BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF MATHEMATICS

ÚSTAV MATEMATIKY

COLOUR EXTENSION OF IMAGE FAST POINT FEATURE HISTOGRAM

ROZŠÍŘENÍ OBRAZOVÉHO DESKRIPTORU FAST POINT FEATURE HISTOGRAM O BAREVNOU INFORMACI

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Aleksander Markovsky**

AUTOR PRÁCE

SUPERVISOR — **Mgr. Jana Procházková, Ph.D.**

VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Mathematics
Student:	**Bc. Aleksander Markovsky**
Degree programm:	Mathematical Engineering
Branch:	no specialisation
Supervisor:	**Mgr. Jana Procházková, Ph.D.**
Academic year:	2023/24

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Colour extension of image Fast Point Feature Histogram

Brief Description:

Point cloud registration methods can be divided into two main categories. You can handle all the data directly with methods such as ICP and NDT. Alternatively, you can use a subset of significant point clouds – feature detection. We will concentrate on the latter approach in this thesis. It involves identifying points with large variability, achieved by analyzing the changes in normals in the surrounding area. Next, it is necessary to define this point by analyzing its neighborhood, which is a memory–intensive process.This work will use a technique called Fast Point Feature Histogram (FPFH) to describe the local area with three specific measurements. Additionally, the study will explore the possibility of enhancing this description with colour information to better characterise the surroundings. With this descriptor, the final scene composition is composed.

Master's Thesis goals:

1. The student will study the theoretical basis of the FPFH detector and its implementation. Then he/she will learn about the colour spaces that can be used to represent colour.
2. Implement the extended FPFH descriptor in the chosen programming language and test it on real data.
3. Evaluate the results and assess whether adding colour information helps to achieve better results in point cloud registration.

Recommended bibliography:

RUSU, R. B.; N. BLODOW a M. BEETZ. Fast Point Feature Histograms (FPFH) for 3D registration. In: 2009 IEEE International Conference on Robotics and Automation [online]. IEEE, 2009, s. 3212-3217 [cit. 2023-09-06]. ISBN 978-1-4244-2788-8. Dostupné z: doi:10.1109/ROBOT.2009.5152473.

RUSU, R. B., N. BLODOW, Z.C. MARTON a M. BEETZ. Aligning point cloud views using persistent feature histograms. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems [online]. IEEE, 2008, s. 3384-3391 [cit. 2023-09-06]. ISBN 978-1-4244-2057-5. Dostupné z: doi:10.1109/IROS.2008.4650967.

LEMMENS, M. Introduction to Pointcloudmetry: Point Clouds from Laser Scanning and Photogrammetry. UK: Whittles Publishing, 2023. ISBN 1849954798.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2023/24

In Brno,

L. S.

doc. Mgr. Petr Vašík, Ph.D. Director of the Institute	doc. Ing. Jiří Hlinka, Ph.D. FME dean

Abstrakt
Diplomová práce se zabývá rozšířením Fast Point Feature Histogramů o barevnou informaci za účelem vylepšení registrace mračen bodů. Popisuje proces registrace mračen bodů pomocí Iterative Closest Point, jakou roli v ní plní FPFH a diskutuje, jaké barevné prostory jsou vhodné pro registraci mračen bodů. Pro vyhodnocení přínosu zahrnutí barevné informace práce prezentuje implementaci ICP a FPFH algoritmů v Pythonu a navrhuje metodologii evaluace registrace mračen. Závěr práce je věnován diskuzi výsledků experimentů, které demonstrují, že použití barevné informace v registraci mračen snižuje počet iterací potřebných ke konvergenci ICP.

Summary
This thesis deals with the extension of Fast Point Feature Histogram with color aiming to improve point cloud registration. The thesis describes point cloud registration using Iterative Closest Point, what is the role of FPFH, and explores available color spaces that are suitable to be used for point cloud registration. In order to evaluate whether adding color information helps to achieve better results in point cloud registration a Python implementation of ICP and FPFH is provided together with a methodology to evaluate registration. The thesis concludes with a discussion of experiments demonstrating that the use of color decreases the number of iterations required for the convergence of ICP.

Klíčová slova
mračno bodů, Point Feature Histogram, Fast Point Feature Histogram, PFH, FPFH, 3D deskriptor, Iterative Closest Point, ICP, registrace mračen bodů, Python, hledání nejbližších sousedů, kd-strom, barevný prostor, 3D sken

Keywords
point cloud, Point Feature Histogram, Fast Point Feature Histogram, PFH, FPFH, 3D descriptor, Iterative Closest Point, ICP, point cloud registration, Python, nearest neighbor search, kd-tree, color space, 3D scan

MARKOVSKY, A. *Rozšíření obrazového deskriptoru Fast Point Feature Histogram o barevnou informaci*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2024. 65 s. Vedoucí Mgr. Jana Procházková, Ph.D.

Rozšířený abstrakt

3D modelování pomocí mračen bodů je technologie využívaná napříč technickými obory. Nachází aplikace v robotice, architektuře, výrobním inženýrství, medicíně, i v zábavním průmyslu. Umožnují mapování městských čtvrtí, architektonických památek, ale i poskytují precizní míry drobných součástek.

Výsledný model objektu se skládá z mnoha mračen bodů nasnímaných z různých pohledů. Tato mračna bodů je následně potřeba navzájem sesadit do jednoho výsledného mračna na základě prolínajících se částí dílčích mračen. Tato problematika se nazýva registrace mračen bodů. Algoritmus standardně využívaný k tomuto účelu, Iterative Closest Point (ICP), je ovšem citlivý na počáteční podmínky a na šum přítomný ve zdrojových mračnech. ICP v každé iteraci vytvoří množinu dvojic nejbližších bodů ze zdrojového a cílového mračna a následně na zdrojové mračno aplikuje transformaci minimalizující vzdálenost mezi jednotlivými dvojicemi bodů. Rychlost a kvalita sesazení se tedy odvíjí od kvality vyhledávání podobných bodů mezi mračny. Standardní verze ICP využívá pro vyhledání podobnosti souřadnice bodů a eukleidovskou vzdálenost mezi nimi. Současný výzkum se zabývá popisem bodů v mračnech nejen na základě souřadnic, ale i lokálních charakteristik okolí, nazývaných point feature representations nebo také deskriptory okolí. Nechť p_q je zkoumaný bod a $\mathcal{P}_q^k = \{p_1^k, p_2^k \dots p_n^k\}$ je množina bodů v okolí p_q. Body náležící okolí jsou určeny nerovností

$$\|p_i^k - p_q\| \leq d, \tag{1}$$

kde d je daná maximální vzdálenost a $\|\cdot\|$ je zvolená norma. Okolí může být též určeno maximálním počtem bodů k. Deskriptor okolí je pak definován jako vektorová funkce F popisující lokální geometrii $\mathcal{P}_q^k$ okolí p_q

$$F(p_q, \mathcal{P}_q^k) = \{x_1, x_2, x_3 \dots x_m\}, \tag{2}$$

kde $x_i, i \in \{1 \dots m\}$ značí hodnotu i-té dimenze vektoru deskriptoru.

R. B. Rusu ve své disertaci [37] popsal deskriptor jménem Point Feature Histogram (PFH) a jeho optimalizovanou variantu Fast Point Feature Histogram (FPFH), tyto deskriptory představují významný přínos a inspirovaly nespočet vědeckých prací rozvíjejících tuto myšlenku. Tyto deskriptory uvažují okolí bodu a v něm pro každou dvojici bodů určí úhly mezi body a jejich příslušnými normálami

$$\begin{aligned}
\alpha &= \arccos\left(v \cdot n_t\right), \\
\phi &= \arccos\left(n_s \cdot \frac{p_t - p_s}{\|p_t - p_s\|_2}\right), \\
\theta &= \arctan\left(\frac{w \cdot n_t}{n_s \cdot n_t}\right), \\
d &= \|p_t - p_s\|_2,
\end{aligned}$$

pro $\forall p_s, p_t \in \mathcal{P}_q^k$, kde n_s, n_t jsou příslušné normály, $v = (p_t - p_s) \times n_s$ a $w = n_s \times v$. Vztah mezi dvojicí bodů v okolí $\mathcal{P}_q^k$ je charakterizován čtveřicí $(\alpha, \phi, \theta, d)$. Z těchto čtveřic následně vzniká histogram charakterizující geometrii okolí bodu. Index histogramu pro danou čtveřici je určen následující rovnicí

$$idx = \sum_{i=1}^{n} step_i(f_i) \cdot \frac{1}{div_1} \prod_{j=1}^{i} div_j, \tag{3}$$

kde div_i počet podintervalů pro prvek f_i z čtveřice $(\alpha, \phi, \theta, d)$ a $step_i$ je schodová funkce se zvolenými hranicemi podintervalů $s_j \in [f_{min}, f_{max}], j \in \{1, 2, \ldots, div_i - 1\}$.

$$step_i(f) = \begin{cases} 0 & f_i < s_1, \\ 1 & s_1 < f_i \geq s_2, \\ \vdots & \\ div_i - 1 & f_i \geq s_{div_i - 1}. \end{cases} \tag{4}$$

Neboť je FPFH invariantní vůči translaci a rotaci bodů, hodí se pro hledání shodných bodů mezi jednotlivými mračny.

