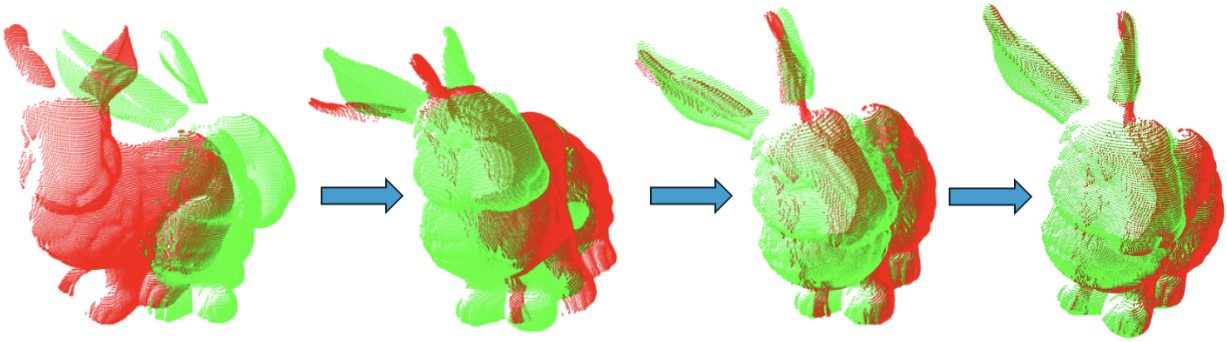
Partial Shape Alignment Using SHOT Descriptors and Optimized Alignment Algorithms

Adam Thomsen, Anis Vrevic, Cecilie Marie Jensen,
Fruzsina Nagy and Oliver Østergaard Olsen

Course: MM571:Førsteårsprojekt

June 2, 2025



Abstract: *3D alignment is an important task in computer vision and robotics, with most research focused on full-to-full point cloud alignment, but with partial-to-partial alignment, many existing algorithms struggle. In this paper, we propose an altered pipeline for partial alignment that combines a modified SHOT-based feature matching strategy with focused variants of RANSAC and ICP. Our approach focuses on higher confidence in correspondences and adapts the alignment process to the limited overlap. Experiments demonstrate that our method outperforms standard alignment pipelines when applied to partial data, although there is still room for improvements. These results show that stricter feature matching and restricted input are needed in partial registration contexts.*

Contents

1 Introduction

As computer vision and robotics develop, the need for 3D surface recognition and shape alignment rises. The main task is to give computers the ability to recognize similarities between 3D surfaces and be able to align shapes, for example, to detect movement. Shape alignment is applicable in many other fields too, including automation, biometric systems, and reverse engineering [19].

Early shape alignment methods relied on direct comparison of raw geometry, but these approaches proved to be sensitive to noise and partial views. To improve robustness, local feature descriptors such as Spin Images, Point Feature Histograms (PFH), and Fast PFH (FPFH) were developed. These descriptors summarize the geometric properties of a points neighborhood, generating a descriptor, making it possible to estimate correspondences between surfaces [19].

In more recent research, Signature of Histograms of Orientations (SHOT), in particular, has gained popularity due to its robustness to noise and invariance to rotation. Once features are matched, transformation estimation typically follows via Random Sample Consensus (RANSAC), which selects a set of correspondences and estimates a rigid transformation that best fits the majority of them. This is often refined further using Iterative Closest Point (ICP), or one of its variants, which is an optimization algorithm that minimizes distances between nearest neighbors to fine-tune the alignment [21] [19] [16].

Despite progress in feature descriptors and alignment algorithms, most existing research assumes that both input shapes represent full, overlapping scans of an object or scene. This full-to-full assumption simplifies correspondence searching and alignment, as large shared regions make the feature matching and transformation estimation steps easier. In contrast, partial-to-partial alignment, where only a small portion of the shapes is available, remains underexplored [22]. Standard pipelines based on full-to-full assumptions often perform poorly in these cases, failing to produce reliable matches or satisfying transformations. This reveals a need for altered or new methods that can handle partial overlap, which is the focus of this paper.

2 Basics of Shape Alignment

2.1 Point clouds

A point cloud is a set of data points in a 3D coordinate system. Point cloud data is a detailed representation af a 3D object, where each point represents a single geometric measurement of the objects surface.

Point clouds are usually created from 3D scanners, such as LiDAR, which uses laser light to measure distances to objects. From these scans we get an abundance of points, which together form a point cloud of an objects surface [9].

Definition 1. *Let $\mathcal{X}$ be a point cloud, then $\mathcal{X}$ can be written as the matrix:*

$$\mathcal{X} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix}$$

Where each row is a vector x_i representing one of the N points in the point cloud. For convenience, we will be writing each vector as column vectors.

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{pmatrix}$$

2.2 Types of 3D-alignment

3D point cloud alignment refers to the process of transforming one point cloud (scan) so that it best matches another point cloud (reference), under a defined geometric transformation. In this paper, we will show examples of three different types of alignments with 3D point clouds:

1) Full-to-full alignment
2) Partial-to-full alignment
3) Partial-to-partial alignment

2.2.1 Full-to-full alignment

Full-to-full alignment refers to an alignment where both the scan and the reference point clouds represent a complete and fully observed 3D model of the same object. Hence, the whole surface is available in both datasets.

Some characteristics of full-to-full alignment are the full overlap between the point clouds, the suitability for rigid or non-rigid transformations, and the assumption of similar scale and resolution.

2.2.2 Partial-to-full alignment

Partial-to-full alignment refers to an alignment where the scan is a partial observation of an object and the reference is the complete 3D model of the same object. The scan may be missing large portions of the point cloud due to occlusion, incomplete scanning, or limited field of view. It is also

possible to do full-to-partial alignment, where the cases are switched, such that the full dataset is the scan, and the reference is the partial point cloud.

Some characteristics of partial-to-full alignment are the asymmetric completeness and the partial overlap, where the scan may contain noise or outliers. Thus, it requires robust matching techniques that are tolerant to the missing data.

2.2.3 Partial-to-partial alignment

Partial-to-partial alignment refers to an alignment where both the scan and the reference point cloud are incomplete, often with unknown or small overlapping regions. This is the most complex alignment out of the three types, due to the ambiguity and limited correspondence.

Some characteristics of partial-to-partial alignment are the minimal or uncertain overlap, the sensitivity to noise and partial views, which therefore requires an estimation of both the alignment and the overlap region. A lower bound for the latter is often nessesary.

2.3 Covariance matrices

Within shape alignment, it turns out that the covariance matrix is a concept often encountered. The covariance matrix is a square, symmetric matrix, that shows the covariances between pairs of variables in multidimensional data.

Suppose what we have some point cloud $\mathcal{X} \in \mathbb{R}^{N\times 3}$. Let X, Y and Z represent the set of all x, y and z coordinates in the point cloud, respectively. Our goal is to describe how strongly these coordinates relate to one another. The structure of the covariance matrix in 3-dimensional space looks as follows [5]:

$$C = \begin{bmatrix} \mathrm{Var}(X) & \mathrm{Cov}(X,Y) & \mathrm{Cov}(X,Z) \\ \mathrm{Cov}(Y,X) & \mathrm{Var}(Y) & \mathrm{Cov}(Y,Z) \\ \mathrm{Cov}(Z,X) & \mathrm{Cov}(Z,Y) & \mathrm{Var}(Z) \end{bmatrix}$$

To find the variance, you first need the centroid of the dataset, which is calculated as the average coordinate value of all coordinates:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Now to find the variance of each variable, we take the sum of the squared distances between each point and the centroid, and divide it by the number of points.

$$\mathrm{Var}(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Now, to find the covariances, we again normalize each coordinate and compute [5]:

$$\mathrm{Cov}(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

The sign of the covariance indicates the linear dependence between the two datasets (here the coordinates). For positive (respectively, negative) values, there exist a positive (respectively, negative) linear correspondence between the dataset. If the covariance turns out to be zero, then no such relation exists .

Covariance matrices are very useful when working with point clouds. If one sorts the eigenvalues of the matrix by size and considers the corresponding eigenvectors, these will describe the directions with the most variance (as proved in the Appendix A). In three dimensions, the eigenvectors to the smallest eigenvalue can therefore be thought of as the normal direction. We will elaborate on this later on. The covariance matrix will prove to be useful when performing alignments.

2.4 Singular Value Decomposition (SVD)

In linear algebra, the singular value decomposition is the factorization af a real or complex matrix A into the product of three matrices $A = U\Sigma V^T$. Here U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix with singular values on the diagonal, and V is an $n \times n$ orthogonal matrix [7].

Before we continue this section on SVD, we first need some results on orthonormal vectors.

Theorem 1 (Spectral theorem)**.** *Let A be a symmetric real $n \times n$ matrix, then A is orthogonally diagonalizable. This means that there exists an orthogonal matrix Q and a diagonal matrix D, such that*

$$A = QDQ^T$$

Theorem 2 (Gram-Schmidt)**.** *Let $\mathbb{F} = \{\mathbb{R}, \mathbb{C}\}$ and $\{w_1, \ldots, w_p\} \subset \mathbb{F}^n$ be a set of linearly independent vectors. Then there exists a set of orthonormal vectors $\{q_1, \ldots, q_p\} \subset \mathbb{F}^n$, such that*

$$span\{w_1, \ldots, w_k\} = span\{q_1, \ldots, q_k\} \quad \forall k = 1, \ldots, p$$

Definition 2. *The Kronecker delta is a function of two variables, defined as*

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Notice that for an orthonormal set of vectors V, we have that $v_i^T v_j = \delta_{ij}$.

Theorem 3 (SVD)**.** *Let A be a real $m \times n$ matrix. Then there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ with nonnegative real numbers on the diagonal, such that*

$$A = U\Sigma V^T$$

Before proceeding with the proof, let's first gain some insight into the nature of this decomposition. We interpret the rows of a matrix A as m points in an n-dimensional space. We aim to minimize the sum of the squared perpendicular distances from the points to the subspace.

This leads to a problem known as the *best least squares fit*. Consider the special case of a 1-dimensional subspace. we seek the best-fitting line through the origin, minimizing the sum of squared distances from the points $\{x_i \mid 1 \leq i \leq m\}$ to this line.

