow a Gaussian map from the model (figure 3.8) this way the rendering will be perpendicular to the scanning zone simulating the scanning process. And next, this will be registered using the result as a 2D image. Though the prototype showed positive results, it was probably that it had bad scalability. Also, a major drawback was how occlusions were a double edge that may help or may make the problem unsolvable. On one hand, if the scanner was positioned at the same position as our simulation, we will have the same occlusions view. But if for some reason the scanner wasn't positioned perpendicular to the sample (with some angle) the solution will never be found. For that reason, this was discarded.

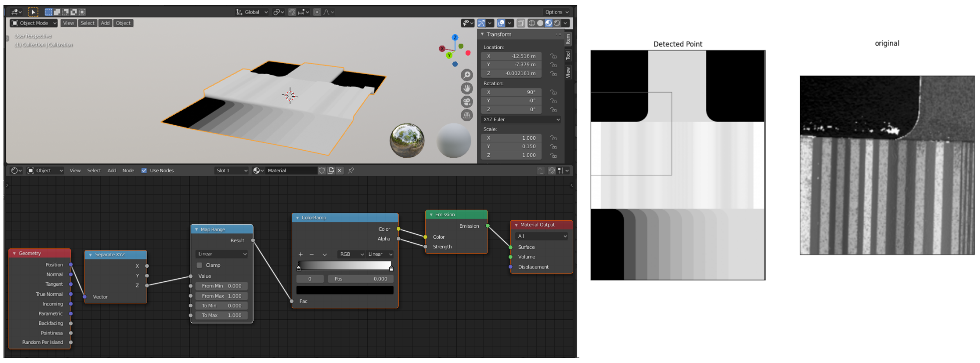


Figure 3.8. A Blender node shader to simulate the scanner behavior, and its result using OpenCV.

- One interesting idea for a simplification method was using spectral spaces. Following chapter four (Spectral transform) from [6]. We could do a Fourier-transform-removing to change the space, remove the lower frequencies, and change back again. The idea behind this is that the CAD has less texture and detail (only the basic high frequencies) than the scan, meanwhile, the scan is formed from a lot of small detail —an scan may be $3.5cm$ and the points are $14.5\mu m$ a part—, so basically it has a lot of high frequencies that can be removed (another good point is that in fact is a 2.5D mesh, so it could be interpreted as a 2D wave). The only inconvenience in this is its major complexity to develop.

- For the registering we knew that the PCL library[18] would already have a starting point. But they have many options (Trajkovic Keypoint 3D, ISS Keypoint 3D, etc...). Ultimately we found the IA RanSac[26] and thought its first solution was poorly with raw input, its parameter was intuitive enough and maybe the solution could be found after many executions with different parameters.
After a lot of execution time, we couldn't find any solution, even though we were changing the parameters, the score was stuck in a big area and all of them were local-solutions. This was counter-intuitive because the IA RanSac was working fine with simple 3D models. That's why we started to work on reducing the cloud size and optimizing for matching a model with only a sub-part (part-to-model).

- Another interesting property in the IA RanSac was that the matrix estimation could be configured. The PCL already includes some options, SVD (Singular Value Decomposition), LM (Levenberg Marquardt-based), DQ (Dual Quaternions), Transformation Estimation with 3 Points...
After doing some research we considered keeping using the SVD. Because it was a numerical method (LM is iterative), we won't worry about estimated eigenvalues, it's simple (DQ offers a matrix that can be interpolated), and the input can be any number of points. A. Lorusso did a comparison of different matrix estimators and conclude that the most

stable and general option is SVD[20].

- One big inconvenient we found is that the error in the CAD-Scan has to be simple (wear not too destructive, only a small missing zone, no infinite ground, etc.) but if the two models have some parts that are completely wrong, they can't be registered. For these, we've added a section in the results chapter Sensitivity And Robustness.

- At some moment, we exported an early version of our IA RanSac (HD IA RanSac) algorithm to another system where it works with any 3D model. There we tested the Part-to-Model optimization with common 3D models and it showed better results than the classic algorithm. Though this optimization was especially thought for big clouds where the global-solution is more occluded this kind of optimization seems interesting and may need more exploration.

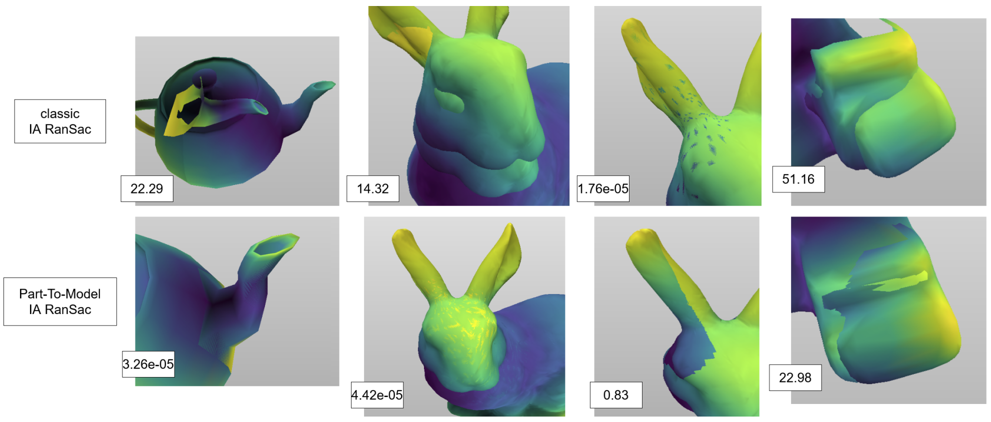


Figure 3.9. Captures and final error from registering only using only 1000 iterations. The only difference in the Part-To-Model is applying the optimization.

4. Results

4.1 CAD processing

The two configuration variables for CAD processing are the theoretical resolution length and the plane check tolerance —if it's necessary—. One important note is that the code has two parts that can be concurrent[1]: the CAD feature tessellation; and the update on the triangle list on remeshing the deleted points.