Cílem této práce je rozšířit Fast Point Feature Histogramy o barevnou informaci za účelem vylepšení registrace mračen bodů. Po úvodu do registrace mračen, deskriptorů a FPFH se tato práce zabývá barvou, její reprezentací a barevnými prostory RGB, XYZ, HSV a LAB.

Pro evaluaci přínosu barevné informace pro registraci mračen vznikla Python implementace ICP a variant Point Feature Histogramů. V implementaci byl kladen důraz na flexibilitu a rozšiřitelnost kódu, aby testovací skripty mohly být jednoduše doplněny o implementace jiných deskriptorů nebo registračních algoritmů.

Pro evaluaci použití barvy jako deskriptoru byly vybrány 4 datasety sestávající se z mračen snímaných z různých úhlů. Pro každý dataset bylo provedeno 5 experimentů. V každém experimentu byly sesazovány mračna pomocí ICP s různými kombinacemi deskriptoru FPFH a barev v reprezentacích RGB, HSV a LAB. Čtyři experimenty simulovaly sesazení mračen tak, že sesazované mračno bodů vzniklo úpravou cílového mračna. První experiment pouze pootočil cílové mračno, další tři navíc přidaly různé úrovně Gaussova šumu pro analýzu citlivosti deskriptoru na přítomnost šumu v datech. Páty experiment sledoval realistické sesazení dvou mračen naskenovaných z různých úhlů pohledu. U každého experimentu se vyhodnocoval počet iterací ICP nutný ke konvergenci a kvalita sesazení určená průměrem a rozptylem vzdáleností mezi dvojicemi bodů ze zdrojového a cílového mračna. U simulované registrace, kde byla dopředu známá translace a rotace, byla navíc vyhodnocována norma posunutí a rotace.

Simulovaná registrace bez šumu ukázala, že použití barvy podstatně snižuje počet iterací ICP potřebných ke konvergenci. V simulované registraci s přidaným šumem barva opět snížila počet iterací potřebných ke konvergenci. Z experimentů s šumem není zřejmé, že by nějaký barevný prostor přispíval k rychlejší konvergenci nebo kvalitnějšímu sesazení více, než ostatní. V realistické registraci se opět potvrdil trend, kdy zahrnutí barvy snížilo počet iterací ICP potřebných ke konvergenci. U realistické registrace byl větší průměr i rozptyl vzdáleností při použití barvy, než v klasickém ICP nebo ICP s FPFH.

Tato práce se zaměřovala na vyhodnocení použití barvy pro rozšíření deskriptoru FPFH pro registraci mračen pomocí ICP algoritmu. Z experimentů vyplynulo, že barevná informace je užitečný nástroj pro sesazování, zejména pro snížení celkového počtu iterací pro dosažení výsledku. Alternativní přístup k registraci mračen navíc před ICP provádí hrubé sesazení na základě klíčových bodů mračna. Navazující výzkum by tedy mohl rozšířit implementaci o hledání klíčových bodů na základě barvy a geometrických deskriptorů pro lepší počáteční odhad sesazení pro ICP.

I hereby declare that I have written my Master's Thesis on the topic of Colour extension of image Fast Point Feature Histogram under the supervision of Mgr. Jana Procházková, Ph.D. using literature listed in the bibliography section.

Bc. Aleksander Markovsky

I wish to express my earnest gratitude to my supervisor, Mgr. Jana Procházková, Ph.D., for excellent guidance, support and provided insight. Furthermore, I would like to thank my family and friends for being there whenever I needed them.

Bc. Aleksander Markovsky

Contents

Introduction

3D modeling using point clouds is a problem set, that spans through a range of seemingly disparate science fields and industries. It provides vision for robotic systems, and when paired with artificial intelligence, serves as a foundation for autonomous vehicle technology. Point clouds are particularly useful for terrain mapping, as they can be employed to model urban landscapes, create maps, analyze the earth's surface, assist with forestry management, and facilitate the digital preservation of cultural heritage. In architecture and construction, point clouds help to accelerate the work of engineers and architects. Manufacturing engineers can leverage the technology for precise measurements, thereby ensuring parts meet specifications and tolerances, while in the medical field point clouds can be used to create 3D models of tissues, assisting in research, diagnosis and surgical planning. Although less noble than the aforementioned, point clouds are also utilized in entertainment and media to construct immersive virtual worlds, turning what was a mere fantasy a couple of decades ago into video games and movies with scenery nearly indistinguishable from reality.

When creating a point cloud model, the resulting model is composed of multiple views. These views must be aligned based on their overlapping regions into a common coordinate system. This problem is called point set registration. However the captured scans are polluted with noise and the staple algorithm for point set registration, the Iterative Closest Point, is sensitive to initial parameters, prompting researchers to attempt to enhance the speed and quality of registration by using point feature representations. In his dissertation thesis [37] R. B. Rusu described Fast Point Feature Histogram, a point feature representation, that made a significant contribution to the state-of-the-art and inspired countless research papers.

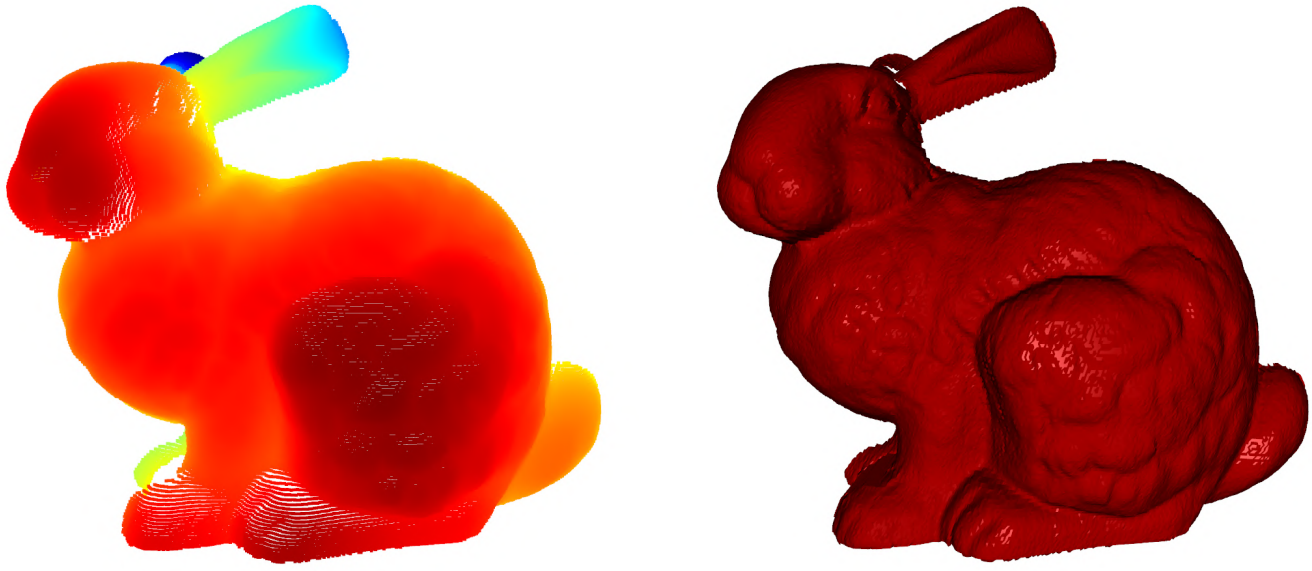
The thesis topic concerns the extension of the Fast Point Feature Histogram with color and the evaluation of its contribution to point cloud registration. The initial chapter introduces the concept of point clouds, how the data is sourced, and explains the point set registration process. Point feature representations are explained in the subsequent chapter, accompanied by the necessary context to calculate Fast Point Feature Histograms. The third chapter is devoted to an examination of color itself, presenting a theoretical framework for the consideration of color as a point feature representation. The final chapter discusses the Python implementation of point set registration using Fast Point Feature Histograms with a particular focus on the beneficial impact of using color.