To understand this geometrically, let x_i be a point projected onto a line through the origin. Then:

$$x_{i1}^2 + x_{i2}^2 + \cdots + x_{in}^2 = (\text{length of projection})^2 + (\text{distance to line})^2$$

Hence, the squared distance from x_i to the line is:

$$(\text{distance to line})^2 = x_{i1}^2 + x_{i2}^2 + \cdots + x_{in}^2 - (\text{length of projection})^2$$

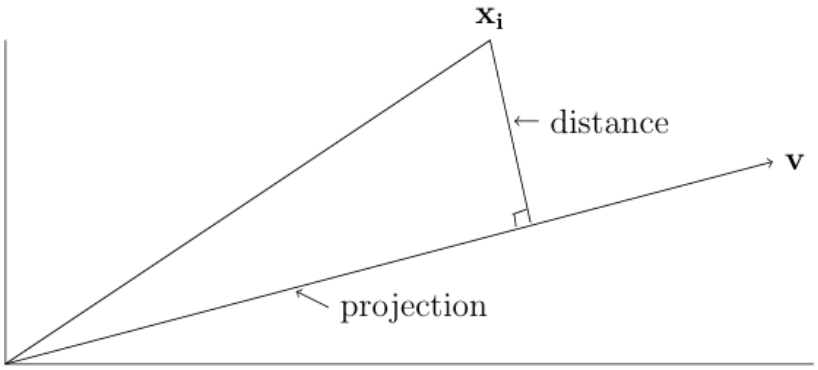


Figure 1: Projection of a point x_i onto the line through the origin in the direction of v [8].

Because the total squared norm $\sum_{j=1}^{n} x_{ij}^2$ is constant for each x_i, minimizing the sum of squared distances is equivalent to *maximizing* the sum of squared lengths of the projections onto the line.

We now introduce the *singular values* of an $m \times n$ matrix A. Let v be a unit vector in $\mathbb{R}^n$ representing the direction of the best fit line. The projections of the ith row x_i onto v has length $|x_i \cdot v|$, so the total squared norm of the projection is $\|Av\|^2$. Thus, the best-fit line corresponds to the unit vector v that maximizes $\|Av\|$:

$$v_1 = \arg\max_{\|v\|=1} \|Av\|$$

We define $\sigma_1(A) := \|Av_1\|$ as the *first singular value* of A.

The *second singular vector* v_2 is the direction orthogonal to v_1 that best fits the data, defined as:

$$v_2 = \arg\max_{\substack{v \perp v_1 \\ \|v\|=1}} \|Av\|$$

The corresponding *second singular value* is $\sigma_2(A) := \|Av_2\|$. Continuing this process, we define successive singular vectors $v_3, v_4, \ldots, v_r$, where $r = \text{rank}(A)$, as the best directions orthogonal to the previous ones, until:

$$\arg\max_{\substack{v \perp v_1, \ldots, v_r \\ \|v\|=1}} \|Av\| = 0$$

Similarly, it is possible to show that the normalized eigenvector corresponding to the largest eigenvalue of the covariance matrix is equal to $\pm v_1$, as shown in Appendix A.

The matrix A can be completely described by its action on these r singular vectors. Any vector $v \in \mathbb{R}^n$ can be written as a linear combination of $v_1, \ldots, v_r$ and some vector orthogonal to all of them. The image Av is then the same linear combination of $Av_1, \ldots, Av_r$. We normalize these images to unit length by defining:

$$u_i := \frac{1}{\sigma_i} A v_i$$

The vectors $u_1, \ldots, u_r$ are the *left singular vectors*, while $v_1, \ldots, v_r$ are the *right singular vectors* [8].

Theorem 4. *Let A be an $m \times n$ matrix with right singular vectors $v_1, \ldots, v_r$, corresponding singular values $\sigma_1, \ldots, \sigma_r$, and left singular vectors $u_1, \ldots, u_r$. Then the reduced SVD is.*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

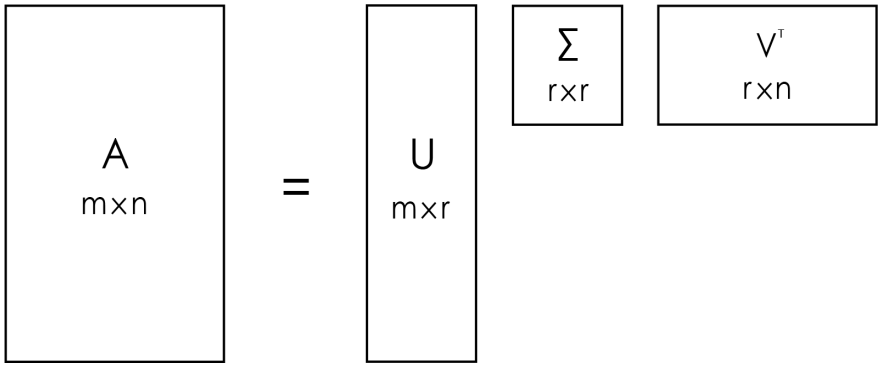


Figure 2: Visual representation of the reduced SVD decomposition of a $m \times n$ matrix

In matrix notation, this can be written as $A = U\Sigma V^T$, where U and V consists of the left and right singular vectors, and Σ is the diagonal marix whose diagonal entries are the singular values of A. For any matrix A, the sequence of singular values is unique, and if the singular values are distinct, then the sequence of singular vectors is also unique. However, if some set of singular values is equal, then the corresponding singular vectors span some subspace. Any set of orthonormal vectors spanning this subspace can be used as the singular vectors [8].

Now we are ready to prove Theorem 3.

Proof Theorem 3. Let $A \in \mathbb{R}^{m\times n}$ and define $M := A^T A \in \mathbb{R}^{n\times n}$. Since M is symmetric and positive semi-definite, the spectral theorem guarantees an orthogonal matrix $V \in \mathbb{R}^{n\times n}$ and a diagonal matrix $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$. Here λ_i are the eigenvalues of M, and the columns of V are the corresponding eigenvectors $v_1, v_2, \ldots, v_n$. Moreover

$$\lambda_1 \geq \cdots \geq \lambda_r > 0 = \lambda_{r+1} = \cdots = \lambda_n$$

where $r = \text{rank}(A)$, such that

$$M = VDV^T$$

Define the singular values by $\sigma_i := \sqrt{\lambda_i} \geq 0$, and set

$$\Sigma := \begin{bmatrix} \text{diag}(\sigma_1, \ldots, \sigma_r) & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times n}$$

For $i = 1, \ldots, r$, define

$$u_i := \frac{1}{\sigma_i} A v_i \in \mathbb{R}^m$$

These vectors form an orthonormal set since

$$u_i^T u_j = \frac{1}{\sigma_i \sigma_j} v_i^T A^T A v_j = \frac{1}{\sigma_i \sigma_j} v_i^T M v_j = \frac{\lambda_j}{\sigma_i \sigma_j} v_i^T v_j$$

Here, $v_i^T v_j = \delta_{ij}$ denotes the Kronecker delta. Thus,

$$u_i^T u_j = \frac{\lambda_j}{\sigma_i \sigma_j} \delta_{ij} = \delta_{ij}$$

Extend $\{u_1, \ldots, u_r\}$ to an orthonormal basis $\{u_1, \ldots, u_m\}$ of $\mathbb{R}^m$ with Gram-Schmidt, and form the orthogonal matrix

$$U := [u_1 \quad u_2 \quad \cdots \quad u_m] \in \mathbb{R}^{m \times m}$$

For $i > r$, since $\lambda_i = 0$, it follows that $Av_i = 0$. Thus,

$$AV = [Av_1 \quad \cdots \quad Av_r \quad 0 \quad \cdots \quad 0] = U\Sigma$$

and multiplying both sides by V^T yields

$$A = U\Sigma V^T$$

□

2.4.1 Use of SVD in 3D-alignment

Singular Value Decomposition can be used as a fast and efficient method for aligning 3D point clouds. It is possible to use SVD to compute the optimal rotation matrix between two sets of 3D points without requiring iterative optimization, like in the common methods of 3D alignment. The way to do this is to first compute the cross-covariance matrix of the two centered point sets. We elaborate in section 3.2. Then we apply SVD to this cross-covariance matrix $H = U\Sigma V^T$. Lastly, we calculate the optimal rotation matrix R as $R = VU^T$ [14]. We will elaborate on this later.

3 Algorithms

3.1 Signature of Histograms of OrienTations (SHOT)

Partial shape alignment relies heavily on finding satisfying correspondences between the partial shapes, meaning finding points that correspond to the same point in the full shape. This might be easy to do for human eyes in a lot of cases, but a computer needs detailed instruction to know what to compare and how to deduce correspondence. For this we need some way to describe points in the point clouds representing the shapes. This description needs to have a form that can be compared with descriptions of other points to find matches. There are several ways to do this but in this paper we will focus on the SHOT algorithm proposed by Tombari, Salti and Di Stefano [19]. The algorithm is a 3D local surface descriptor with the ability to capture geometrical information around a given point on a 3D surface. The found descriptor of a point can then be compared to those of other points. The algorithm is an improvement from earlier descriptors as it addresses repeatability issues and rotational dependency. Overall, the algorithm consists of 4 steps:

SHOT algorithm

1) Local Reference Frame (LRF) Computation
2) Support Region Definition
3) Spatial Partitioning of the Support Volume
4) Orientation Histogram Computation

3.1.1 Local Reference Frame

The algorithm uses local reference frames to ensure rotational invariance. Rather than basing the descriptor on a globally defined 3D-grid, the algorithm constructs a local reference frame around each point when it is being described. The point being described will be referred to as the "keypoint" p. The fact that an LRF is constructed around the given keypoint ensures that the descriptor remains consistent even if the object is rotated in space. The goal with LRF is essentially to define a repeatable and consistent 3D coordinate system centered at each keypoint. With this method, all features computed are comparable, no matter the rotation. The LRF for a keypoint p is computed by defining a support region of radius r centered at p, and then defining a set of all points q_i within this sphere, the neighboring points. Using these points, a covariance matrix (see section 2.3) is computed. This describes the spacial distribution of the local surface. In [19], the covariance matrix is weighted by a distance based weighing function giving points closer to p more importance, for stability. The weighted covariance matrix is the following:

$$C = \frac{1}{\sum w_i} \sum w_i (q_i - p)(q_i - p)^T$$

Where $w_i = r - \parallel q_i - p \parallel$ is the weight given to a point depending on the distance from the keypoint and whether it is within the radius of the support region.

Then an eigenvalue decomposition is performed on the weighted covariance matrix, where the found normalized eigenvectors define the axes of the local frame. The smallest one (the keypoint normal) of these typically aligns with the surface normal (see section 2.3). The other eigenvectors span the tangent plane, where the variance is the highest (see Appendix A). The paper [19] mentions the fact that the eigenvectors from PCA-like analysis are directionally ambiguous and so resolving this issue is necessary to ensure repeatability. They briefly propose a way of doing so this by ensuring the eigenvectors have signs that correspond to the majority of the points they represent. After doing

so, the LRF is now a 3D orthonormal basis consisting of x, y and z axes corresponding to the three eigenvectors. The LRF is centered around the keypoint to orient all future computations.