Input	Features	Open Cascade CAD Points [2]	Open Cascade CPU Time	Resolution[3]	Our Approach CAD Points	Our Approach CAD Time[1]
Calibration	91	3.594	0,033s	118,896	6.905	17,547s | 10,435s
”	”	”	”	371,716	573	2,218s | 0,376s
Diaphragm[4]	124	98.376	0,414s	92,147	44.159	101,995s | 33,358s
”	”	”	”	386,851	468	1,639s | 0,285s
Bunny	700	207.876	0,944s	168,262	124.950	49,107s | 11,412s
”	”	”	”	595,394	6.740	2,226s | 0,377s
USB	2232	6.696	0,252s	142,707	679,49	24,604s | 24,605s
”	”	”	”	679,49	166	2,073s | 2,073s

Table 4.1. Times from Open Cascade triangulation[4] and our tessellated CAD with different resolutions. Averaging 3 executions.

As we can see in the table 4.1, our method is slower even when we create fewer points than the Open Cascade. At the same time if we need to do a point-to-plane check the time increases a lot and may break the regular distance property. Another important concept we can see is

[1]Time is expressed as (CPU time | execution time)

[3]Open Cascade process the cad with split edges.

[4]This is the empirical resolution length (RL_e). Reference: SensoSCAN SWide: 14.16 and SensoSCAN SNeox: 2.76

how the number of features is a bottleneck, the USB design is composed of triangles and this is counterproductive in the processing.

4.2 Registration

For this section we will see the time (execution, CPU, and real time) and its accuracy for each method we develop. Also, we tested how the noise and the cleaning process for the same input with different criteria can affect the registration process. Finally, we show how an experienced user can reduce the execution time using some operators.

While the *fitness* is quite easy to understand as a precision value, one way of interpreting the *score* and *qtty score* is using the *qtty score* as a percentage of points in the threshold ($D/8$) range and the *score* as a mean divided by the threshold —considering the penalization in the *qtty score*—.

Although these variables can be used to evaluate the registration each input is different and may have some local-solutions at different points. Here are some examples to visualize these values.

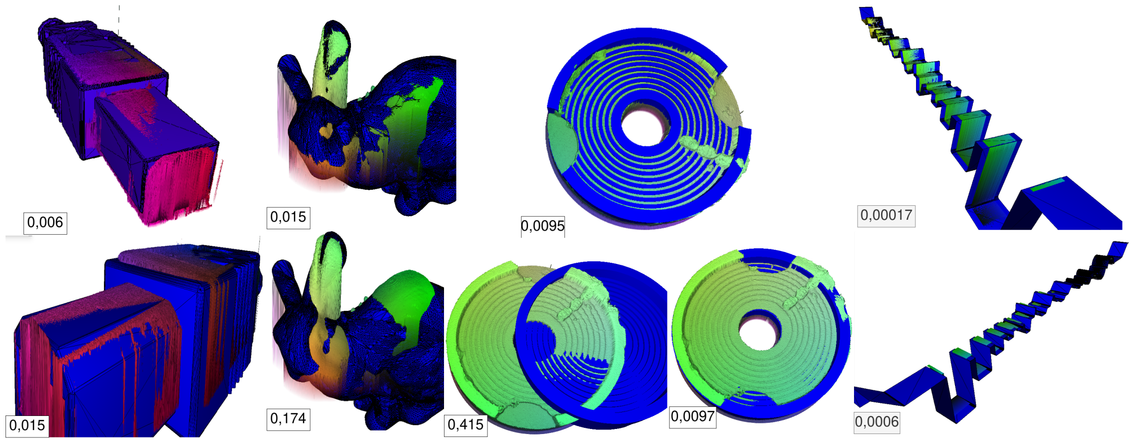


Figure 4.1. Upper row: true-positive solution. Lower row: false-positive of some local-solutions. The value in each picture is the *score*.

4.2.1 Sensitivity And Robustness

Cleaning Process

Normally, after the scanning the user cleans the data using the SensoView software, this cleaning process depends on the next usage, purpose, and the scanning quality (despike, background removal, restoring missing points...), so it isn't always the same.

For this reason we tested how different approaches may end up. The first one will be a rough cleaning, only removing the background (diaphragm raw), next we will reconstruct the mesh and create a moderated recreation (diaphragm), and finally, a strict cleaning, ensuring all points are real (diaphragm flat).

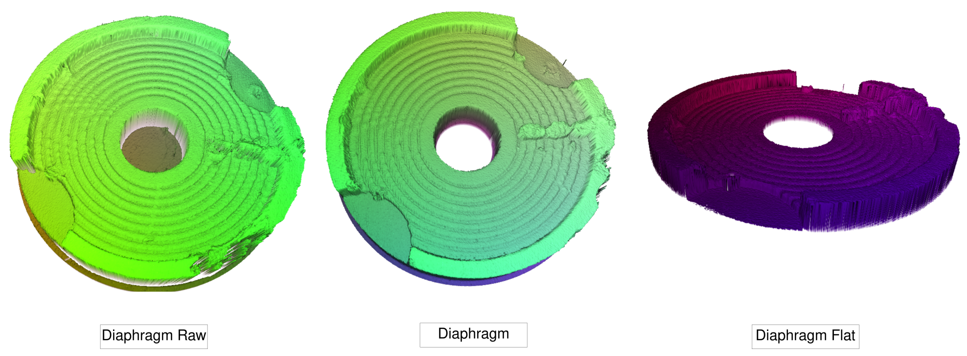


Figure 4.2. Diaphragm raw: only cropping the diaphragm circle.
Diaphragm: same but also the inner circle is cropped and the missing points are restored.
Diaphragm Flat: Limited depth to the known diaphragm notches.