1. Point clouds and their registration

The aim of this chapter is to introduce the fundamental concepts of 3D scene representation with point clouds and to discuss the data representation, capture, and processing. Specifically, it covers point clouds, methods of data acquisition such as laser scans or photogrammetry, space partitioning using a kd-tree data structure, and concludes with point set registration using the ICP variants.

1.1. Point cloud

One of the simplest ways of representing a 3D scene is via a point cloud. Point cloud is defined as a finite discrete set $\mathcal{P} = \{p_i \mid i = 1, \ldots, n\}$ of data points in space. The most basic representation is using a set of three Cartesian coordinates $(x, y, z) \in \mathbb{R}^3$ to represent a point. This representation can be further extended for a specific use case to include color information, a direction of a normal, volumetric data, etc. When processed, it is possible to create features and other statistics that describe a given point. For example, these features can describe local geometry such as surface curvature.



(a) Point cloud (b) Triangular mesh

Figure 1.1: Comparison of visualization between a point cloud and a triangular mesh of the same model.

Certain use cases demand a more complex structure. Visualization of point clouds is very well possible, but surface representation might be a better fit, as demonstrated in Figure 1.1. Surfaces can be represented with polygonal meshes or even simpler triangle meshes, which are the de facto standard due to their efficient representation. To represent a smooth surface a spline-based model such as NURBS (Non-uniform rational basis

spline) can be used. A combination of these structures can be used to create a 3D CAD (Computed Aided Design) model, which can, for example, contain information about a physical system such as in architectural software to calculate critical loads. There are techniques, such as the Delaunay triangulation, that allow to create a 3D surface from a point cloud. Although the subject matter of this thesis centers around point clouds, these representations and techniques are certainly interesting enough to study on their own.

1.2. Data capture

Before discussing the processing of point clouds, it is important to understand how the source data is obtained. A plethora of methods for data capture are available, each best suited for the specific use and required precision. These methods can be categorized based on the principle of data acquisition into contact methods and non-contact methods.

Contact methods utilize a physical probe that detects contact. This approach provides highly accurate results, but imposes a set of strict limitations to the scannable subject matter. Therefore, its use is often limited to assembly lines to ensure that the produced part has the correct size and is free of defects. Considering these limitations, this technological approach it is not be described further.

Non-contact methods are further categorized into active and passive. Active scanners use light or radiation to hit the measured object instead of a probe and then estimate the scene based on the information from the reflection of the emitted probe. Such scanners are based on Light Detection And Ranging (LIDAR), Laser Detection And Ranging (LADAR), ultrasound or an x-ray. Passive scanning uses available light instead of emitting it. Photogrammetry combines digital image capturing with projective geometry to capture 3D scenery. The simplest approach is stereophotogrammetry (not dissimilar to stereoscopy) using two cameras to estimate depth, while sophisticated implementations integrate multiple stationary cameras (as shown in Figure 1.2) or moving cameras.

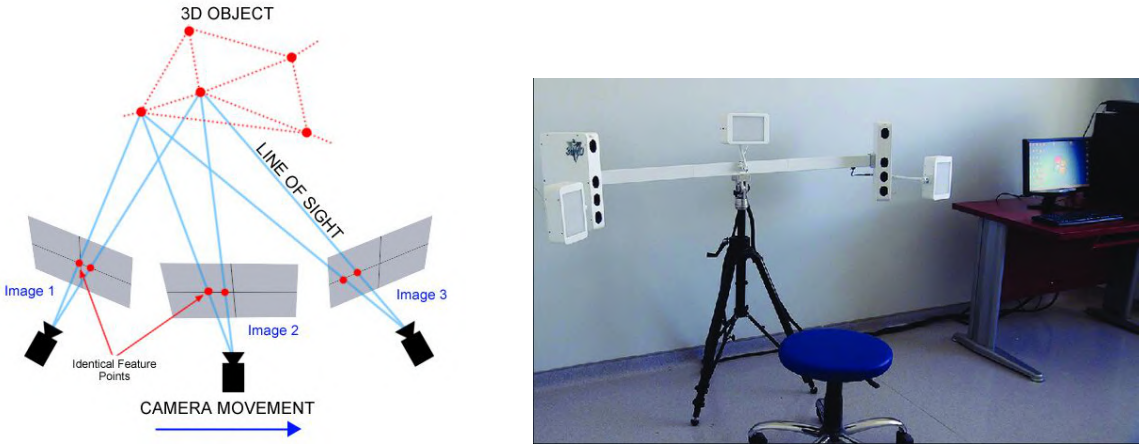


Figure 1.2: Illustration of photogrammetry principle on the left [30], photo of a stereophotogrammetry camera setup [23] on the right.

Another idea is to use the physical properties of camera lenses, namely their limited focus depth. By changing the focus distance and performing image analysis one can detect

how far are given parts of the image from the camera and reconstruct the scene this way. Figure 1.3 shows how the focus plane is affected by the distance of the subject.

Figure 1.3: Sharpness changes with distance.

Naturally, there exists a variety of hybrid approaches utilizing multiple technologies to achieve greater precision or to add different types of data (such as color) to the spatial information. Surveillance systems use RGBD (RGB Depth) cameras that are composed of a time-of-flight sensor and a regular surveillance camera to enhance the device's object recognition abilities. Since those systems' primary goal is not scene reconstruction, they are not described in further detail.

Time-of-flight

Time-of-flight (ToF) 3D scanning is a method that relies on the fundamental principle of measuring the travel time of light or laser pulses as they reflect off an object and return to the scanner. The distance to each point on the object is determined by calculating the time it takes for the light to travel to the object and back to the sensor since the speed of light is known, as indicated in Figure 1.4. The precise measurement of the time interval between the emission and detection of the light pulse is critical for accurately determining the distance to each point on the object's surface. This measurement is facilitated by high-speed electronics and timing systems. The time-of-flight data collected from numerous points across the object's surface is then used to construct a detailed 3D model.

The accuracy of ToF 3D scanning depends on several factors, including the precision of timing mechanisms in the scanning device, and the reflectivity and geometry of the object being scanned. Environmental factors such as lighting conditions and atmospheric disturbances can also impact the accuracy of the measurements.

They are widely used in industrial design, quality control, heritage conservation, virtual reality, and geographical mapping. These scanners offer several advantages over other 3D scanning technologies, including the ability to capture large volumes quickly and the potential for real-time data processing. However, they can be limited by their sensitivity to ambient light. Time-of-flight range finders are capable of operating over long distances

and thus are well suited for scanning architectural or geographic structures. The closer the measured object is the shorter the round-trip interval is which makes it harder to distinguish the points. This means the accuracy of measurements is comparatively low, in the order of millimeters.

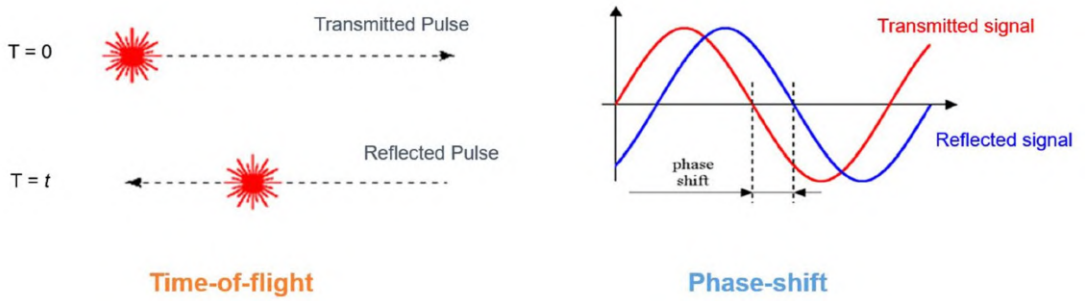


Figure 1.4: Time-of-flight schematic [47].

Triangulation

Triangulation is a technique that relies on the principles on geometry and trigonometry to determine the coordinates of scanned points in a similar fashion to stereophotogrammetry, but it replaces one camera with a laser projector. The laser projector emits a beam of light which is reflected back at the sensor. Depending on the distance of the measured surface from the projector the reflection lands at a different position on the center. The method is called triangulation because the projector, the camera, and the measured point form a triangle. Since the position of the camera and the projector is fixed, the distance between them is known and the angles at the camera and the projector vertices are known. These three pieces of information fully determine the triangle, thus using basic trigonometry we can calculate the distance from the camera. Figure 1.5 illustrates the geometric relationship between distance and image sensor.