3.1.2 Support Region Definition and Spatial Partitioning

After finding the LRF, a spherical support region is defined, having the same radius r, centered around the keypoint. All points within this radius are considered when computing the descriptor. Subsequently this support volume is divided into a 3D grid using three kinds of divisions; radial, azimuth and elevation divisions. The elevation divisions are oriented using the z-axis. The paper [19] mentions that their experimentations indicate that 32 spatial bins are fitting, which is obtained by 2 radial (the support region and a smaller radius too), 8 azimuth and 2 vertical (aligned with the x-y plane) divisions.

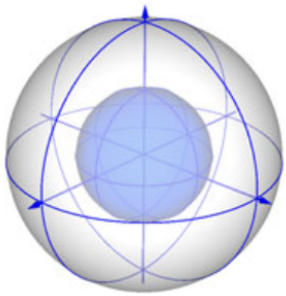


Figure 3: Support region divided into 16 bins (only 16, for visual clarity) [19]

.

3.1.3 Computing the Orientation Histograms

At this point, the support region consists of 32 spatial bins that will be analyzed one by one. The algorithm computes the angular differences between the normal of each neighboring point (points within the spatial bin) and the normal of the keypoint; the z-axis of the LRF. This captures local curvature and shape in a compact and rotation-invariant form. The local geometric signature is captured because the normal of the points change with surface shape. The angle is then quantized (using a function of the angles) into one of 11 orientation bins, representing angular intervals, which then correspond to an 11-way divided histogram. [1] The function of the angles is $\cos\theta_i$, chosen because it can be computed fast as the function value can be computed by simply taking the dot product of the two normals [19].

$$\cos\theta_i = n_p \cdot n_{q_i}$$

Where n_{q_i} is the normal of the given neighbor point and n_p is the z-axis of the LRF around the keypoint so the normal at the keypoint. Furthermore "an equally spaced binning on $\cos\theta_i$ is equivalent to a spatially varying binning on θ_i, whereby a coarser binning is created for directions close to the reference normal direction and a finer one for orthogonal directions"[2]. The resulting histogram of angles for each spatial bin captures the curvature of the surface in each direction. If the angles are grouped together, then the normals don't vary a lot which means the surface is flat and vice versa. The histogram of each bin is then added up into a single descriptor vector that has 32 (spatial bins) $\times$ 11 (orientation bins) = 352 dimensions.

[1] The choice of 11 bins is not justified in the paper but is likely a result of balancing descriptiveness and computational cost.

[2] [19] page 363 line 19-22

These are the feature vectors that can be compared to find matches.

3.2 Kabsch Algorithm

Kabsch algorithm is a method for aligning two sets of points in a three-dimensional space based on predetermined correspondences, such as the results of SHOT. We assume the point sets are already scaled consistently. If they weren't, then we would scale them using Procrustes Analysis [18].

Given two point clouds $\mathcal{X}$ and $\mathcal{Y}$, the algorithms objective is to align $\mathcal{X}$ to $\mathcal{Y}$ with some rotation $R \in \mathrm{SO}(3)$ [3] such that the sum of the squared distances between corresponding points is minimized [15]. To do this, Kabsch goes through the following steps:

Kabsch algorithm

1) Centroid calculation
2) Cross-covariance Matrix Calculation
3) Singular Value Decomposition
4) Optimal rotation matrix

Firstly, we calculate the centroids $C_{\mathcal{X}}$ and $C_{\mathcal{Y}}$ of $\mathcal{X}$ and $\mathcal{Y}$, respectively (see section 2.3).

We already know what a covariance matrix is and how to calculate it from section 2.3. Sometimes, the covariance between variables within one point cloud is not wanted, but rather between two point clouds. In this case, we call the resulting matrix the cross-covariance matrix:

$$H = \sum_{i=1}^{N} (x_i - C_{\mathcal{X}})^T (y_i - C_{\mathcal{Y}})$$

The cross-covariance matrix represents the relationship between the centered points of our pointsets, and each entry H_{ij} of the matrix represents how much the points vary along the i-th and j-th dimensions of $\mathcal{X}$ and $\mathcal{Y}$, respectively. High values indicate a strong linear relationship between the corresponding dimensions of $\mathcal{X}$ and $\mathcal{Y}$.

Next, we use SVD (see section 2.4) on our calculated covariance matrix, which we calculated in the previous step.

$$H = U\Sigma V^T$$

By decomposing our covariance matrix, we make it easier to find the optimal rotation matrix, since it provides a way to rotate our set of points $\mathcal{X}$ such that it aligns with the directions of maximum variance in $\mathcal{Y}$.

Lastly, one can show [16] that the optimal rotation matrix R can be computed using the orthogonal matrices U and V from SVD:

$$R = VU^T$$

The optimal rotation matrix is the matrix that aligns the centered points $\mathcal{X}$ with those of $\mathcal{Y}$. To ensure that the determinant of R is positive, since the contrary would indicate an improper rotation (mirroring), then we adjust V. This adjustment is the negation of the last row of V before recomputing R. Hence, the matrix R effectively transforms the coordinates of $\mathcal{X}$ such that the variance is maximized along the same axis as $\mathcal{Y}$.

[3] Special orthogonal matrices in three dimensions.

3.3 Iterative Closest Point (ICP) Algorithm

One of the applications of Kabsch Algorithm is in the Iterative Closest Point (ICP) Algorithm. ICP is the historically most used method for fine alignment and comes in lots of variations [17]. First introduced by [4], it is used to get a point to point correspondence when iterated. The idea is to minimize the squared differences between two point clouds $\mathcal{X}$ and $\mathcal{Y}$. The algorithm consists of three parts:

ICP algoritm

1) Find point correspondences.
2) Find rotation and translation that minimizes error.
3) If point clouds are closer, iterate.

Many different methods exist for seeking matches between points in different point clouds, for example by finding the closest point in the reference $\mathcal{Y}$ for each point in the scan point cloud $\mathcal{X}$. Other method include using information about each point such as color (in the case of scans which include this), curvature, normals etc. [17]. Later on, when using this algorithm, we will be using the discussed SHOT algorithm to match $x_i \in \mathcal{X}$ to $y_j \in \mathcal{Y}$.

Kabsch algorithm can thereafter be used to find the rotation $R \in \mathrm{SO}(3)$ and a translation $t \in \mathbb{R}^3$ such that the following squared difference is minimized:

$$\sum_{i=1}^{N} ||y_i - (Rx_i + t)||^2$$

where the translation is the difference in centroid between the rotated scanned point cloud and centroid of the reference.

As long as this error is reduced (and results in a value above some preset tolerance), this algorithm is repeated. Other variations include just setting a number of iterations to perform [16]. We will later build upon this algorithm when taken into the context of partial-to-partial alignment, as the standard version of ICP is often applied to full-to-full alignments.

The reason for the ICP to be called a "fine" alignment is the fact that it often requires two point clouds to be reasonably aligned already. Otherwise, one risks to find a local minimum when iteration the above equation rather than a global one [17].

3.4 Random Sample Consensus (RANSAC)

To find the reasonable alignment needed for ICP, the algorithm dubbed RANSAC, which stands for RANdom SAmple Consensus can be applied. It was first proposed in an 1981 paper as a way to fit a mathematical model to experimental data. Since real world observations are more than likely to contain noise and outliers, one of the main points was to be able to fit a model to the data without the gross outliers or “poisoned points” unduly influencing the model, resulting in a bad fitting [6]. RANSAC assumes that all of the data points are either inliers or outliers. Inliers are the subset of data points that can be explained by a model with a particular set of parameters, whereas outlier do not fit a model in any circumstance [21].

The algorithm has proved to be robust in datasets that contain significant amount of outliers, even in cases where outliers make up over 50% of the dataset [23]. The core idea behind RANSAC is to iteratively select a small random subset of the input data, treating these as hypothetical inliers,

and then determining how many other data points agree with the model based on the hypothetical inliers, within a set tolerance. Iterating over this process allows the algorithm to find the model that has the most inliers, called the consensus set. The model with the largest corresponding consensus set is considered to be the best fit for the data [21]. Summing up what has been stated, the algorithm consists of 4 main steps, which, in the perspective of 3D shape alignment, will be detailed below.

RANSAC algorithm

1) Select a random subset of data points
2) Find the model to this subset
3) Measure how many other points agree with the model
4) Repeat for a fixed number of iterations and choose the best model

3.4.1 RANSAC method

Rather than trying to fit a model to the whole dataset at once, RANSAC uses only as many initial points as necessary. Let us call the random subset of points, hypothetical inlier points, and use the abbreviation HIP from now on. The number of points contained in HIP is determined by the specific task at hand. For example, if we need to fit an arc of a circle to a dataset containing two-dimensional points, we would need to select three data points at random, since three points are required when determining a circle [6]. In 3D shape alignment the goal is to find the rigid transformation that best aligns related points via a rotation and a translation.

In order to uniquely solve for the rotation and translation here, we need at least three point correspondences, since each pair of points gives three equations and a rotation in three dimensions will have nine entries. From here, the translation can be found using the same three point. If one wishes to find the rotation and translation simutaniously, four points will be needed [23].

Finding the model depends completely on what type of dataset the algorithm is meant to be applied on, but in our case, the model is a rigid transformation as mentioned above. This can be done in various ways, for example by using Kabsch algorithm, see section 3.2.

The next step is to find all the data points in the whole dataset (HIP excluded) that match the model based on the points in HIP. In our case, this means we need to calculate the point cloud alignment. There are several approaches depending on whether correspondances are known beforehand or not. If there are established or hypothetical pairs of corresponding points, the inlies can be found by simply finding the Euclidian distance between the pair of points using the formula [20]:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \tag{1}$$

If the correspondences are unknown, a popular method to compare two point clouds is the Chamfer Distance (CD). This method measures bidirectional nearest-neighbor distances for the two point clouds, meaning it adds up the euclidian distance between nearest neighbour in point cloud $\mathcal{X}$ for all points in point cloud $\mathcal{Y}$ and vice versa. The formula is as follows [13]:

$$CD(\mathcal{X},\mathcal{Y}) = \sum_{x\in\mathcal{X}} \min_{y\in\mathcal{Y}} \|x-y\|_2^2 + \sum_{y\in\mathcal{Y}} \min_{x\in\mathcal{X}} \|y-x\|_2^2 \tag{2}$$

All the data points that fit the model within a predetermined margin of error are considered to be inliers with respect to the current model, making up what we will call the consensus set, CS for short [21].