Input	HDIA RanSac Score	HDIA RanSac Qtty Error	ICP Fitness (μm)
Diaphragm Raw	0,502	0,394	951,595
Diaphragm	0,486	0,39	551,168
Diaphragm Flat	0,419	0,334	686,360

Table 4.2. Results after a geometric filter to the scan followed by an HDIA RanSac and a normal ICP. The coarse registration is the mean of 5 executions and 8 threads and the fine is just 5 execution.

As expected, the raw input has the worst registration. But interesting enough the coarse registration was better in the strict (Diaphragm Flat) version than in the moderated one, even so, the ICP fixed this deviation.

Noise

Another inconvenience that may occur is that the scanning process was careless and the result includes a lot of noise. We tested how strong was the RanSac algorithm to see how it may deviate.

To do so we used the Calibration input –is the input with less noise we have and there is no symmetry—. The process consisted of: simplifying to save time, adding noise to the scan cloud points on the Z axis, reconstructing the normals with a normal estimator, proceeding to an HDIA RanSac, recovering the original cloud, and computing the plot values using a threshold of $1mm$.

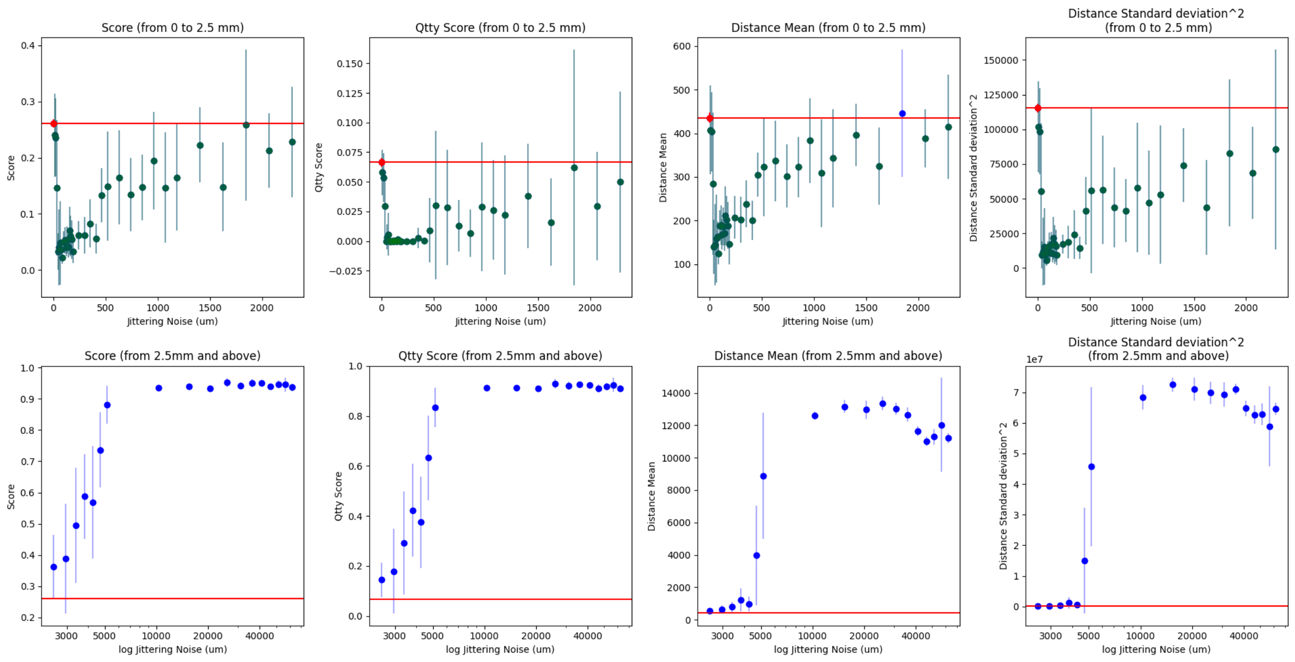


Figure 4.3. In red we have the reference value (no jittering), blue worst, and green better. This result is from averaging 11 executions and the threshold for the score values is $1mm$ for all of them. The resolution length for the models was: Scan $98,37um$ and CAD $109,94um$.

The noise has a clear effect here and the RanSac was given good initial results up to $2,5mm$, then it gets worse until at $5mm$ the registration is useless. One interesting observation, is how the variability at zero noise is really small (not noticeable in the picture) and this variability increases the next $0.1mm$ and then it stabilized for all the samples around $0.1mm$ and $0.5mm$. But after that the RanSac stability is chaotic. Mention again that this plot shows 11 executions where the result is the top value from 8.000 iterations (1.000 iterations by 8 threads).

Since the jittering offered a substantial improvement we repeated the experiment using the Diaphragm input and with two different resolutions.

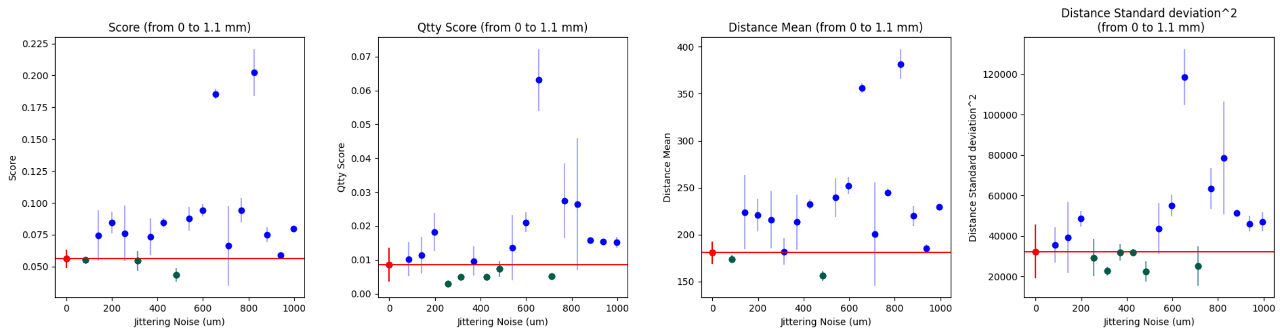


Figure 4.4. Score, Qtty Score, Mean and standard deviation from 0 to $1.1mm$. Here the input resolution for the CAD is $92um$ and for the scan $203um$. This figure is from the results of 3 executions.