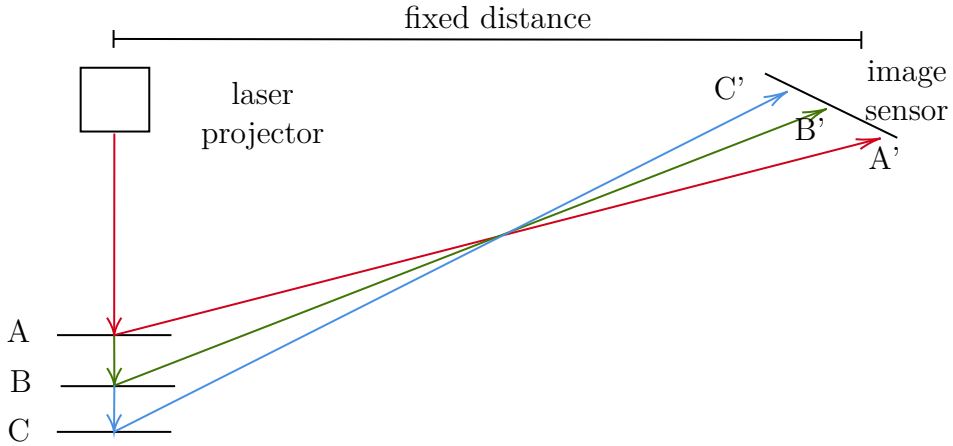


Figure 1.5: Triangulation scheme.

In practice, a laser stripe, its principle demonstrated in Figure 1.6, can be used to speed up the process of data capture. This technique is best suited for fine measurements

as the precision of triangulation scanners is in the order of tens of micrometers. This comes at a cost of a limited range of measurement.

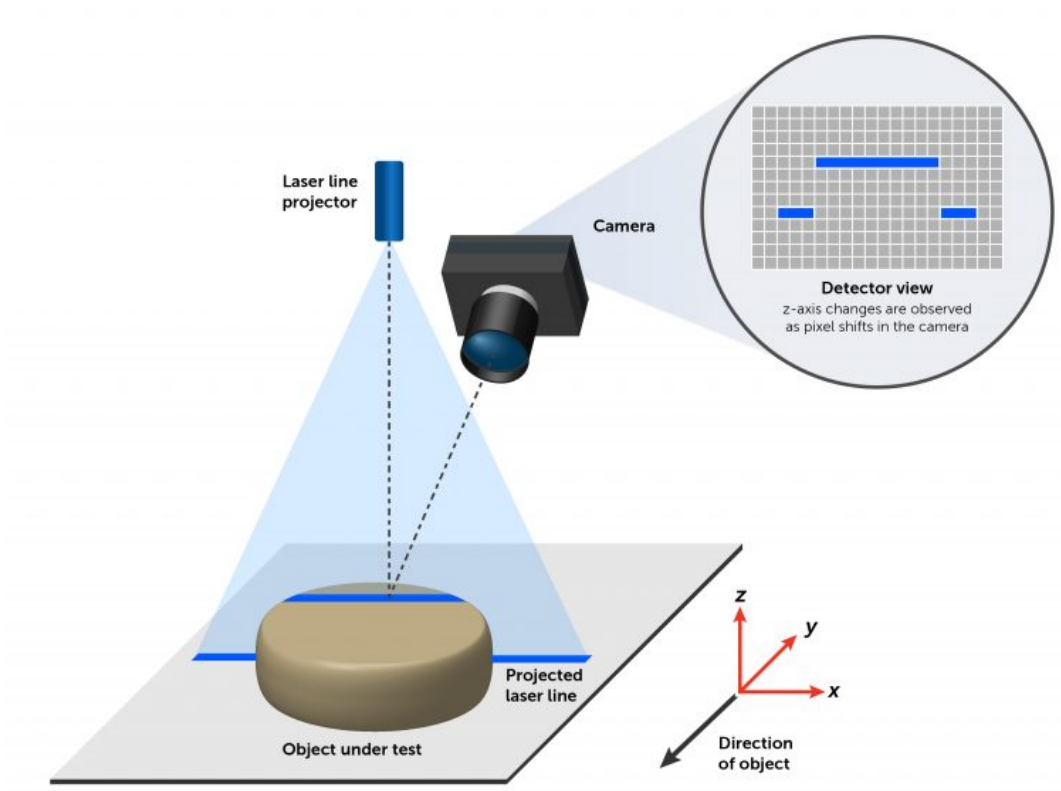


Figure 1.6: Using a line scan for triangulation [42].

Structured light

This approach can be considered a special use of triangulation. Structured light 3D scanners use light patterns projected onto a subject to measure its shape. These patterns, often created by an LCD projector or a similar light source, are distorted by the subject's surface. A camera, positioned near the projector, as indicated in Figure 1.7, captures the distorted patterns and they are used to calculate the distance of various points within its view. Reconstruction of the distances is often based on frequency analysis techniques like the Fourier transform or wavelet transforms, but more recently deep neural networks are trained for the determination of distances.

One of the main benefits of using structured light is speed and accuracy. They are capable of capturing multiple points or an entire scene simultaneously, rather than one point at a time. This capability significantly reduces motion distortion, as scanning a whole scene can be done in a fraction of a second. Some current models can even scan moving objects in real time. A downside is the fact that the projected light can interfere with other sources of illumination, this is the reason why infrared light sources are used often in modern implementations.

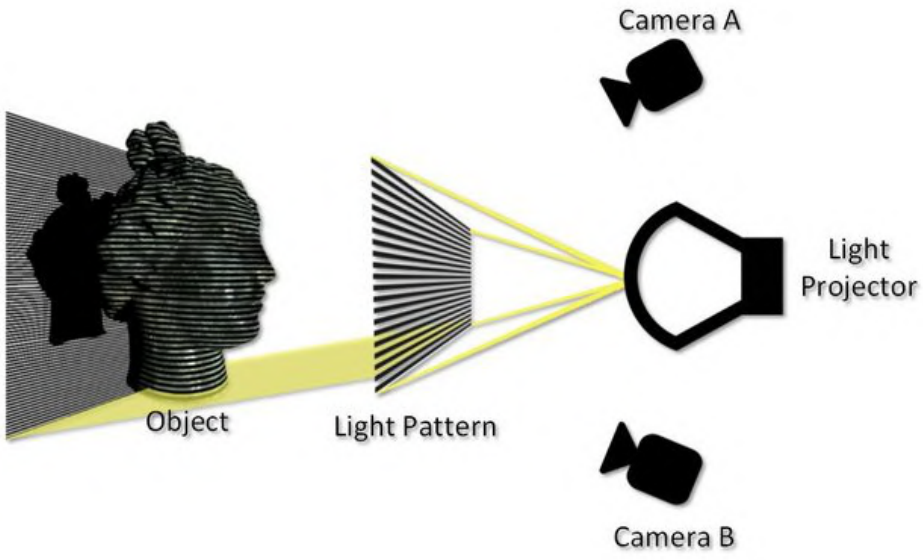


Figure 1.7: Using structured light for triangulation [36].

This technology is an evolving field, with significant academic interest and frequent research publications [14]. For instance, a real-time scanner employing digital fringe projection and phase-shifting techniques (variants of structured light methods) was created to capture and render detailed deformations of dynamic objects, like facial expressions, at 40 frames per second. Using defocusing techniques it is possible to push this frame limit even further.

A notable use of this approach is the early version of Microsoft Kinect, a gaming console extension that allows users to play games without the use of a controller, purely with the movement of their body. Another example is Face ID, the technology used in Apple devices for biometric authentication and also facial expression tracking. Coincidentally, both technologies were developed by the same company.

1.3. The nearest neighbor problem

The collected data comes as an unorganized set of triplets (x, y, z) in Euclidean space, frequently given by a matrix where the rows represent a data entry for a single point and the columns contain Euclidean coordinates and additional features. Such an approach for point cloud representation is intuitive and highly convenient as it allows algebraic operations and thus statistical analysis. A common problem when processing point clouds, such as when performing point set registration or creating a polygon mesh, requires the knowledge of which points are in the neighborhood of the analyzed point. The naive brute force approach iterates through a set of all points and compares their distances keeping track of the shortest distance. The Iterative Closest Point (ICP) algorithm's complexity is dominated by the complexity of nearest neighbor search and hence calls for a more efficient approach. That is where space partitioning methods become helpful. Commonly used space partitioning algorithms for point clouds are quadtree and octree (for 2D and 3D data respectively), kd-tree, and R-tree. While R-tree is high performance for 3D

data, its lookup performance rapidly deteriorates when going to higher dimensions [48]. Therefore kd-tree has been selected for further use in nearest neighbor search and is the focus of this section. To build the intuition on kd-tree the following paragraphs outline the key ideas of space partitioning with a regular grid using quadtree and octree.