3.4.2 Repeat and conclude

RANSAC is an iterative algorithm meaning that the same process is repeated. The best case scenario is that RANSAC returns a model where the cardinality of CS is the same as the cardinality of the entire dataset (HIP excluded), meaning all points are inliers with respect to the model. This result is hard to obtain because of the sheer number of points in the clouds. Moreover, false matches and points with no matches can occur. Especially symmetrical shapes pose a problem, since symmetries mean that the topology of the shape is identical in several points. Several proposals adressing this problem have surfaced [10].

One example is that the alignment needs to preserve lenghts within a single scan between the points in question (see Figure 4).

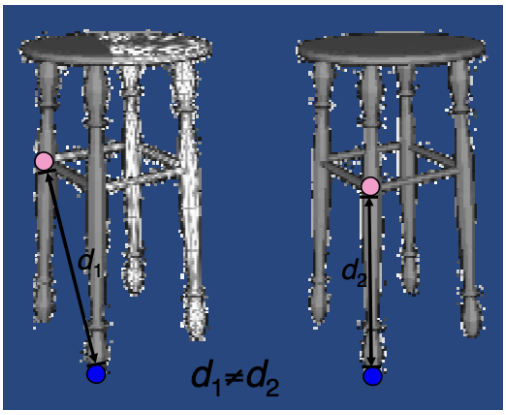


Figure 4: The distance between the blue and pink points in the scans are not equal, hence these blue and pink points cannot be matching points in this context [10]

This way, rather than immediatly finding an alignment, the algorithm can compare the distances between points from one set, and the hypothetically corresponding ones in the other set. If the distance is not the same in both point clouds then the two pairs of points are not corresponding. This allows the algorithm to terminate early, saving computing power and rooting out symmetry errors [10].

When running RANSAC, it will yield a model for every iteration. The task is now to determine which model is the best fit to the dataset as a whole. In the original paper from 1981 the best model is simply the one with the corresponding CS that has highest cardinality [6]. Hence at every iteration, if the current model results in a larger CS, it replaces the one previously saved. Since this algorithm is likely used on huge datasets and it is simply told to check random subset of n points, it will run as many times as there are n-combinations within the dataset. This number is way too large and thus iterating this many times is very costly. This means that the algorithm needs some kind of stopping criteria. There are three popular approaches, which can be combined if desired. Firsly, the algorithm can stop after a fixed maximum number of iterations. The risk with this approach is setting the number of iterations too low, and risking that all the worst hypothetical matches are the ones randomly selected, and so the algorithm never gets a chance to align based on good matches [23].

The second approach adresses this risk, since it relies on an estimated number of iterations needed to produce a satisfying model. It is based on the probability of only selecting inliers in the random

subset (HIP) selection. This number can be computed as such: Let p be the desired probablity that the algorithm provides at least one satisfying model after termination. Let w be the probability of choosing an inlier each time a data point is selected for HIP. This is rougly: $w = \frac{inliers}{allpoints}$. In many cases, w is not known a priori but a rough value can be given. In shape alignment, a minimum percentage of correspondance is often a criteria.

Now assuming that the n number of points needed to estimate the model are chosen independently, the probability that at least one of the n points is a outlier is: $1 - w^n$

This probability to the power of T (the number of iterations) is the probability that the algorithm never selects a set of n points which all are inliers. This is the same as $1 - p$ (the probability that the algorithm does not result in a successful model estimation), both the worst case scenario:

$$(1 - w^n)^T = 1 - p$$

Taking the logarithm on both sides to isolate T yields:

$$T = \frac{log(1 - p)}{log(1 - w^n)}$$

This is the estimated number of iterations that produce a useful model with the desired probability [21].

Lastly, the algorithm can terminate when the CS reaches a predetermined threshold, depending on the application and the dataset. In 3D alignment, this threshold would likely be a given percentage of inliers, i.e. a given correspondance between the two point clouds. This criteria is often combined with one of the first two, to allow the algorithm to terminate early if a satisfying model is found [21].

4 General problem

We will now be combining these pieces into a formulation of the general problem for partial-to-partial alignment. When trying to solve the nonlinear set of equations proposed in the section on the ICP, which we will build a bit upon, it turns out that it is possible to reduce them to a collection of linear equations. To get a closed solution to these linear equations, we will need some knowledge about the pseudoinverse of matrices.

4.1 The Pseudoinverse

Not all matrices are invertible. To generalize the idea of an inverse, we introduce the *pseudoinverse*.

Definition 3. *Let $A \in \mathbb{R}^{m\times n}$. The pseudoinverse (also called the Moore-Penrose inverse) $A^\dagger \in \mathbb{R}^{n\times m}$ is a linear transformation such that:*

i) $AA^\dagger A = A$

ii) $A^\dagger AA^\dagger = A^\dagger$

iii) Both $AA^\dagger$ and $A^\dagger A$ are symmetric

Note that if A is invertible, this $A^\dagger = A^{-1}$ meets the criteria in Definition 3. To find the pseudoinverse, we will use the SVD from earlier.

Theorem 5. *Let $A \in \mathbb{R}^{m\times n}$ and $A = U\Sigma V^T$ be the SVD of A. The pseudoinverse $A^\dagger$ is unique and is given by:*

$$A^\dagger = V\Sigma^\dagger U^T$$

where $\Sigma^\dagger \in \mathbb{R}^{n\times m}$ is the pseudoinverse of Σ where each non-zero entry in Σ is inverted.

Proof. We use Definition 3 and use U, V are orthogonal such that $V^{-1} = V^T$, $U^{-1} = U^T$.

$$\begin{aligned}
&[1] \quad AA^\dagger A &&= (U\Sigma V^T)(V\Sigma^\dagger U^T)(U\Sigma V^T) && \\
&&&= U\Sigma\Sigma^\dagger\Sigma V^T && \\
&&&= U\Sigma V^T = A && \textit{(i)} \text{ for } \Sigma \\
&[2] \quad A^\dagger AA^\dagger &&= (V\Sigma^\dagger U^T)(U\Sigma V^T)(V\Sigma^\dagger U^T) = A^\dagger && \textit{(ii)} \text{ for } \Sigma \\
&[3] \quad (AA^\dagger)^T &&= (U\Sigma\Sigma^\dagger U^T)^T = U(\Sigma\Sigma^\dagger)^T U^T = U\Sigma\Sigma^\dagger U^T = AA^\dagger && \textit{(iii)} \text{ for } \Sigma \\
&[4] \quad (A^\dagger A)^T &&= (V\Sigma^\dagger\Sigma V^T)^T = V\Sigma^\dagger\Sigma V^T = A^\dagger A && \textit{(iii)} \text{ for } \Sigma
\end{aligned}$$

To prove that the pseudoinverse is unique, suppose that two matrices B, C both satisfy the conditions in Definition 3 for A, meaning they are representations of $A^\dagger$. Then

$$\begin{aligned}
B &= BAB && \textit{(i)} \text{ for } B \\
&= (BA)^T B && \textit{(iii)} \text{ for } B \\
&= (BACA)^T B && \textit{(i)} \text{ for } C \\
&= (CA)^T (BA)^T B && \\
&= CABAB && \textit{(iii)} \text{ for } B \text{ and } C \\
&= CAB && \textit{(ii)} \text{ for } B
\end{aligned}$$

$$
\begin{aligned}
&= CACAB && \textit{(i)} \text{ for } C \\
&= C(AC)^T(AB)^T && \textit{(iii)} \text{ for } B \text{ and } C \\
&= C(ABAC)^T && \\
&= C(AC)^T && \textit{(i)} \text{ for } B \\
B &= C && \textit{(i)} \text{ and } \textit{(iii)} \text{ for } C
\end{aligned}
$$

□

Next, we will present an important property of the pseudoinverse. As will be shown, $A^\dagger$ is heavily connected to projections (or rather, orthogonal projections). From [1], we can consider any point $x \in \mathbb{R}^m$ and a subspace U and write it uniquely as $x = u + v$ where $u \in U$ and $v \in U^\perp$. We can think about a projection onto U as a surjective map $P : \mathbb{R}^m \to U, x \mapsto u$. The definition of such a orthogonal projection is:

Definition 4. *Let $P : \mathbb{R}^m \to \mathbb{R}^m$ be a linear transformation. Then P is an orthogonal projection if P is symmetric and idempotent ($P = P^2$).*

In the following theorem, we show that $AA^\dagger \in \mathbb{R}^{m\times m}$ is an orthogonal projection onto the range of A. Let $I \in \mathbb{R}^{m\times m}$ be the identity transformation.

Theorem 6. *$AA^\dagger$ is an orthogonal projection onto $R(A)$ and $I - AA^\dagger$ is an orthogonal projection onto $R(A)^\perp$.*

Proof. $AA^\dagger$ is symmetric by condition (*iii*) in Definition 3.

Moreover by condition (*ii*) in Definition 3, $AA^\dagger$ is idempotent:

$$(AA^\dagger)^2 = AA^\dagger AA^\dagger = AA^\dagger$$

What is left to show is that $R(A) = R(AA^\dagger)$. Because if this is true, then $AA^\dagger : \mathbb{R}^m \to R(AA^\dagger) = R(A)$ is indeed an orthogonal projection onto the range of $R(A)$.

Let $y \in R(AA^\dagger)$, then $AA^\dagger x = y$ for some $x \in \mathbb{R}^m$. But then $y = A(A^\dagger x)$ and $y \in R(A)$.

Let $z \in R(A)$. Then $z = Aw$ for some $w \in \mathbb{R}^n$. Hence $AA^\dagger z = AA^\dagger Aw = Aw = z$. Thus $z \in R(AA^\dagger)$ and $R(A) = R(AA^\dagger)$.

To show $I - AA^\dagger$ also is an orthogonal projection, take $x \in \mathbb{R}^m$. We can write $x = y + z$ where $y \in R(A)$ and $z \in R(A)^\perp$. Likewise, we can show that $R(I - AA^\dagger) = R(A)^\perp$:

$$
\begin{aligned}
(I - AA^\dagger)x &= x - AA^\dagger x = x - y = y + z - y = z \in R(A)^\perp \\
(I - AA^\dagger)z &= z - (AA^\dagger)z = z \in R(I - AA^\dagger)
\end{aligned}
$$

And using Definition 4, the transformation also turns out to be an orthogonal projection:

$$
\begin{aligned}
(I - AA^\dagger)^T &= I^T - (AA^\dagger)^T = I - AA^\dagger \\
(I - AA^\dagger)^2 &= I^2 - (AA^\dagger)^2 = I - AA^\dagger
\end{aligned}
$$

the transformation $I - AA^\dagger$ is the orthogonal projection onto $R(A)^\perp$. □

The use of the pseudoinverse can be seen in Lemma 7. When trying to solve the linear system $Ax = b$, then it is not always the case that b is in the range of A, i.e. that there exists one or more solutions. It turns out that the pseudoinverse can be used to find the best value for x such that the euclidean distance from Ax to b is minimized. This result is connected to the pseudoinverse, as it not only is a generalization of the inverse, but as shown in Theorem 6, it can also be used to project a point, say b, onto the point in $R(A)$ to which b is closest geometrically.