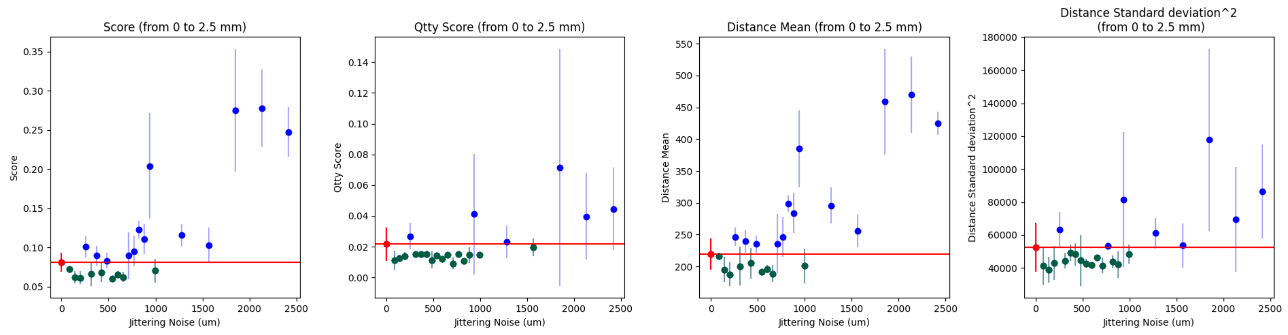


Figure 4.5. Jittering values with CAD resolution $92um$ and scan $128um$. This figure is from the results of 3 executions.

After this, we attempted to implement this idea in the registration adding a Z jittering of half the resolution length (RL_e) before the RanSac. Using the bunny input and only applying a Voxel Grid filter to 20 on both models, we got a good coarse registration —good enough for finishing with a fine registration— in $1m6s$.

4.2.2 HDIA RanSac

An execution is done using 8 HDIA RanSac threads where each one executes 1.000 iterations. Also, the input had already done the pre-process globally, and the final score is the best of them. For better quality in this table 4.3, we used the values of each thread in the score.

Input	Score	Qtty Score	Time[1] Pre-process	Time[1] HDIA RanSac
Calibration	0,004	0%	539,578s ǀ 395,912s	508,867s ǀ 65,798s
Scan at 13.729 Points	0,041	0%	0,804s ǀ 0,113s	883,168s ǀ 111,670s
Diaphragm	0,03	0%	95,632s ǀ 69,888s	1196,44s ǀ 154,26s
Scan at 2.695 Points	0,5	39,888%	2,215s ǀ 0,296s	233,012s ǀ 30,041s
Bunny	0,254	3,26%	227,691s ǀ 164,901s	909,541s ǀ 116,992s
Scan at 20.806 Points	0,537	31,466%	4,118s ǀ 0,709s	1400,085s ǀ 177,178s
USB	0,024	0%	150,3s ǀ 110,939s	392,576s ǀ 50,832s
Scan at 9.060 Points	0,351	16,782%	0,673s ǀ 0,097s	533,638s ǀ 68,627s

Table 4.3. The time is from 3 execution and the score is from averaging the 8 threads (24 in total). These results are from the original input —All scan points and the CAD at the highest resolution (Table 4.1)— and the second result where the scan had a Geometrical filter to a size lower than the HDIA RanSac scan limit.

The concurrency here is done in the FPFH Estimation from PCL (pre-process) and in the 8 threads. As we can see the feature estimator is quite fast and the Poisson Disk Sampling sampling for a high number of points, 40.000, is quite bad. At the same time, reducing the number of in some cases increases the time, this is due that the k-nn is working by distance and the data structure is slowing down.

Despite that this may be interesting. The overall time is lower and in some cases, the penalization in the score values is good enough for a coarse registration. The drawback of this is that this is not linear and the correct size is hard to find.

4.2.3 Input Optimizer

Input	Random Score	Random Time[1]	Ranged Score	Ranged Time[1]
Calibration	0.013	9410,38s ǀ 1348,460s	0.0003	617,517s ǀ 177,576s
Diaphragm	0.006	10028,9s ǀ 1347,44s	0.018	282,588s ǀ 61,734s
Bunny	0.148	19405,7s ǀ 2858,81s	0.136	340,205s ǀ 257,155s
USB	0.019	12180,6s ǀ 1660,86s	0.022	227364s ǀ 55,366s

Table 4.4. Results from one sample with the original input. And the ranged time doesn't include the minimum LOD searcher pre-process time and the maximum LOD is 1 time its resolution.

Note that the table 4.4 includes both coarse and fine registration, and we are doing 5 (2 times as mentioned in Naive Global Optimizer) iterations from minimum to what the random stride arrives. Another interesting observation is that the ranged time has the best times so far and in some cases is good enough, although the same random optimizer proves that they aren't the global solution.

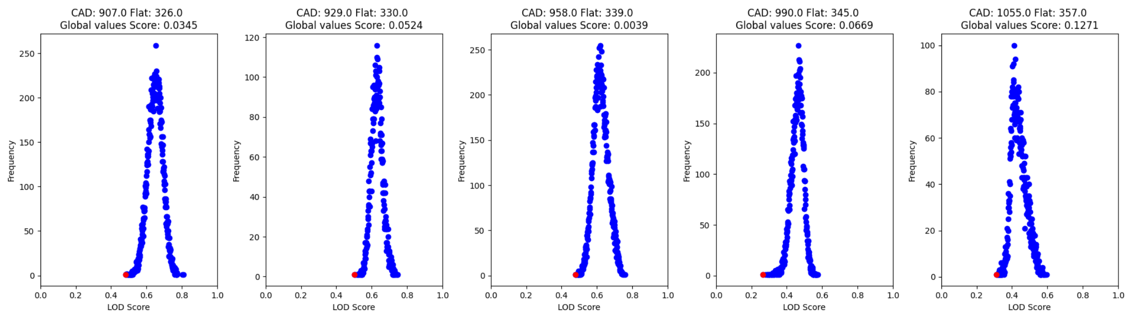


Figure 4.6. Histogram of the different scores obtained by the HD IA RanSac for different LODs and its global score.