1.3.1. Space partitioning using quadtree and octree

Quadtrees, shown in Figure 1.8, are an extension of the idea of binary search to two dimensional data. It is a special case of a tree graph where each internal node (one that does not contain leaves) has exactly four children. To create a quadtree select a starting point in the middle of the region. Divide the region into four quadrants, for each quadrant determine if there are at least two points present. In the positive case continue recursively dividing the subregion into four quadrants. In the negative case turn the quadrant node into a leaf containing one or zero points.

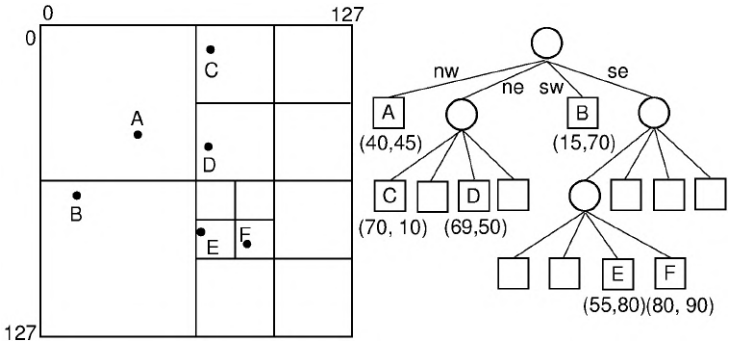


Figure 1.8: Partitioned region afig:quadtreend its tree representation. [33]

This way the quadtree encodes a sense of direction when performing a search for the nearest neighbor and significantly improves the average lookup time. The number of points in a leaf is sometimes referred to as bucket capacity and is an example of how a quadtree can be parametrized, similarly, the quadtree might have a set limit on depth. These parameters can be tuned to further improve lookup performance. Octree is a generalization of quadtree to three dimensions, recursively partitioning the point region into octants, as demonstrated in Figure 1.9.

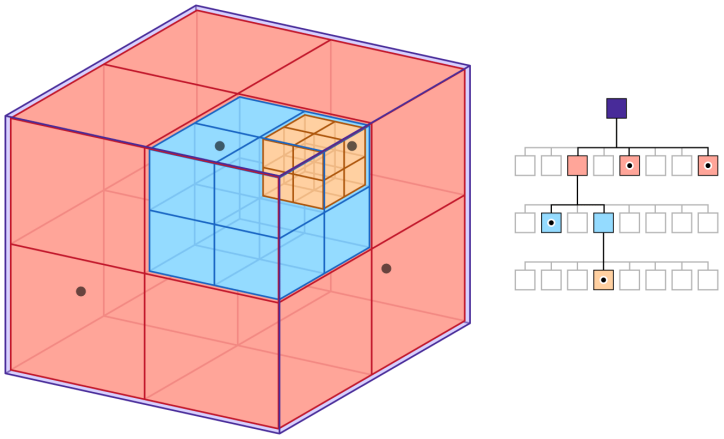


Figure 1.9: Octree space partitioning [1].

1.3.2. Space partitioning using kd-tree

The creation of quadtree and octree structures is fast due to the regularity of the grid it creates, however, the same regularity ultimately limits the lookup speed. Furthermore, if it were necessary to generalize this idea into further dimensions the number of children nodes would grow exponentially with the number of dimensions, making it rapidly more inefficient. Kd-tree elegantly circumvents both issues.

Kd-tree, sometimes spelled as kd-tree standing for k-dimensional tree, is a type of binary search tree. Each non-leaf node generates a splitting hyperplane, that splits the space into two subspaces. Nodes in the tree store information about the split. Nodes contain information about the split, the axis and the value, and pointers to the left and right child nodes. In case of the median splitting strategy the node also holds the splitting point. A leaf has child pointers empty and in the case the bucket capacity is larger than one it also contains a set of points belonging to the subregion.

Construction

A key feature of a kd-tree is the fact that the splitting hyperplanes are perpendicular to the space axis. In practice, this means when traversing through the tree it is sufficient to compare the point along a single dimension, for each level of the tree. The strategy of selecting the splitting hyperplane is not strictly defined and can serve as a parameter for performance tuning. The common approach is to cycle through all dimensions in an arbitrary order and choose the splitting hyperplane at the median point of the subspace. This leads to a balanced kd-tree, however, balanced trees do not always guarantee optimal performance.

To summarize, a step-by-step construction process:

1. **Select an axis.** Systematically cycle through all axes as the algorithm recurses.
2. **Find the median.** Sort the points along the chosen axis and select the median point. This point becomes a node in the tree.
3. **Split the points.** Points less or equal to the median are put in the left child node, points larger than the median are put in the right child node.
4. **Recurse.** Recursively apply previous steps on the point subset of each child node. Stop the recursion when there are no more points to split, a depth limit is reached, or a point threshold is reached.

The time complexity of building a kd-tree using the median split is $\mathcal{O}(N \log^2 N)$. High-performance kd-tree variants are able to lower the construction time complexity to $\mathcal{O}(N \log N)$ [46]. An example of how a space with a few points is partitioned and represented with a tree structure is shown in Figure 1.10.

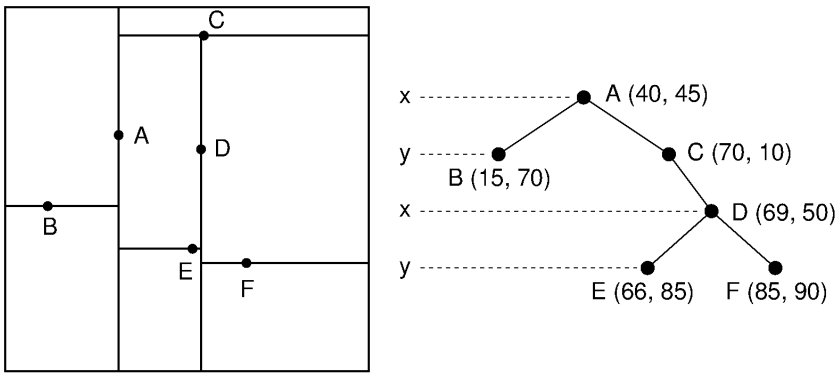


Figure 1.10: KDtree used to partition 2D space [32].

Point insertion and deletion

The addition of points is identical to any other search tree. It involves traversing the tree, finding the appropriate leaf and then adding it to the bucket of points or splitting the leaf further. Repeated addition can lead to an unbalanced tree, possibly hurting lookup performance.

Deleting a point is more computationally costly. It requires finding the appropriate node and reconstructing that part of the tree. That demands forming a set of points from all child nodes and recursively splitting it until a termination condition is satisfied.

Nearest neighbor search

The idea of the kd-tree nearest neighbor search is to first traverse down the tree until a leaf is reached, and a minimal distance in the leaf is noted. Then the tree is traversed back to the root node while updating the minimal distance by checking for minimal distances in the intersection of the splitting hyperplane with a hypersphere with the radius of the current minimal distance around the queried point. The process is formally described by a recursive Algorithm 1. The recursive search begins at the root of the tree with the initial closest distance set to the maximal value.

When kd-tree is used for nearest neighbor search it achieves the average time complexity of $\mathcal{O}(\log N)$ with the worst case complexity $\mathcal{O}(N)$ [16]. This is a significant improvement compared to the $\mathcal{O}(N)$ complexity of a naive approach. Furthermore, the there are computational savings. With naive search algorithm a full Euclidean distance must be computed between each point of the point cloud and the queried point, whereas with kd-tree lookup most comparisons are along a single dimension for the tree traversal. These savings become more noticeable as features are added to the point clouds and the dimensionality increases. Note that both approaches often keep track of squared distances, since Euclidean distance asks for a square root, which is a costly operation for a processor, unlike simple multiplication and addition.

This algorithm can be simply modified to perform k nearest neighbors search, simply by keeping a list of closest points instead of a single point. Another popular modification is the approximate nearest neighbors search [35] which provide a "good enough" result in a significantly shorter time. This approach is crucial in time-sensitive applications such as robotic navigation or real time graphics and is an active area of research [19].