Lemma 7. *The minimum of* $||Ax - b||$ *is* $x_0 = A^\dagger b$.

Proof. Let $x \in \mathbb{R}^n$ and $x_0 = A^\dagger b$. Then:

$$Ax - b = Ax - AA^\dagger b + AA^\dagger b - b = A(x - x_0) + (I - AA^\dagger)(-b) \qquad (*)$$

Since $A(x-x_0) \in R(A)$ and $(I-AA^\dagger)(-b) \in R(A)^\perp$ (see Theorem 6), then they must be orthogonal vectors.

Thus, taking the norm and using Pythagorean theorem yields:

$$\begin{aligned} ||Ax - b||^2 &= ||A(x - x_0)||^2 + ||(I - AA^\dagger)(-b)||^2 \\ &= ||A(x - x_0)||^2 + ||Ax_0 - b||^2 \\ &\geq ||Ax_0 - b||^2 \end{aligned}$$

Therefore, the minimum of $||Ax - b||$ occurs exactly when $x = x_0$. □

4.2 Partial Alignment

We are trying to solve the problem of making partial alignment problems easier and faster. When trying to make a partial alignment between objects, many problems will appear depending on the method you use. Each method has its own strengths and weaknesses. We will therefore try and combine different methods. In this paper, we will use the SHOT-descriptors in combination with RANSAC and ICP.

Formally, we are given two point clouds $\mathcal{X} \in \mathbb{R}^{N\times 3}, \mathcal{Y} \in \mathbb{R}^{M\times 3}$ consisting of N and M points respectively. Here, $\mathcal{X}$ is the scanned point cloud and $\mathcal{Y}$ is the reference. We wish to transform $\mathcal{X}$ such that it aligns as well as possible with $\mathcal{Y}$, when only a subset of $N_0 < N$ points in $\mathcal{X}$ match with something in $\mathcal{Y}$.

Given these partial shapes, one wishes to find N_0 points (x_i, y_i) where $x_i, y_i \in \mathbb{R}^3$ are matching points on $\mathcal{X}$ and $\mathcal{Y}$, respectively. We will make use of the SHOT descriptors, as for each x_j and y_k, we can create SHOT descriptors:

$$d_{x_j} = \begin{pmatrix} d_{x_j}^{(1)} \\ \vdots \\ d_{x_j}^{(352)} \end{pmatrix}, \quad d_{y_k} = \begin{pmatrix} d_{y_k}^{(1)} \\ \vdots \\ d_{y_k}^{(352)} \end{pmatrix} \in \mathbb{R}^{352}$$

that are in $\mathbb{R}^{352}$ since each of the 32 bins are split up in 11 intervals based on the angle (see section 3.1). Now for each x_j, one can compute:

$$\arg\min_{y_k \in \mathcal{Y}} ||d_{x_j} - d_{y_k}||$$

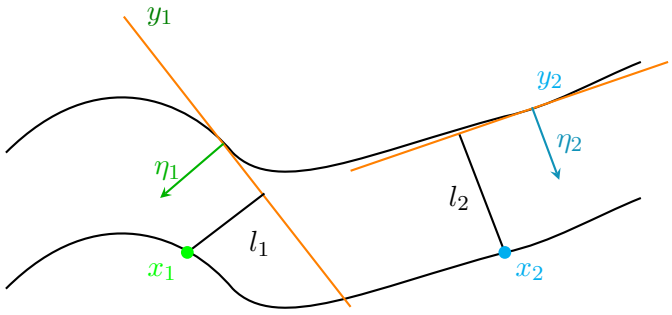
Now, matching each x_j to this y_k with repeats allowed gives N correspondences. Now, one can find the N_0 smallest pairs of $\min_{y_k \in \mathcal{Y}} ||d_{x_j} - d_{y_k}||$. These will mark the initial correspondences provided to the alignment process.

As mentioned in [12], a rough alignment is often needed to run ICP. For this purpose, we use the RANSAC algorithm. Only running on the N_0 matches limits the time needed for RANSAC to run.

Now with these correspondences and the rough alignment, we can gather a transformation consisting of a rotation matrix $R \in \mathrm{SO}(3)$ and translation vectors $t \in \mathbb{R}^3$ by solving the following optimization problem corresponding to the ICP problem:

$$R_{opt}, t_{opt} = \arg \min_{\substack{R \in \mathrm{SO}(3) \\ t \in \mathbb{R}^3}} \sum_{i=1}^{N_0} ([y_i - (Rx_i + t)]^T \cdot n_i)^2 \tag{3}$$

where n_i denotes the normal vector to y_i in unit size. As mentioned by [11], this minimization is often referred to as the *point-to-plane* method in literature. The additional term n comes in handy in the case a strict matching between each point does not exist (for instance when 3D-scans are made of two parts of a object, and each detection point on the data set does not align directly with a point on the model set). Here, the model is satisfied as long as the perpendicular distance between y_i and $Rx_i + t$ is minimized. The idea is illustrated in Figure 5.



$$l_i = (y_i - Rx_i - t)^T n_i$$

Figure 5: How point-to-plane is visualized for aligning to shapes in 2D.

Moreover, we have chosen to include a weigh w_i, which depends on how sure we are on that x_i and y_i match. Using theory, we will include this term to be more sure that we reach a global rather than a local minimum.

We now derive a solution for this equation. We note that because of the trigonometric terms in the rotation matrix R, the equations would become non-linear. Although the point-to-plane method generally converges faster than the traditional point-to-point [11], each iteration requires more computing power. Therefore, we observe that for small enough rotations (up to 30° according to [11]), we can for $\psi \approx 0$ approximate $sin(\psi) \approx \psi, cos(\psi) \approx 1$. Let α, β and γ denote the rotation

about the x, y and z axes, respectively. Thus, the rotation matrix can be written linearly as [11]:

$$R = \begin{pmatrix} \cos(\gamma)\cos(\beta) & -\sin(\gamma)\cos(\alpha)+\cos(\gamma)\sin(\beta)\sin(\alpha) & \sin(\gamma)\sin(\alpha)+\cos(\gamma)\sin(\beta)\cos(\alpha) \\ \sin(\gamma)\cos(\beta) & \cos(\gamma)\cos(\alpha)+\sin(\gamma)\sin(\beta)\sin(\alpha) & -\cos(\gamma)\sin(\alpha)+\sin(\gamma)\sin(\beta)\cos(\alpha) \\ -\sin(\beta) & \cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{pmatrix}$$
$$\approx \begin{pmatrix} 1 & -\gamma+\beta\alpha & \gamma\alpha+\beta \\ \gamma & 1+\gamma\beta\alpha & -\alpha+\gamma\beta \\ -\beta & \alpha & 1 \end{pmatrix} = \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix}$$

where each quadratic or cubic term has been approximated to 0. Using this as our rotation matrix in the i-th term in equation 3:

$$(y_i - (Rx_i + t))^T \cdot n_i \approx \left(\begin{pmatrix} y_{1i} \\ y_{2i} \\ y_{3i} \end{pmatrix} - \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \begin{pmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right)^T \begin{pmatrix} n_{xi} \\ n_{yi} \\ n_{zi} \end{pmatrix}$$

$$= \begin{pmatrix} y_{1i} - x_{1i} + \gamma x_{2i} - \beta x_{3i} - t_x \\ y_{2i} - \gamma x_{1i} - x_{2i} + \alpha x_{3i} - t_y \\ y_{3i} + \beta x_{1i} - \alpha x_{2i} - x_{3i} - t_z \end{pmatrix}^T \begin{pmatrix} n_{xi} \\ n_{yi} \\ n_{zi} \end{pmatrix}$$

$$= \alpha(x_{3i}n_{yi} - x_{2i}n_{zi}) + \beta(x_{1i}n_{zi} - x_{3i}n_{xi}) + \gamma(x_{2i}n_{xi} - x_{1i}n_{yi})$$

$$+ t_x(-n_{xi}) + t_y(-n_{yi}) + t_z(-n_{zi})$$

$$+ (y_{1i}n_{xi} - x_{1i}n_{xi} + y_{2i}n_{yi} - x_{2i}n_{yi} + y_{3i}n_{zi} - x_{3i}n_{zi})$$

Let $b[i] = -(y_{1i}n_{xi} - x_{1i}n_{xi} + y_{2i}n_{yi} - x_{2i}n_{yi} + y_{3i}n_{zi} - x_{3i}n_{zi})$, the i-th row of the matrix A be $A[i] = \begin{pmatrix} x_{3i}n_{yi} - x_{2i}n_{zi} & x_{1i}n_{zi} - x_{3i}n_{xi} & x_{2i}n_{xi} - x_{1i}n_{yi} & -n_{xi} & -n_{yi} & -n_{zi} \end{pmatrix}$ and $x = \begin{pmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{pmatrix}^T$. Then combining all N_0 terms into a matrix equation yeilds :

$$\min_{\substack{R \in SO(3) \\ t \in \mathbb{R}^3}} \sum_{i=1}^{N_0} ([y_i - (Rx_i + t)] \cdot \eta_i)^2 \approx \min_{x \in \mathbb{R}^6} ||Ax - b||^2$$

where solution x then gives the translation and rotation for R and t. We have therefore reduced the problem of least squares optimization problem. From Lemma 7, the solution to the equation on the right is $x = A^\dagger b$, where $A^\dagger = VD^\dagger U^T$.

As mentioned in [11], when implementing these optimal rotations α, β, γ into the rotation-matrix, note that the original matrix R is used rather than the approximated one.

5 Code development and experimentation

As mentioned, we want to develop a method for matching partial shapes only sharing a subset of their point clouds. We will be building upon the method in the GitHub repository implemented by Aubin Tchoi [2], which uses SHOT in combination with RANSAC and ICP.

The code starts by taking two files, which are the point cloud data. Since running the SHOT descriptors on all points can be very time consuming, a subsampling is performed be computing the descriptors.

Then a matching process is run on these descriptors, and for each point in the scan data set, its best partner in the reference set is found using the euclidean norm. To then find the transformation, RANSAC is used first to get a rough alignment.

Assuming the two point clouds now somewhat agree, ICP (both point-to-point and point-to-plane can be chosen) can be used with good hope of not reaching a local minimum.

The file returns the two updated point clouds as well as printing the overlap and inlier percentages.

Our improvements to this code can be found in the following GitHub repository [3].