4.2.4 Learning Curve

As we have seen so far, the trading between cloud size and our software precision has potential even though in some situations or the score is a local-solution or the time is too big to consider it good enough. For that reason, we have done some manual executions combining our operators to find what could be considered a good solution.

Input	Recipe	Real Time	Score
Calibration	1) Geometrical filter to the scan at 50 2) Ranged local optimizer at [1, 10][5] 3) Limited ICP	3m 42s	0,0001
Diaphragm	1) Geometrical filter to the scan at 20 2) Random local optimizer	14m 52s	0,009[6]
Bunny	1) Ranged local optimizer at [200, 50][5] 2) Geometrical filter both clouds at 20 3) Limited ICP	6m 10s	0,015
USB	1) Geometrical filtered to the scan at 5 2) Normal smoothing at 200 3) Ranged local optimizer at [1, 10][5] 4) Limited ICP	5m 13s	0,006[7]

Table 4.5. Process to get the best results, time from the opening until closing the application, and final score.

As we can see in the table 4.5. The times and score improve considerably only by starting with a bigger LOD and a final ICP limited to the 10 times resolution length, another interesting operator is smoothing the scan normals —any small scanner noise can change drastically the normals—. The drawback is that knowing the initial LOD is particular to each input.

4.3 Visual Representation

4.3.1 Error Calculator

Here we have some values of how much time costs to smooth the normals to reduce the noise and compute the distance values. For the moment we computed both distances from CAD to scan and vice-versa since we are using a point-to-disk approach we could reduce the CAD size to speed up the time.

Input	Smooth Normals Time[1]	CPU Time
Calibration	8,331s | 2,963s	11,435s
Diaphragm	9,142s | 2,034s	2,156s
Bunny	308,863s | 39,837s	4,217s
USB	3,666s| 1,147s	2,523s

Table 4.6. Smoothing and error extracting times from 3 executions.

For the precision, we did multiple executions to see how the sampling could variate. In table 4.6, we started using a 5% sampling but the execution time was too big and we limited it to the minimum between the 5% and 50.000 samples. Another optimization we added is to execute in concurrent the CAD distance calculations.

[5][biggest cloud size, stride to search the minimum]

[6]This input has some false local solution around this value and the solution is not stable

[7]The solution is not stable. The simple reason is that an upside-down register is worst in the connector but better in strain relief

Input	Time[1]	Precision	Standard Deviation
Calibration	275,6s ǀ 37,7s	3,58 μm	0,008
Diaphragm	916,8s ǀ 131,0s	521,67 μm	0,020
Bunny[8]	19,49h ǀ 2,49h	2,10 μm	-
USB	1.383,8s ǀ 194,9s	1.100,13 μm	0,051

Table 4.7. Precision values and its time. Average of 5 executions.

As we can see, in the table 4.7, the sampling looks stable but the computing CAD distance is highly demanding and almost impossible in the Bunny input where the scan is a subset of the CAD. Interestingly enough the precision values seem positive. Though theoretically, we are encapsulating the three errors in one value (scanning noise, registration error, and wear/input error) the precision offers a good uncertainty for an analysis, when the input has small differences like the Calibration and Bunny. But this encapsulation is disturbing the Diaphragm precision which has many input differences.

4.3.2 Visual Comparison Toolkit

Tough to extract some real results we should do user testing. For the moment we can see that the values can be easily visualized and configuring the color range is useful.

Also subjectively, having the two models at the same time is a little hard to understand but our Transparency options offer good options: Using a mate color for one of them is better to understand only the missing material of one part; having constant transparency is handy in the exchange of some color perturbations but let the user compare how much is missing and how much material is left; and the transparency following the error has a subtle difference but has a less information in the screen, offering cleaner image. We collected all options using the Bunny data set (Figure 4.7).

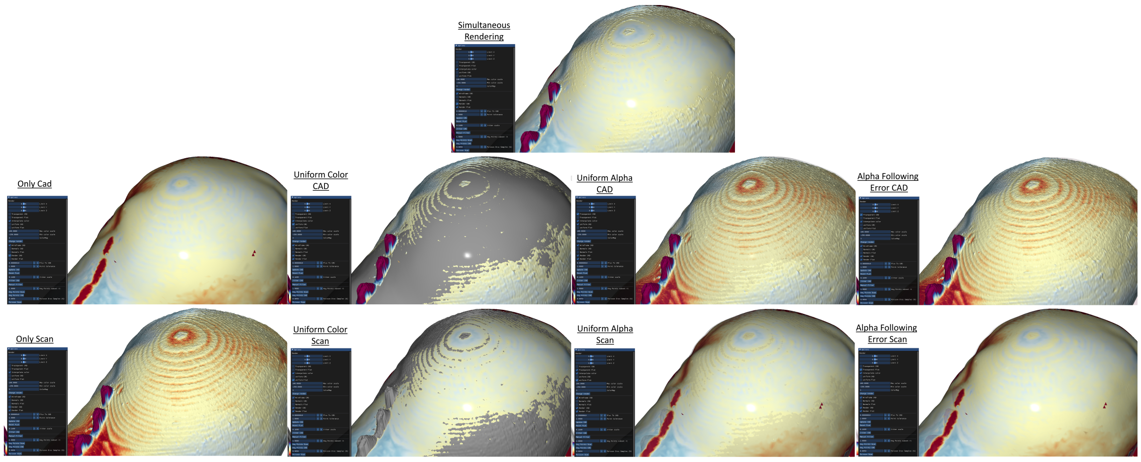
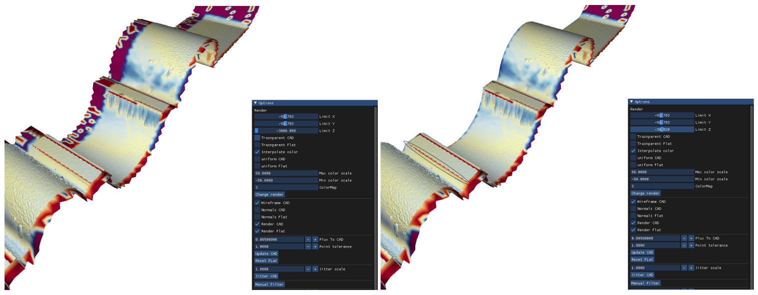