Algorithm 1 Kd-tree nearest neighbor search

```
1: NNS(Q, root, root.point, ∞)
2: procedure NNS(Q, N, P, dist)
   ▷ Q - query point
   ▷ N - current node in the tree
   ▷ P - currently closest point
   ▷ dist - current closest distance
3:     if N is a leaf then
4:         localdist ⇐ ‖Q − N.point‖
5:         if localdist < dist then
6:             dist ⇐ localdist
7:             P ⇐ N.point
8:         end if
9:     else
10:        if Q(N.axis) < N.value then          ▷ first search left child node
11:            NNS(Q, N.left, P, dist)
12:            if Q(N.axis) + dist ≥ N.value then
13:                NNS(Q, N.right, P, dist)
14:            end if
15:        else                                 ▷ first search right child node
16:            NNS(Q, N.right, P, dist)
17:            if Q(N.axis) − dist ≤ N.value then
18:                NNS(Q, N.left, P, dist)
19:            end if
20:        end if
21:    end if
22: end procedure
```

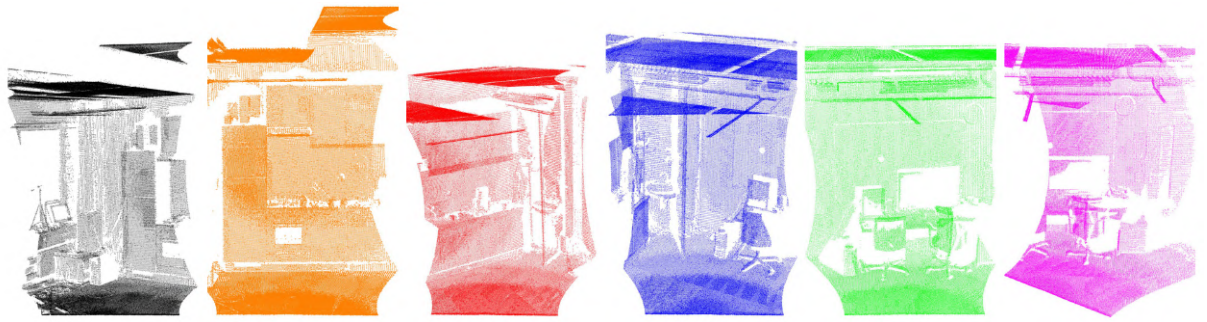
Note that each tree node X is a structure consisting of a pointer to the left node $X.left$, pointer to the right node $X.right$, coordinates of the splitting point $X.point$, index of the splitting axis $X.axis$ and a value $X.value$ by which the axis is split.

1.4. Point set registration

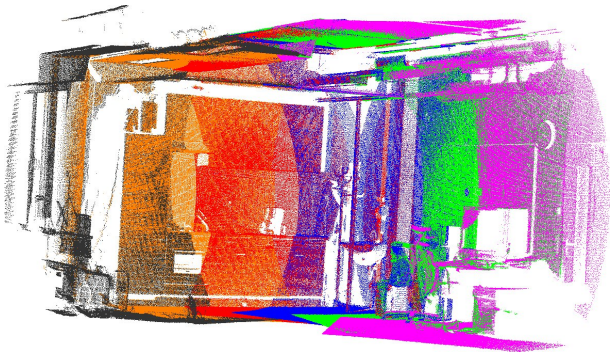
To create a comprehensive map of a scenery or even a single object it is often necessary to capture point cloud data from multiple viewpoints. This involves aligning multiple point clouds into a single dataset within a common reference frame, a process known as point set registration. Figure 1.11 shows an example of a processed scene.

Note that depending on the scanning circumstances, the transformation between viewpoints can be either rigid or non-rigid. The thesis assumes proper calibration of scanning devices, making the point set registration problem into a matter of finding a rigid transformation, a rotation and translation required to find the alignment of point clouds.

One approach to the registration problem is to perform it in two stages: coarse registration and fine registration. The Iterative Closest Point (ICP) algorithm and its variations are commonly used for fine registration. However, as ICP aims to optimize a highly non-convex function [47] and is thus not guaranteed to converge, it works best when point clouds are already somewhat aligned. This is the reason why coarse registration is first



(a) Scene from 6 different viewpoints [37].



(b) Result of registration [37].

Figure 1.11: Process of point cloud registration.

employed to roughly align the point clouds providing a starting point for the ICP-based fine registration to refine the alignment. Coarse alignment can be as simple as performing the Principle Component Analysis (PCA) and aligning the principle components, or it can be more sophisticated such as using Random Sample Consensus (RANSAC) to find a rough alignment without a significant influence of outliers.

Another approach, one that this thesis is focusing on, is to omit coarse registration and perform fine registration directly. For example, a moving robot scanning the scene multiple times per second is creating point clouds with such a small shift that does not impair the convergence of ICP. However the presence of noise or movement in the scene, such as moving leaves on a tree, might impair the process. In order to improve the robustness of the registration algorithm the point clouds can be matched not only on point coordinates but also on additional descriptors, such as Point Feature Histograms and color information.

1.4.1. Preprocessing

Several optional steps are available to further ensure the success of point set registration. In this section, the most commonly used ones are introduced.

Normalization

The registered point clouds may originate from different devices, or even the same devices with different calibration settings, therefore normalization of data might be necessary.

Filtering

Acquired point clouds might contain noise, and erroneous points that are not present in the scene as well. If possible, outliers should be detected and removed from the scene. Depending on the application it might be required to remove undesirable elements such as ground points, which are a set of planar points under the scanned subject. A simple pass-through filter might suffice or a more sophisticated segmentation-based method might be needed to separate the ground points from the main subject.

Downsampling

When data storage space is limited or there are processing time constraints, downsampling is useful. Downsampling is a process of reducing the amount of points present in the point cloud while preserving the detail. A variety of methods are described in detail in [26], while Figure 1.12 provides a visual summary of common approaches. The common approaches are:

- Random sampling removes a given percentage of points at random.
- Farthest Point Sampling (FPS): Selects points based on their maximum distance from previously selected points, ensuring a uniform distribution through the point cloud.
- Octree Decomposition: Uses space partitioning algorithm to find a representative point for each segment.
- Voxel Grid Filtering: Divides the point cloud into cells and selects a representative point within each cell.
- Feature Based Sampling: Attempts to preserve defining features, such as corners and edges.

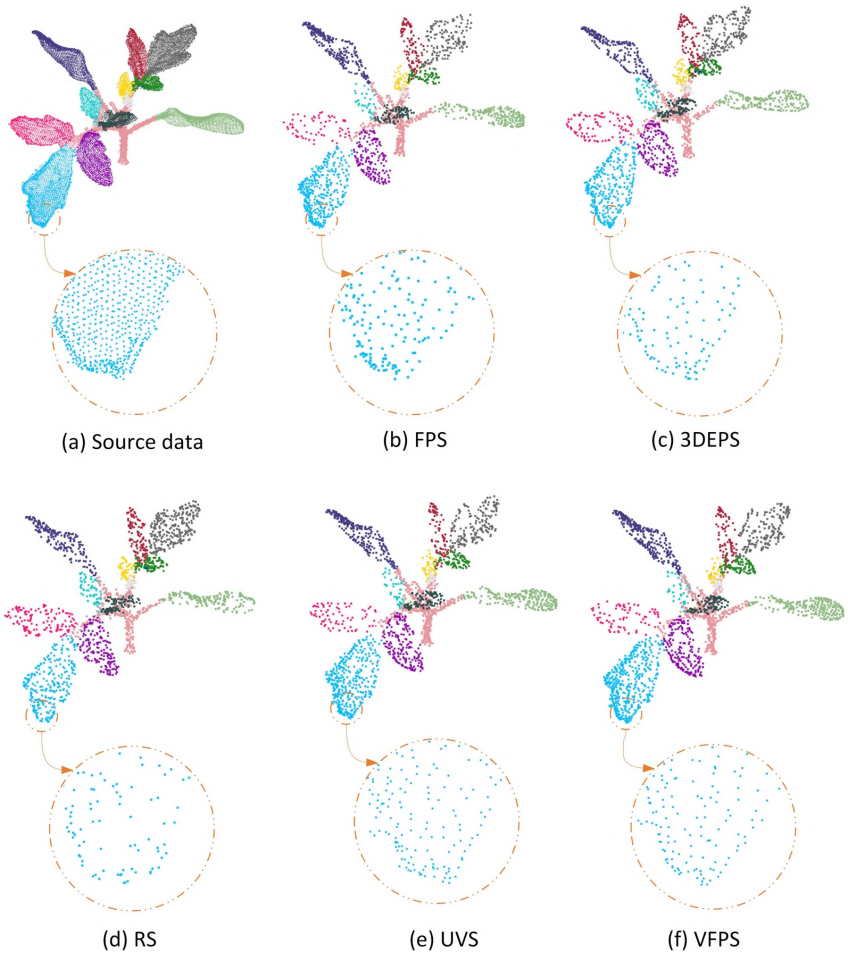


Figure 1.12: Differences between downsampling strategies. Source data contain 18521 points and each method downsampled to 2000 points. Compared methods are Farthest Point Sampling (FPS), 3D Edge-Preserving Sampling (3DEPS), Random sampling (RS), Uniformly Voxelized Sampling (UVS), Voxelized Farthest Point Sampling (VFPS) [26]