5.1 Preexisting code

5.1.1 SHOT

The preexisting code [2] implements the SHOT descriptor based on the paper mentioned in section 3.1. The repository contains several implementations of the descriptor, the difference in them being the number of radii used. The first implementation uses the radius for computing the local reference frame as the support region for the SHOT descriptor, meaning that this is the spherical region that gets subdivided when computing the descriptor. This is the method originally presented in the SHOT-paper [19] and is the method we chose to use.

5.1.2 RANSAC

At this step, the code starts by randomly choosing a different subsets of correspondences to be tried, the subsets being the Hypothetical Inlier Points. The size of HIP for 3D alignment is 3, as this is enough to have one unique solution, but in this code it has been set to 4 different point pairs for each iteration because the algorithm finds the rotation and the translation simultaneously. In this code the author has chosen to run 10000 iterations.

A threshold is preset for what the maximum distance between two points can be, in order for them to be considered as inliers. After selecting the four random points in the scan point cloud, it finds the rigid transformation that aligns these points the best to their pairs in the reference point cloud. This outputs a matrix containing the rotation and translation.

The Consensus Set is found using equation (1) in 3.4.1 and the set inlier threshold. If there are more inliers than the previous best, this amount of inliers will be the new "best_inliers", and this transformation will be saved as the new "best_transformation".

It will then choose 4 new random points, and iterate the rest of the 10000 times. When all iterations have been run, we will be left with the transformation with the most amount of inliers throughout the iterations, which is out best "guess" at a rough alignment.

5.1.3 ICP

Within the code, the implementation follows the aforementioned algorithm in section 3.3. Firstly, it takes the scan point cloud and performs a new uniform supsampling of it. It uses an initial transformation found in the RANSAC algorithm and finds the points that, when transformed, lie within some distance of a point in the reference point cloud. This ensures that potential outliers will not be used in the ICP-algorithm. With a nearest neighbor-search, it finds the closest point to each of these inliers.

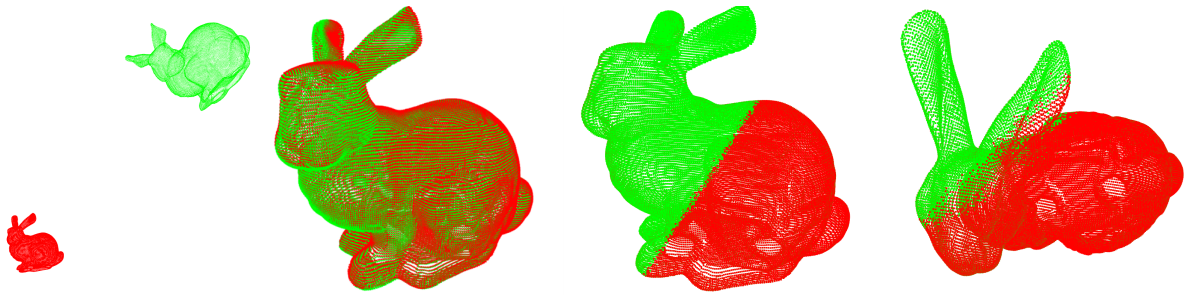
For each iteration, a rotation and translation can then be found to align these points with each other using normals for the points in the reference point cloud. It uses the method explained in section 4.2, where the nonlinear minimization problem is reduced to a linear minimization problem $||Ax - b||$.

The program lastly computes the error between the inliers and their corresponding pairs in the subsampled point clouds. A threshold is preset, which means that in the case where the point clouds are reasonably aligned, even if the maximum iterations is not reached, the loop breaks and stops the algorithm.

5.1.4 Tests

To evaluate the functionality of the preexisting code, we tested it on a 3D scan of the Stanford Bunny. The Stanford Bunny is a standard benchmark in 3D computer vision commonly used for assessing alignment and registration algorithms. Since rendering the two point clouds directly would result in two already aligned bunnies, we applied a transformation to one of them, here both a rotation and a translation (see Figure 6a), while fixing the other point cloud. This allowed us to test whether the code could correctly recover the transformation and realign the two point clouds.

We found that the code yielded highly satisfactory results, with an almost perfect alignment of the two point clouds. The results can be seen below.



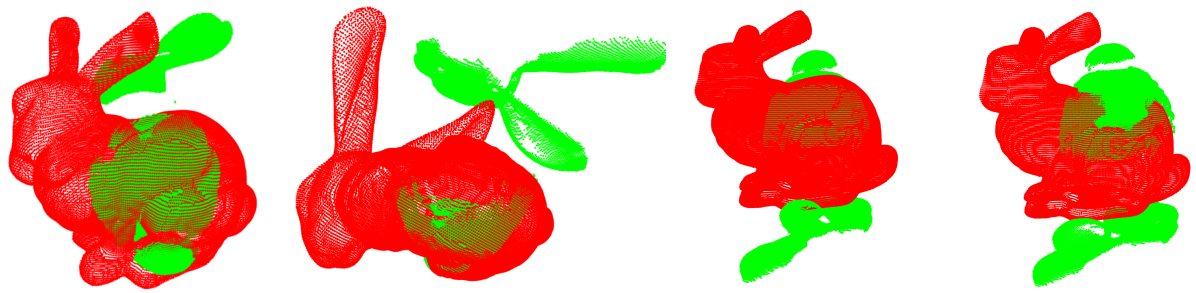
(a) (b) (c) (d)

Figure 6: a) Two identical Bunny models before alignment. b) Bunnies after rough alignment with RANSAC. c) and d) Bunnies after alignment with RANSAC and ICP.

After testing the codes functionality for full-to-full alignment, for which it was originally designed, we wanted to explore how it performed in partial-to-full and partial-to-partial alignment. While we didn't expect it to work in these cases, we wanted to verify this assumption in case it did.

We began by testing partial-to-full alignment, as it is conceptually closer to full-to-full alignment. In this test, we used a partial point cloud, which included the bunny's head and tail. We then

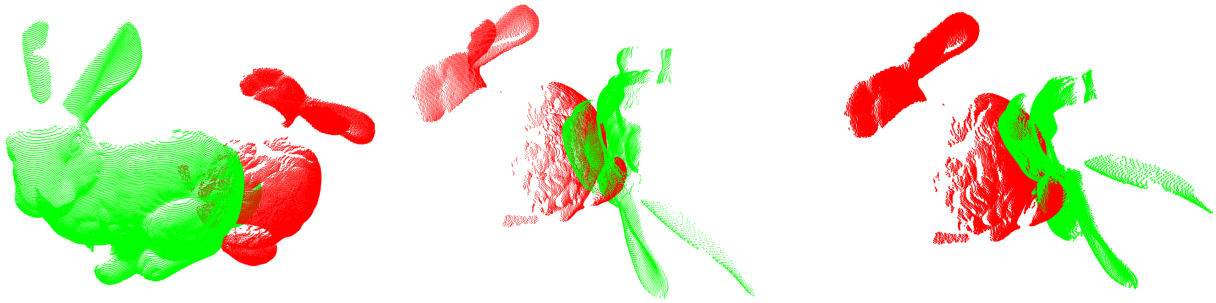
attempted to align this partial point cloud to the original bunny. The partial bunny was also rotated to simulate misalignment. As expected, the results showed that the code struggled in this scenario: when the shape to be aligned is incomplete and differs from the reference except for potential rigid transformations, the implementation of the alignment algorithms (RANSAC and ICP) failed to produce accurate results. The results can be seen below.



(a) (b) (c) (d)

Figure 7: a) and b) Different views of partial and full bunny model before alignment. c) Bunnies after rough alignment with RANSAC. d) Bunnies after alignment with RANSAC and ICP.

Finally, we tested the code on two partial point clouds of the Stanford Bunny, ensuring there was substantial overlap between them, so the algorithm, in theory could match up a subset of these points and align the two point clouds. Based on the results from the partial-to-full alignment tests, we already anticipated that the code would not perform well. After all, in the partial-to-full case, all the points in the partial bunny were also present in the full bunny, yet the code still failed to align them properly. In the partial-to-partial case, the challenge is even greater, as there are points present in one point cloud that are missing in the other and vice versa. The results of the partial-to-partial testing are shown below. Here we can see that while part of the tail aligns, it does so in the wrong orientation.



(a) (b) (c)

Figure 8: a) Partial point clouds before alignment. b) After rough alignment with RANSAC. c) After alignment with RANSAC and ICP.

5.2 Improvements and changes

5.2.1 Descriptor matching

When running the code, we noticed that the matching stage was somewhat lacking in accuracy when applied to partial shapes. The main problems were symmetries in the shapes and that the matching was not strict enough, meaning a lot of false matches were accepted by the algorithm. We addressed this issue by altering the matching of the descriptors such that only the matches with the highest confidence are used in the later steps of the alignment algorithm. Our method focuses on using the descriptors that are closest to each other, using a predetermined number of these (250 as default) as a minimum and the ones where the ratio between mutual distances are above a given threshold (0,7 as default).

The method starts by collecting all the nonzero descriptors. Then we arrange the distances between all the descriptors from the scan and reference shapes with the scan descriptor number as row indices and reference descriptor number as column indices. The entries of the matrix are the distances between the row index scan descriptor and the column index reference descriptor, as such:

ref $\rightarrow$ scan $\downarrow$	y_1	y_2	y_3	$\cdots$	y_M
x_1	d_{11}	d_{12}	d_{13}	$\cdots$	d_{1M}
x_2	d_{21}	d_{22}	d_{23}	$\cdots$	d_{2M}
x_3	d_{31}	d_{32}	d_{33}	$\cdots$	d_{3M}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
x_N	d_{N1}	d_{N2}	d_{N3}	$\cdots$	d_{NM}

From here a list, "indices", is made, each entry being the column index of the distance that is lowest in the row. Then another list, "distances", is made, each entry being the distance in the column that corresponds to the same entry in "indices".

Using all this information a list, "data_set", is made, consisting of 3-tuples: (x, y, d), with x being the row index, y being the column index retrieved from "indices" and d being the distance between at this entry in the matrix. When this list is made, the tuples are sorted so that the distances within them are in ascending order.

Now we create an empty list "trimmed_data_set" into which we insert the first 250 of the 3-tuples, since the tuples are in ascending order so the first 250 are the 250 best matched descriptors based on distance. From here we look to the next entry in "data_set" and check whether the ratio between the best and the current distance is larger than 0.7 (by default, can be changed). If the ratio satisfies the threshold we add it to the list; if not, we stop checking "data_set" since if one descriptor has too large a distance, the ones after it have even larger distances.

Then for all the descriptor matches in "trimmed_data_set", we look at the scan descriptor in the match and find the n (set to 3 by default) lowest distances to descriptors in the reference shape. This is done because there is a possibility that the best matches we found are still false because of symmetry, so we want more than one possible match for them in the reference shape when running RANSAC.