Figure 4.7. Some visualizations of the Bunny sample. On the left, we have both at the same time, and in the upper row, we have the CAD as a priority and the lower the Flat. From left to right: both, only one, the secondary with constant color and the secondary with constant transparency.

One big flaw we can see is that we have 2.5D data, they aren't solid, and they may represent only a part of the CAD design. So when we compute the encoding for those points the sign of the value may be wrong. For those cases, the clipping Plane can be helpful to remove this kind of outliers (Figure 4.8). Another problem we can see here is when the CAD resolution isn't enough and we can see the borders of two disks with different normals.


```
▼ Options
Render
        -92.783   Limit X
        -92.783   Limit Y
      -3000.000   Limit Z
[ ] Trasnparent CAD
[ ] Trasnparent Flat
[✓] Interpolate color
[ ] uniform CAD
[ ] uniform Flat
50.0000           Max color scale
-50.0000          Min color scale
3                 ColorMap
Change render
[✓] Wireframe CAD
[ ] Normals CAD
[ ] Normals Flat
[✓] Render CAD
[✓] Render Flat
0.00500000   - +  Plux To CAD
1.0000       - +  Point tolerance
Update CAD
Reset FLat
1.0000       - +  Jitter scale
Jitter CAD
Manual Filter
```

```
▼ Options
Render
        -92.783   Limit X
        -92.783   Limit Y
        -30.928   Limit Z
[ ] Trasnparent CAD
[ ] Trasnparent Flat
[✓] Interpolate color
[ ] uniform CAD
[ ] uniform Flat
50.0000           Max color scale
-50.0000          Min color scale
3                 ColorMap
Change render
[✓] Wireframe CAD
[ ] Normals CAD
[ ] Normals Flat
[✓] Render CAD
[✓] Render Flat
0.00500000   - +  Plux To CAD
1.0000       - +  Point tolerance
Update CAD
Reset FLat
1.0000       - +  Jitter scale
Jitter CAD
Manual Filter
```

Figure 4.8. Clipping Plane on the Z axis to clean CAD outliers.

Another interesting observation is that the shader interpolation for the values is quite handy for CAD visualization since the faces are considerably big. But this interpolation in the scan may trick the user into a thing this is a continuous value instead of a discrete one that already has a high density (en example is illustrated in Figure 4.9).

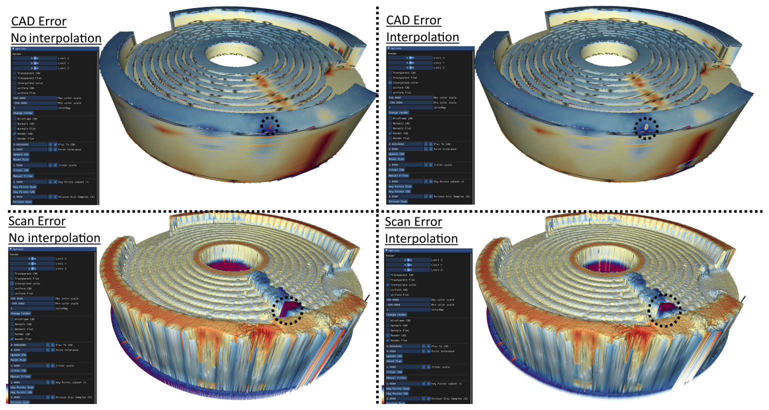


Figure 4.9. Interpolation on/off for CAD and Scan error. Note the dotted circle is highlighting differences.

Finally the color maps we selected seems to be informative and helpful to understand the error (Figure 4.10).

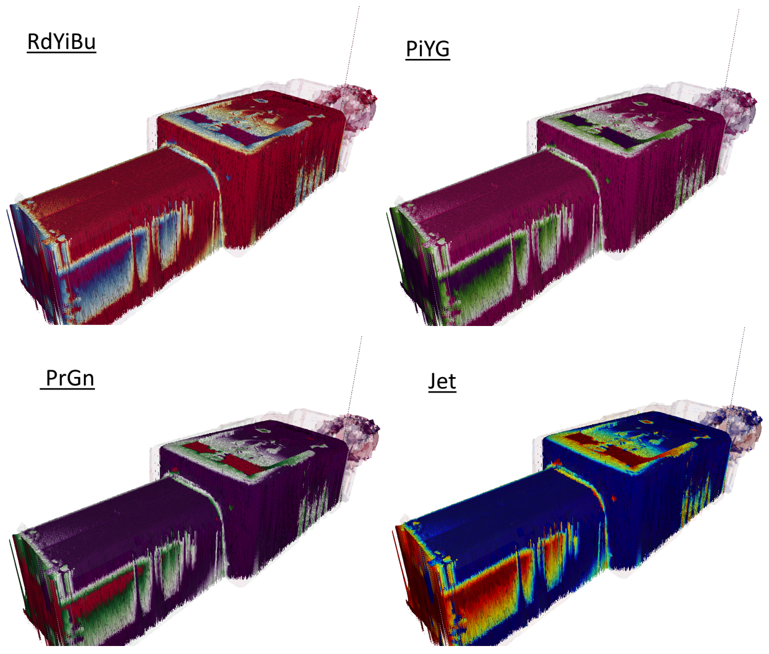


Figure 4.10. All 4 colormaps we implemented on the USB input and CAD are transparent following the error.