1.4.2. Fine registration - ICP

The Iterative Closest Point (ICP) algorithm constitutes a fundamental component of computational geometry and computer vision by solving the problem of finding an optimal rigid transformation to achieve alignment of two point clouds. It facilitates advancements in robotics, where reconstructing a scene enables path planning, and medical imaging, where bone surface models allow for computer-assisted surgical interventions [3].

The quick and precise registration of large-scale point clouds is a current focal area of research, with the introduction of ICP by Chen and Medioni [6], and Besl and McKay [5], presenting a significant advancement. This method computes the translation and rotation between matching points to achieve a specified level of accuracy, culminating in the determination of the transformation to facilitate the registration process. Nevertheless, the traditional ICP algorithm encounters challenges, notably in the selection of an initial value for the iterative process. The choice of this initial value critically influences the outcome of the registration. An inappropriate selection can result in convergence to a local optimum, preventing the algorithm from reaching the correct registration result. Only the most common variants are discussed further, namely point-to-point ICP and point-

to-plane ICP, however, Pomerleau et al. present a cohesive overview of other variants in [34].

The algorithm considers a reference point cloud and a matched point cloud, producing a translation and rotation necessary to achieve alignment. The following steps describe the ICP process:

1. **Pair matching.** For each point in the matched point cloud find the closest point in the reference point cloud.
2. **Estimate transformation.** Find the rotation and translation that minimizes an error function between the pair-matched subsets. When computing the transformation estimate, outliers may be filtered away or penalized using a weight function.
3. **Perform transformation.** Based on the previous step, apply the transformation to the entire matched point cloud.
4. **Compute the error metric.**
5. **Iterate.** Repeat from step 1 until the error metric is sufficiently small.

Pair matching

The pair matching procedure within the ICP algorithm is predicated on establishing correspondences between points across the source and target point clouds. The core idea behind pair matching is the assumption, that there is a subset of the reference and matched point clouds, that has been sampled from the same surface and thus there exist pairs sampling the same point in space in both point clouds. Naturally, it is impossible to sample the identical point twice, however the algorithm is robust enough that as long as both samples are close enough, they are going to be matched eventually.

For each point p_i in the source set P, the algorithm seeks to identify the closest point q_i in the target set Q, typically employing the Euclidean metric to measure distance, creating a set of point pairs $K(p_i, q_i)$.

Mathematically speaking,

$$q_i = \underset{q \in Q}{\operatorname{argmin}}\, d(p_i, q), \tag{1.1}$$

where $d(\cdot, \cdot)$ is a distance metric of choice. Kd-trees, as described in section 1.3.2, are particularly efficient for nearest neighbor search when the Euclidean metric is used. When creating the point pair set $K(p_i, q_i)$, the matched pairs are usually not exclusive, however certain ICP variants eliminate the point from search once it is selected.

To reflect the idea, that only parts of the point clouds are overlapping, the set K can be further restricted by applying a threshold to the distances:

$$\text{if } d(p_i, q_i) > \delta, \text{ then exclude the pair } (p_i, q_i). \tag{1.2}$$

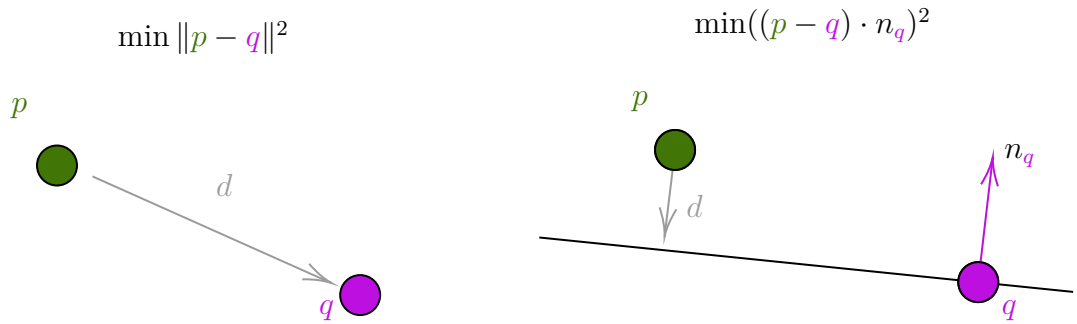
The value of δ can be arbitrary or selected based on a percentile of all pair distances in K, this approach being useful when an estimate of the overlap exists.

Estimating the transformation

When the pair correspondence set $K(p_i, q_i)$ is established, the next step is to determine the rigid transformation that best aligns set $\{p_i\}$ to set $\{q_i\}$. The transformation between the given sets is described as

$$q_i = Rp_i + t + e_i \tag{1.3}$$

where R is a rotation matrix, t is a translation vector and e_i is the additive noise vector. It encapsulates the measurement errors, sensor noise and numerical errors. The following analysis assumes the noise vector e_i to have zero-mean isotropic Gaussian distribution. The best alignment is a choice of rotation matrix $\hat{R}$ and translation vector $\hat{t}$ to minimize the error Σ^2. The error function choice determines the algorithm used to estimate the alignment of point pairs. The most common variants are the point-to-point and point-to-plane error functions, defined in Eq. (1.4) and Eq. (1.15) respectively, and illustrated in Figure 1.13. Note that the presence of noise is the limiting factor of precision of alignment. This means that even in the case $\Sigma^2 = 0$, the alignment might not be perfect.



(a) Point-to-point (b) Point-to-plane

Figure 1.13: Two distinct distance metric approaches.

Point-to-point ICP

The initial ICP proposal by Besl and McKay [5] used the point-to-point approach to calculating the quality of fit. It stems from the idea that both point clouds sample identical points that get matched together and their distance determines the error of fit. Point-to-point approach then uses Singular Value Decomposition (SVD) to determine the rotation and translation to create the best fit. The error function is defined as

$$\Sigma^2 = \sum_i \|q_i - (Rp_i + T)\|^2. \tag{1.4}$$

The initial step is to shift point sets into the origin. Let

$$\bar{p} = \frac{1}{k}\sum_{i}^{k} p_i, \tag{1.5}$$

$$\bar{q} = \frac{1}{k}\sum_{i}^{k} q_i, \tag{1.6}$$

$$p_i^* = p_i - \bar{p}, \tag{1.7}$$

$$q_i^* = q_i - \bar{q}, \tag{1.8}$$

where k is the number of point pairs, p_i^*,q_i^* are centered points and $\bar{p}$,$\bar{q}$ are respective centers of original point sets. Then the error function has a form

$$\Sigma^2 = \sum_i \|q_i^* - Rp_i^*\|^2. \tag{1.9}$$

The problem of finding optimal alignment is then solved in two steps:

1. Find $\hat{R}$ to minimize Σ^2.
2. Translation is determined by
$$\hat{t} = \bar{q} - \hat{R}\bar{p}. \tag{1.10}$$

The following steps describe how to find the rotation matrix $\hat{R}$ using an (SVD) algorithm.

1. calculate the covariance matrix
$$H = \sum_i p_i^* q_i^{*\top}. \tag{1.11}$$
2. Find the SVD of H
$$H = U\Sigma V^\top. \tag{1.12}$$
3. Calculate
$$X = VU^\top. \tag{1.13}$$
4. Calculate the determinant of X. If $\det(X) = 1$, then the rotation is found, $\hat{R} = X$. In a special case it might happen that $\det(X) = -1$. Then there are two possibilities:
   - Some of the singular values λ_i are zero. In that case step 3 is modified
   $$X = V'U^\top \tag{1.14}$$
   where V' is obtained from V by changing the sign of the i-th columns.
   - None of the singular values λ_i are zero. In that case the algorithm fails and a RANSAC-like method should be utilized instead.