5.2.2 RANSAC

As mentioned, the code [2] performed poorly with partial-to-partial alignment. The starting theory was that if we can restrict RANSAC to run only on points that are in the shared, or so-called overlapping area, the alignment would be better, since the algorithm would run on points that actually have correspondences. The task was to identify these points and match them correctly.

As described in section 5.2.1, we have optimized the way we match the SHOT descriptors. We altered the input on which RANSAC was running, meaning that we restricted the points the algorithm could choose from to the set of points we got in the matching stage. Now, the random choice, the algorithm makes when selecting the hypothetical inlier points (HIP), was from an already checked subset of points, and so we ensured that only matches that had a chance at being meaningful were checked.

From here RANSAC was run, but we realized symmetry or ambiguity errors could still arise, meaning false matches. We addressed this problem by implementing "distance preservation", the idea presented in section 3.4.2. This means that if the distance between two points in HIP is not the same as the distance between the two points they are supposed to be matched with, then this match cannot be valid and the iteration terminates early. This does not mean that all matches in the iteration are wrong, just that at least one of them is.

After these two changes, RANSAC was set to run as normal; using Kabsch algorithm to find the rough alignment, maximizing the number of inliers for the alignment. Testing this implementation revealed that the alignment was quite random, which can be explained by the innate randomness of the algorithm. When lucky, a good subset is chosen as the HIP, resulting in a good alignment. In the worst case, all the worst subsets are chosen and so the algorithm terminates after the set number of iterations, even though the potentially best alignment is yet to be found.

We also found that a lot of the SHOT-descriptors only had zero-entries, meaning no information around the given keypoint could be encoded. To address this problem, we decreased the voxel sizes in the keypoint selection. Voxels are the 3D sections the point cloud is divided into, and a mean point is found in each voxel and used as a representative for all points within. This means that if the voxel size is bigger, more points get reduced to a single one. In reducing voxel size, we had more points to work with after subsampling, and so there were more points in each keypoint support radius to compute the descriptor on. This resulted in fewer zero-descriptors.

We also increased the threshold in the descriptor matching phase, since a lower threshold means fewer matches are accepted and RANSAC has fewer points to run on. Although initially it seemed counterproductive as the ratio between the matches and the inliers in the RANSAC models became smaller, we concluded that having more points as inliers seemed to be more effective in getting a better overall alignment.

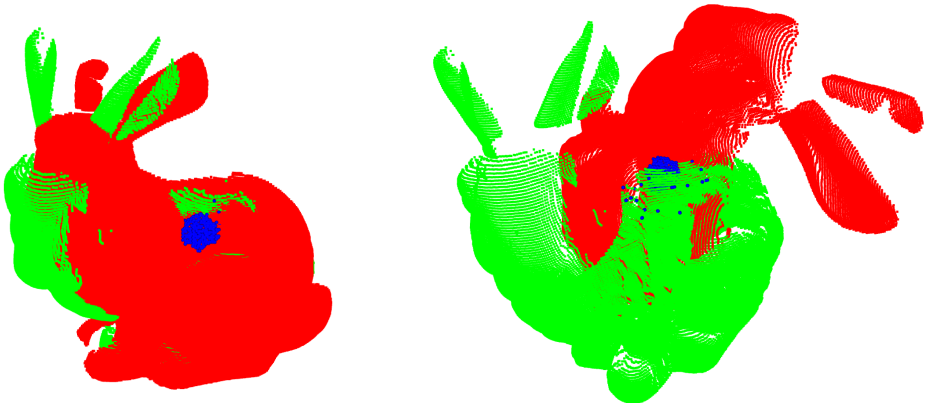
We first applied these changes to partial-to-full alignments, since the original algorithm was doing much better in these cases than with partial-to-partial. After getting satisfying results in the partial-to-full alignment, we started experimenting with partial-to-partial alignment. As can be seen in Figure 10a, the rough alignment was far better than the original, and so we accepted the outcome as is, and looked to see whether the alignment was good enough to run ICP on.

5.2.3 ICP

When running the code on partial alignments, we found that both with the original implementation and our eventual implementation of RANSAC, the later ICP run did not improve the alignment,

and often did not reach the threshold. We ascribe this to the fact that the programs uses the point-to-plane ICP on a uniform subsample of the point cloud, which does not take the partial overlap into account.

To improve the result of ICP, we tried different approaches to perform a subsampling. We first tried to use only the points found to be inliers when the best alignment is found in the RANSAC algorithm.



(a) (b)

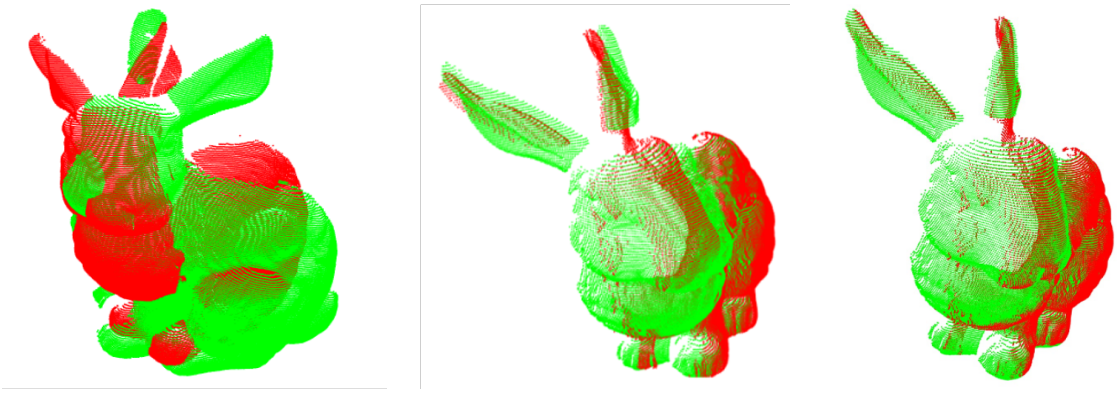
Figure 9: a) Alignment before ICP. Blue points are the inliers found in RANSAC. b) Alignment after ICP.

Figure 9 shows the point-to-plane ICP run on these subsamples. When comparing to the point clouds before ICP is used, the alignment worsens. We attribute this error to the minimal number of points used for the actual alignment (see the blue points in Figure 9). We can see that there should be a lot more points in the blue region (eg. the head), and when the program does not use those to align, it it easy to get a bad final result.

As a result of this, we sought to use a larger subsample of points in the alignment. We tried to perform ICP on all points within a certain radius of the aforementioned points, both in the scan and reference point clouds. Similarly, for several different configurations, satisfactory results were hard to come by. Again, our best assessment was that the points were not sufficiently spread out across the shapes, consequently only aligning those areas.

To address this, we referred back to our function which optimizes the matching of the descriptors (see section 5.2.1). To make sure only the best matches were used in the alignment, point-to-plane ICP was run on matches with a stricter threshold. The results can be seen in Figure 10b, where it is obvious a better alignment has been reached for partial-to-partial alignment with the updated ICP.

As a last optimization, a standard point-to-plane algorithm is run on the aligned shapes (after the aformentioned ICP has run). A point in the scan point cloud is included in this ICP run if the distance to the closest point in the reference point cloud is under some preset low threshold (set to 0.05 as standard). When running this at the end, we found that the misalignment that can be seen in 10bb at the ears is gone.



(a) (b) (c)

Figure 10: a) Alignment after improved RANSAC b) Alignment with our improved code (before last optimization of ICP) c) Alignment with our complete improved code

6 Discussion

6.1 RANSAC Iterations

Due to the time consumption of the implemented RANSAC algorithm, an upper bound of 1000 iterations has been set. If we were to use the formula from section 3.4.2: $T = \ln(1-p)/\ln(1-w^n)$ with a certainty of finding a useful model of $p = 0.9$, an inlier-ratio of $w = 0.1$ and $n = 4$ chosen points used for the RANSAC for each iteration, then $T = 23024$ of such would be needed. The reason for the relatively low inlier-ratio is the occurrence of many SHOT-descriptors being just the zero-vector. For future implementations, goal could be to get more non-zero descriptors. A direction for this study could be to implement other types of descriptors. Overall, due to the small sample of inliers, a suitable upper bound on the RANSAC implementation was hard to find.

We also considered the third termination method for RANSAC, but this one depends on setting a threshold for how big we want the Consensus Set to be. While this can work for full-to-full alignment, where we are sure all points in the scan shape have a match in the reference shape, it is more complicated for partial-to-partial alignments, since we don't know the overlap size.

When testing our implementation with respect to how much overlap between the partial shapes is needed, we concluded that a lower bound could be reached. We tried running the program on two bunnies with a less substantial overlap (where only the ears and parts of the face and back are shared) as shown in the figure below.

(a) (b)

Figure 11: The two partial shapes are the two sides of the bunny, only overlapping at the ears and some of the top of the back a) Two partial bunnies before alignment. b) The two partial bunnies after alignment with improved RANSAC.

As seen in Figure 11b, an alignment is found that seems to prioritize the alignment of the sides of the bunny even though it is opposite sides, and should not be aligned. The "symmetry" of the bunny seems to pose an issue, although precautions were taken in order to prevent those (see section 5.2.2). The issue here is that the descriptors of the points on the two sides are very alike, and there are no true matches for these points, so the "closest" descriptors are still the wrong ones.

Another reason the program fails in this setting is the vulnerability of RANSAC, as the use of random selection makes it harder to choose overlapping points with such a low real inlier-percentage.

6.2 Future work

The preexisting code [2] also implements something called bi-scale SHOT descriptors. The bi-scale computation uses a different radius for the SHOT descriptor calculation than for the LRF calculation. This second radius is the LRF radius multiplied by some number "phi" set by the user, but set to 3 as default. The goal of this approach is to make sure the LRF is computed in a sufficiently small neighborhood making it robust to noise that could be a problem if too much of the shape was within the radius. The goal with the larger SHOT-radius is to capture not only the finer details around the keypoint, but also global geometric information that might be ignored if the descriptor is computed on only the small radius of the LRF.

A problem with this approach is that it gets even more sensitive to the choice made by the user. Already, the choice of the LRF radius needs to be large enough to contain enough points to actually capture enough information (especially around flat or noisy areas), but not so large that it contains too much unrelated geometry. But now the choice of "phi" affects the descriptor too, meaning that if the radius used to compute the descriptor is too large, the algorithm will consider too many points that might just be noise.