5. Conclusions

5.1 Results Discussions

After collecting all the results we can extract some discussions.

From the CAD processing we can conclude that the times are worst than normal processing. We achieved our goal for the regular distance point cloud, but there are some ill inputs that need a different approach.

The Registration showed good results, the algorithms are subject to how accurate the cleaning and scanning process is, but our algorithm is robust enough to find a good solution.

Our first consideration —having the two inputs with a similar regular distance— was almost correct. Still, we didn't consider the magnitude of the worse case. When the regular distance adds a constraint to the SVD transformation matrix approximation and the best solution is obstructed —e.g. trying to match two waves with the same frequency but different phases—. Now with the jittering results, we see that having the two inputs with similar RL and jittering one a little bit increases the efficiency.

At the same time the LODs concept is partially valid, the jittering makes more accessible the global-solution but the high-density points we have is unfeasible and this idea could be reused for validation criteria or time optimization.

In the visualization we need a user study to determine the viability of the transparency policies and interpolation.

Also we overestimated how the point-to-disk distance works and in some cases we have small artifacts because of bad normals and/or low-density points in some regions. Maybe we should consider a classic approach point-to-mesh, which may be slower, or improve the CAD. Nevertheless, these artifacts have an easy fix.

5.2 Final Conclusion

The toolkit we have developed includes enough options to have a good registration and visualize the error in a reasonable amount of time. But depending on the input complexity and the user experience the automatic options aren't good enough and a manual solution should be done. Even though the operator parameters are simple (one or none) and can be scripted so the recipe solution can be reused.

On the bad side our CAD processing depends a lot on how well is designed the input. The time increases badly and if it's built using boolean operators the time suffers a big penalization and also the regular distance between points may not be achieved.

At the same time we adopted the IA RanSac to speed up its time, have better results, and parameterized its input parameters to reduce it to *number of iterations* and two optional more, *percentage* and *correspondences*. Also, the LODs strategy we found seems a starting point to solve the registration problem for our input magnitude or bigger.

For the error visualization we have a completed set of options to view the error in different scales and keep track of the reference model.

5.3 Future work

The major drawback in the CAD processing is merging the Tears and its time increases with the number of features, one way to optimize this is creating a better data structure to configure the k-nn by features and not only interior-border points.

Our HDIA RanSac can be enhanced. First, there is no information shared between iterations so we can paralyze a lot more. But also the Part-to-Model is incomplete. For instance, we are ignoring the vectors' direction between samples and our distance idea can be applied similarly to the angle. A more elaborated idea could be a second level of features similarity evaluation to weigh the random.

The Naive Global Optimizer is a naive solution to prototype the viability of the LODs concepts. With the results we have, we can say that this idea can be improved. Starting with the pre-process of finding the minimum LOD, we can automate with a binary search. Again the LODs are causing a bottleneck because we are repeating the Voxel Grid simplification, this can be computed once and stored for later reuse.

Also. after adopting the jittering depending on the LOD detail, we could use this optimizer with the simulated annealing method. Using this method we can define many options. Like a dynamic number of iterations depending on the LOD —for example, the Bunny can find the solution with only 80 iterations, in contrast, the USB needs the 8.000—, or a termination criteria, or even a voting IA classifier for the coarse alignment.

For the Precision value we can have better times if instead of measuring against the whole CAD for every point, we detect first the CAD features candidates with the feature borders points and the inner points will be only against a subset of features and not the whole design. Simultaneously this optimization will help to what a good sampling ratio may be. And another interesting topic could be a new precision definition to separate the three errors (scanning noise, registration error, and wear/input error).

The visualization we need user feedback to know if Transparency has potential. At the same time, the visualization is interesting, but this data should be exportable and quantifiable.

6. References

Bibliography

[1] Roger Artigas, Ferran Laguarta, and Cristina Cadevall. Dual-technology optical sensor head for 3d surface shape measurements on the micro- and nanoscales. In Wolfgang Osten and Mitsuo Takeda, editors, *Optical Metrology in Production Engineering*. SPIE, September 2004.

[2] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, 1987.

[3] Carlos Bermudez, Pol Martínez, Cristina Cadevall, and Roger Artigas. Active illumination focus variation. In Peter Lehmann, Wolfgang Osten, and Armando Albertazzi Gonçalves, editors, *Optical Measurement Systems for Industrial Inspection XI*. SPIE, June 2019.

[4] Open CASCADE. Brepmesh_incrementalmesh class reference. `https://dev.opencascade.org/doc/refman/html/class_b_rep_mesh___incremental_mesh.html`.

[5] Chu-Song Chen, Yi-Ping Hung, and Jen-Bo Cheng. RANSAC-based DARCES: a new approach to fast automatic registration of partially overlapping range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1229–1234, 1999.

[6] Daniel Cohen-Or, Chen Greif, Tao Ju, Niloy J. Mitra, Ariel Shamir, Olga Sorkine-Hornung, and Hao (Richard) Zhang. *A sampler of useful computational tools for applied geometry, computer graphics, and image processing*. CRC Press, 2020.

[7] Stefano Corazza, Lars Mündermann, Emiliano Gambaretto, Giancarlo Ferrigno, and Thomas P. Andriacchi. Markerless motion capture through visual hull, articulated icp and subject specific model generation. *International Journal of Computer Vision*, 87(1):156, Sep 2009.

[8] Mark Harrower Cynthia Brewer. Color brewer 2.0. `https://colorbrewer2.org/#type=diverging&scheme=RdYlBu&n=11`.