When experimenting with the code we ran into the previously described problem; having too many zero-vectors as descriptors. We wanted to see if this is changed by which SHOT implementation is applied, and when applying the bi-scale SHOT descriptors we saw a decrease in the number of zero-descriptors. We are not sure why this is the case, maybe the fact that the bigger descriptor radius means more points are considered and so the LRF radius was not large enough to capture any information. Nonetheless, it is an interesting direction to work in, since we believe that the more nonzero descriptors we get, the more of the shape's information we capture and this will ultimately lead to better alignment.

Another interesting find is that when applying the first round of the ICP algorithm to improve the rough alignment, we found it necessary to work with the same points from the matching function as RANSAC (section 5.2.1). Although both the quantity and the method of using those points have been changed (ICP does not work with the exact correspondences between points found in matching, rather their geometric location), the fact that it does use the same points does hinder the algorithm somewhat. A point of future work would be a fitting subset of points that could effectively be used for matching in the ICP.

Lastly, we would like to be able to compare the alignments quantitatively instead of just visually. With our data, this could be done by taking the matches used in the first round of ICP and considering them as two point clouds: "inliers from scan" and "inliers from ref". Then the final transformation is applied to the point cloud containing the points from the scan. Afterwards, we compute the Chamfer Distance (see section 3.4.1). This shows how well the points thought to be inliers have been aligned.

Although, to definitively describe the effectiveness of the alignment, we would need to know the ground truth. To obtain this, we would need to produce the partial point clouds manually. Since we would have control over the subsets of points left in both point clouds, we would know the true inliers. An error metric will then be applied to those pairs, after the program have run and found the alignment. This method is not optimal for real world applications, since the overlap size is often unknown and is the very thing the algorithm is supposed to find.

7 Conclusion

In this paper, we proposed a model for solving partial-to-partial alignment. The idea built upon the SHOT descriptors mentioned in [19] to perform matching between points in the scan and reference point clouds. By prioritizing the closest N_0 matches for each point, we sought to take care of symmetries. Moreover, the matches were sorted by confidence. Using the matching, RANSAC used an internal distance correlation to make sure each iteration was performed on plausible pairs of matches. Lastly, ICP was developed with partial alignment in mind, by only taking the best matches into account.

In conclusion, we improved the preexisting code to perform better on partially aligning point clouds, especially for higher overlap. However, we found that for point clouds overlapping less, the algorithms seemed to align non-corresponding parts of the shapes, returning high overlap percentage, even though the matchings were incorrect.

We attribute these shortcomings to several factors. Mainly, the used implementation of SHOT descriptors gave a notably high percentage of zero-vectors, and points with these descriptors were therefore not put to use in the alignment. This created problems when estimating a suitable upper bound for iterations in RANSAC. Moreover, suitable candidates for matches with descriptors were not sufficiently spread out across the shape, meaning only a small set of points were used in the alignment. Lastly symmetry seemed to pose an issue as well, especially for shapes with low overlap percentage.

We found that bi-scale descriptors only produced non-zero feature vectors. For future work, implementing this to test for shapes with less partial matches could yield a better model. Moreover, comparisons with similar implementations of descriptors could be made in order to quantify the results.

References

[1] Anthony M. & Harvey M., (2012). *Linear Algebra - Concepts and Methods* (pp. 367-378). Cambridge.

[2] Aubin-Tchoi, *SHOT / FPFH Descriptors on 3D Point Clouds, GitHub repository*, `https://github.com/aubin-tchoi/shot-fpfh`, Accessed: May 9, 2025.

[3] BatDrag1 Authors: Jensen, C. M., Nagy, F., Olsen, O. Ø., Thomsen, A., and Vrevic, A. *Partial Shape Alignment Using SHOT Descriptors*, GitHub repository, `https://github.com/BatDrag1/Partial-Shape-Alignment-Using-SHOT-descriptors`, Accessed: May 27, 2025.

[4] Besl, P.J.; McKay, N.D., *A Method for Registration of 3-D Shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1992, 14(2), pp. 239–256, `https://graphics.stanford.edu/courses/cs164-09-spring/Handouts/paper_icp.pdf`, Accessed: May 5, 2025.

[5] Cansiz, S. **Covariance Matrix: Definition, Derivation and Applications**, Built In, 2025, `https://builtin.com/data-science/covariance-matrix`. Accessed: May 12, 2025.

[6] Fischler, and Bolles *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography* SRI International, published in: Communications of the ACM, Vol. 24, No. 6, June 1981, pp. 381–395, `https://dl.acm.org/doi/pdf/10.1145/358669.358692`, Accessed: May 19, 2025.

[7] Guangliang Chen. *Lecture 5: Singular Value Decomposition (SVD)*. San Jose State University, 2020, `https://www.sjsu.edu/faculty/guangliang.chen/Math253S20/lec5svd.pdf`. Accessed: April 29, 2025.

[8] Hopcroft, J E. and Kannan, R. *Foundations of Computer Science: Theory for the Information Age*, 2012, `https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/hopcroft-kannan-feb2012.pdf`. Accessed: May 5, 2025.

[9] JOUAV. *What is a Point Cloud?*, `https://www.jouav.com/blog/point-cloud.html`. Accessed April 29, 2025.

[10] Kazhdan, M. *Shape Matching for Model Alignment: 3D Scan Matching and Registration, Part I*. ICCV 2005 Short Course, Johns Hopkins University, 2005, `https://gfx.cs.princeton.edu/proj/iccv05_course/iccv05_matching.pdf`, Accessed May 19, 2025.

[11] Kok-Lim Low. *Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration*, Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill, February 2004. `https://www.researchgate.net/figure/Point-to-plane-error-between-two-surfaces_fig1_228571031`

[12] Krsek P., Stepanov D. *The Trimmed Iterative Closest Point algorithm*, `https://www.researchgate.net/publication/3974183_The_Trimmed_Iterative_Closest_Point_algorithm`, Accessed: May 13, 2025.

[13] Park Cheolhee Lab. *"Chamfer Distance"*, `https://parkcheolhee-lab.github.io/chamfer-distance/`, accessed May 19, 2025.

[14] Salvi, M. A., Ansaloni, R., and Cannella, F. *Point Cloud Matching Using Singular Value Decomposition*. ResearchGate, 2016, `https://www.researchgate.net/publication/303353734_Point_cloud_matching_using_singular_value_decomposition`. Accessed: April 29, 2025.

[15] Stackademic. *Understanding the Kabsch Algorithm: A Step-by-Step Guide with Python.* Published on Medium, https://blog.stackademic.com/understanding-the-kabsch-algorithm-a-step-by-step-guide-with-python-a20d7dec7646, Accessed: May 14, 2025.

[16] Stachniss C., (2021a, March 20). *ICP & Point Cloud Registration - Part 1: Known Data Asisciation & SVD*, Youtube, https://www.youtube.com/watch?v=dhzLQfDBx2Q Accessed: May 2, 2025.

[17] Stachniss C., (2021b, March 22). *ICP & Point Cloud Registration - Part 2: Unknown Data Asisciation & SVD*, Youtube, https://www.youtube.com/watch?v=dhzLQfDBx2Q Accessed: May 2, 2025.

[18] Stegmann M.B. & Gomez, D.D., (2002, March 6). *A Brief Introduction to Statistical Shape Alignment*, Informatics and Mathematical Modelling, DTU Accessed: May 19, 2025

[19] Tombari, F., Salti, S., Di Stefano, L. (2010). *Unique Signatures of Histograms for Local Surface Description* In: Daniilidis, K., Maragos, P., Paragios, N. (eds) Computer Vision – ECCV 2010. ECCV 2010. Lecture Notes in Computer Science, vol 6313. Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-15558-1_26, Accessed: April 19, 2025

[20] Wikipedia contributors, "Euclidean distance," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Euclidean_distance, Accessed: May 19, 2025.

[21] Wikipedia *Random Sample Consensus* Wikipedia, last edited on 22 November 2024, https://en.wikipedia.org/wiki/Random_sample_consensus Accessed: May 8, 2025

[22] Zhou F., Zhao Y., and Deng Y., *Partial-to-Partial Shape Matching with Geometric Consistency*, arXiv preprint arXiv:2404.12209, 2024, https://arxiv.org/abs/2404.12209, Accessed: April 29, 2025

[23] Zuliani M., *RANSAC for Dummies*, Technical Report, 2009. http://www.marcozuliani.com/docs/RANSAC4Dummies.pdf, Accessed: May 19, 2025.

A SVD of the covariance matrix

Theorem 8. *Let $\mathcal{X} \in \mathbb{R}^{N\times 3}$ be N point in 3D-space. Let $v \in \mathbb{R}^3$ be a normalized vector in the direction of the most variance of $\mathcal{X}$. Then v is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of $\mathcal{X}$.*

Proof. Note that $\mathcal{X}v = \begin{pmatrix} x_1 \cdot v & \cdots & x_N \cdot v \end{pmatrix}^T$ describe the length of the vectors of each point x_i projected onto the span of v. For the variance to be maximized along v, then the variance of the set $\{x_i \cdot v : i = 1, ..., N\}$ has to be maximized.

This is given by:

$$\begin{aligned} var(\mathcal{X}v) &= \frac{1}{n}(\mathcal{X}v)^T(\mathcal{X}v) \\ &= \frac{1}{n}v^T\mathcal{X}^T\mathcal{X}v \\ &= v^T H v \end{aligned}$$

where H is the covariance matrix of Pv. Note that since this is symmetric, one can use the spectral theorem to write $H = QDQ^T$ in an orthonormal basis of its eigenvectors q_i:

$$v^T H v = v^T Q D Q^T v = y^T D y$$

where $y = Q^T v$ and $D = \text{diag}(\lambda_1, ..., \lambda_n)$ is the diagonal matrix of the sorted eigenvalues. Since $||v|| = 1$ and $Q \in O(3)$, then $||y|| = 1$.

We have now shown that:

$$\arg\max_{v\in\mathbb{R}^3} \big(\text{var}(Pv)\big) = \arg\max_{v\in\mathbb{R}^3} v^T H v = \arg\max_{y\in\mathbb{R}^3} y^T D y$$

Since $\sum_{i=1}^n y_i^2 = ||y||^2 = 1$ and λ_1 is chosen to be the largest eigenvalue, we get:

$$y^T D y = \sum_{i=1}^n \lambda_i y_i^2 \leq \lambda_1 \cdot \sum_{i=1}^n y_i^2 = \lambda_1$$

for all $y \in \mathbb{R}^3$. This maximum is achieved exactly when $y = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^T$ We can then find v as:

$$y = Q^T v \quad \Leftrightarrow \quad v = Qy = q_1$$

where q_1 is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of $\mathcal{X}$. This then describes the direction of the most variance in the point cloud.

Similarly, the direction orthogonal to v with the most variance will be q_2, corresponding to the second highest eigenvalue, etc. □