[9] Bailin Deng, Yuxin Yao, Roberto M. Dyke, and Juyong Zhang. A survey of non-rigid 3d registration. *Computer Graphics forum*, 41:559–589, 2022.

[10] Sai Siva Gorthi and Pramod Rastogi. Fringe projection techniques: Whither we are? *Optics and Lasers in Engineering*, 48(2):133–140, 2010. Fringe Projection Techniques.

[11] Oshri Halimi, Or Litany, Emanuele Rodola Rodola, Alex M. Bronstein, and Ron Kimmel. Unsupervised learning of dense shape correspondence. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019.

[12] R. Hänsch, T. Weber, and O. Hellwich. Comparison of 3d interest point detectors and descriptors for point cloud fusion. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3:57–64, August 2014.

[13] Industrial automation systems and integration — product data representation and exchange — part 21: Implementation methods: Clear text encoding of the exchange structure. Standard, International Organization for Standardization, Geneva, CH, March 2016.

[14] András P. Keszei, Benjamin Berkels, and Thomas M. Deserno. Survey of non-rigid registration tools in medicine. *Journal of Digital Imaging*, 30(1):102–116, October 2016.

[15] Point Cloud Library. impl/fpfh.hpp. https://github.com/PointCloudLibrary/pcl/blob/master/features/include/pcl/features/impl/fpfh.hpp#L238.

[16] Point Cloud Library. impl/ia_ransac.hpp. https://github.com/PointCloudLibrary/pcl/blob/master/registration/include/pcl/registration/impl/ia_ransac.hpp#L79.

[17] Point Cloud Library. impl/transformation_estimation_svd.hpp. https://github.com/PointCloudLibrary/pcl/blob/master/registration/include/pcl/registration/impl/transformation_estimation_svd.hpp#L129.

[18] Point Cloud Library. Module registration. https://pointclouds.org/documentation/group__registration.html.

[19] Point Cloud Library. pcl::voxelgrid class reference. https://pointclouds.org/documentation/classpcl_1_1_voxel_grid_3_01pcl_1_1_p_c_l_point_cloud2_01_4.html.

[20] Adele Lorusso, David W Eggert, and Robert B Fisher. *A comparison of four algorithms for estimating 3-D rigid transformations*. Citeseer, 1995.

[21] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[22] A. NÜCHTER. *3D Robotic Mapping The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer, 2009.

[23] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps. *ACM Transactions on Graphics*, 31(4):1–11, August 2012.

[24] Gianluca Percoco, Maria G. Guerra, Antonio J. Sanchez Salmeron, and Luigi M. Galantucci. Experimental investigation on camera calibration for 3d photogrammetric scanning of micro-features for micrometric resolution. *The International Journal of Advanced Manufacturing Technology*, 91(9-12):2935–2947, 08 2017. Copyright - The International Journal of Advanced Manufacturing Technology is a copyright of Springer, (2017). All Rights Reserved; Última actualización - 2019-07-23.

[25] Sai Manoj Prakhya, Bingbing Liu, Weisi Lin, Vinit Jakhetiya, and Sharath Chandra Guntuku. B-SHOT: a binary 3d feature descriptor for fast keypoint matching on 3d point clouds. *Autonomous Robots*, 41(7):1501–1520, December 2016.

[26] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

[27] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Michael Beetz, Intelligent Autonomous Systems, and Technische Universität München. Persistent point feature histograms for 3d point clouds. In *In Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10*, 2008.

[28] Yusuf Sahillioğlu. Recent advances in shape correspondence. *The Visual Computer*, 36(8):1705–1721, September 2019.

[29] Sensofar Tech. SL. Sensofar industry & research. https://www.sensofar.com/metrology/industry-research/.

[30] G. K. L. Tam, Zhi-Quan Cheng, Yu-Kun Lai, F. C. Langbein, Yonghuai Liu, D. Marshall, R. R. Martin, Xian-Fang Sun, and P. L. Rosin. Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1199–1217, July 2013.

[31] Open CASCADE Technology. Shapeanalysis_surface class reference. https://dev.opencascade.org/doc/refman/html/class_shape_analysis___surface.html.

[32] Open CASCADE Technology. Topods class reference. https://dev.opencascade.org/doc/refman/html/class_topo_d_s.html.

[33] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Computer Vision – ECCV 2010*, pages 356–369. Springer Berlin Heidelberg, 2010.

A. Resources

List of software

- Programming languages: c++20, python3.6, bash
- Build automation: Cmake (v3.16.3)
- OpenGL API: gl3w, openGL4.6-freeglut
- UI: Imgui (v1.86)
- CAD: OpenCASCADE-OCCT (v.7.6.1)
- Maths: Eigen (v.3.4.0)
- Point cloud: PCL (v1.12.1)
- kd-tree: nanoflann (v1.3.0)
- zip: LibZip (v1.8.0)
- XML: rapidXML (v1.13)

List of Tools

- OS: Ubuntu 20.04
- Github
- Atom
- Paint.net
- Overleaf
- Grammarly
- DrawIO
- SensoView 1.9
- FreeCAD 0.19
- Blender 3.2
- Kazam 1.4.5

List of Hardware

- SensoSCAN S Wide
- SensoSCAN S Neox
- Formlabs Form 3, Form Wash and Form Cure
- Lenovo Ideapad 530S-14IKB Intel[1] Core i5-8250U/8GB/256 SSD/14"

[1]CPU with 4 cores, 8 threads and a max frequency 3.40GHz

Timesheet

Field	Time (h)
Organization	24,85
Coding	376,79
Input samplings and execution analysis	61
Research	58,74
Writing the memory	188,35
Estimated time for the presentation	24,27
Others	16

Table 1.1. time spent in each field.

B. Open Resources

Basic code framework:

`https://github.com/serk12/gl3w-Template`

Latex schema, pictures, videos (lfs) and raw results:

`https://github.com/serk12/TFM-latex-Scheme`

Demo video:

Drive