A detailed analysis of this approach to finding the alignment transform including the derivation of the algorithms with proofs is available in [2].

Point-to-plane ICP

When Chen and Medioni [6] introduced their version of ICP, they opted for a different approach. Unlike the standard point-to-point ICP, which minimizes the Euclidean distance between corresponding points, the point-to-plane ICP minimizes the orthogonal distances from points in one set to the tangent planes at the corresponding points in another set. This approach is more aligned with the intuitive notion of surface fitting, making it especially suitable for applications in 3D scanning and modeling, where surfaces are often the primary objects of interest. However, this approach requires the knowledge of surface normals at each point. This is either provided by the scanner or can be estimated from the point's neighborhood. The following page is a summary of the main ideas of the algorithm derivation available in [34].

The error is formulated as

$$\Sigma^2 = \sum_i \left\| (Rp_i + t - q_i)^\top \cdot n_{q_i} \right\|^2 \tag{1.15}$$

where n_{q_i} is the surface normal of the point cloud Q at q_i.

The following formulas describe a solution for a 3D point cloud. The rotation matrix R has a form

$$R(\alpha,\beta,\gamma) = \begin{bmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha & \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha & -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{bmatrix}$$

The solution of alignment relies on the linearization of the rotation matrix that can be achieved using the small-angle approximation. When an angle $\theta \approx 0$, then $\sin\theta \approx \theta$ and $\cos\theta \approx 0$,

$$R(\alpha,\beta,\gamma) \approx \begin{bmatrix} 1 & -\alpha\beta - \gamma & \alpha\gamma + \beta \\ \gamma & \alpha\beta\gamma + 1 & \beta\gamma - \alpha \\ -\beta & \alpha & 1 \end{bmatrix} \tag{1.16}$$

$$\approx \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix}. \tag{1.17}$$

The full transformation is then represented as

$$\tau = \begin{bmatrix} r \\ t \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix}. \tag{1.18}$$

Thanks to the linearization the point-to-plane error function can be rewritten as

$$\Sigma^2 = \sum_i \| r \cdot (p_i \times n_{q_i}) + t \cdot n_{q_i} - (q_i - p_i) \cdot n_{q_i} \|^2. \tag{1.19}$$

Minimization of the error function with respect to r and T is achieved by setting the respective partial derivatives to zero, which can be then assembled as a linear form $A\tau = b$ as

$$\sum_i \begin{bmatrix} p_i \times n_{q_i} \\ n_{q_i} \end{bmatrix} \begin{bmatrix} (p_i \times n_{q_i})^\top & n_{q_i}^\top \end{bmatrix} \tau = \sum_i \begin{bmatrix} p_i \times n_{q_i} \\ n_{q_i} \end{bmatrix} ((q_i - p_i) \cdot n_{q_i}). \tag{1.20}$$

Finally by rewriting the linear form as

$$GG^\top \tau = Gh, \tag{1.21}$$

where

$$G = \begin{bmatrix} \cdots & p_i \times n_{q_i} & \cdots \\ & n_{q_i} & \end{bmatrix}, \tag{1.22}$$

$$h = \begin{bmatrix} \vdots \\ (q_i - p_i) \cdot n_{q_i} \\ \vdots \end{bmatrix}. \tag{1.23}$$

This equation can be solved using the Cholesky decomposition.

2. Point feature representations

A point is a fundamental unit of spatial information representation, natively defined by its three Cartesian coordinates in 3D space. However, on its own, the three coordinates provide very little context about local properties of the surface. This limits applications concerned with point comparison. For example, two points might have the same coordinates, but they come from different surfaces, or on the contrary, the two points might come from the same surface, but due to noise their coordinates differ. Without further analysis, it is difficult to reason about the relationship of points. These applications benefit from the concept of point descriptors. Academic resources offer a wealth of terminology for this concept, including terms such as shape descriptor, local descriptor, or geometric feature, however, to stay consistent with the primary source [37] for this chapter, the preferred terminology is point feature representation.

This chapter discusses the tools that help improve point set registration. It begins with a formalization of point feature representations, explains how to estimate the normals of a point cloud and then moves on to the topic of Point Feature Histograms and Fast Point Feature Histograms.

2.1. Formalization and properties of feature representations

The formal definition of a point feature representation as given in [37] is the following. Let p_q be the query point and $\mathcal{P}_q^k = \{p_1^k, p_2^k \dots p_n^k\}$ be a set of points in the neighborhood of p_q. The points belonging neighborhood must satisfy

$$\|p_i^k - p_q\| \leq d \tag{2.1}$$

where d is a specified maximum distance to the query point and $\|\cdot\|$ is a norm of choice. Furthermore, the restriction of the neighborhood can be given by the number of points k. A point feature representation is then defined as a vector function F that describes the local geometry captured by the neighborhood $\mathcal{P}_q^k$ around the query point p_q

$$F(p_q, \mathcal{P}_q^k) = \{x_1, x_2, x_3 \dots x_m\} \tag{2.2}$$

where $x_i, i \in \{1 \dots m\}$ stands for the value of the ith dimension of the point feature representation vector. Comparison of different points p_1 and p_2 then translates to the comparison of their point feature vectors F_1 and F_2. Let Γ be a similarity measure, then the similarity of points p_1 and p_2 is described as

$$\Gamma(p_1, p_2) = d(F_1, F_2) \tag{2.3}$$

where d is a distance metric.

In simple terms, a point feature representation, or a local shape descriptor, is a vector signature at a point describing the local environment. Trivial example of a point feature representation is a surface normal at a point, or a point average in a ball-shaped neighborhood with a fixed radius. A more sophisticated point feature representation includes information about surface curvature or detects the presence of an edge. With a wide

variety of approaches, there is a necessity to establish what constitutes a "good" or "bad" point feature representation. According to [24, 37] it is useful to evaluate the following characteristics.

1. **Discrimination accuracy:** The primary goal of point feature representations is to improve the capability to distinguish different points and match the same points in the point cloud
2. **Transformation invariance:** Cartesian coordinates change completely with rotation and translation, therefore having a point signature not be affected by transformation allows to use it as an identifier for matching.
3. **Varying sampling density:** A local surface patch should have the same signature regardless of the amount of points sampled in the area
4. **Robustness against model degeneracies**
5. **Uniqueness**
6. **Performance and memory efficiency**
7. **Ability to do partial matching**
8. **Noise insensitivity:** Small changes in the local topology lead to small changes in the point feature representation

The literature is abundant with approaches to point feature representations. A comparison and a taxonomy of state-of-the-art descriptors is provided in [17]. The following paragraphs present a selection point feature representation solutions.

- View-based descriptors build upon decades of research in the field of image processing. The general idea is to have several fixed viewpoints and project an image of the point cloud's surface onto a 2D plane. This way a scanned object is represented by a series of images and is mostly used for object recognition and matching. An overview is presented in [27].
- Transform-based descriptors embed the information about local topology using Fourier transform or wavelet transforms. An example of such a descriptor is RIDF, described in detail in [20].
- Geometry-based descriptors use a more direct approach. Radius-based Surface Descriptor (RSD) [29] models radius as a relationship of the distance between two points within the neighborhood and the angle of their normals. Principal Curvatures algorithm describes local geometry as a function of curvatures along principal directions.
- Histogram-based descriptors are a broad category of descriptors. The core idea is to take a large set of values describing the local geometry and sort them into bins, effectively compressing the information into a histogram. SHOT (Signature of Histograms of OrienTations) [41] is a popular example, along with Point Feature Histograms (PFH) described in detail in a later section 2.3.

2.2. Surface normal estimation

As a preliminary to PFH, surface normal estimation must be discussed first. Normals are the core of not only PFH but also many other point feature representations. Furthermore, having normals at a point is crucial in general for computer graphics applications, such as to being able to generate shadings by bouncing the light off and other visual effects.

The problem of determining the normal to a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem in $\mathcal{P}_q^k$ [37]. This can be performed using PCA.

The fitted plane is represented by a point x and a normal vector n_q. The plane is fitted in the least-square sense so that distances d_i from points $p_i \in \mathcal{P}_q^k$ defined as

$$d_i = (p_i - x) \cdot n_q \tag{2.4}$$

are minimal. This is achieved by setting x

$$x = \bar{p} = \frac{1}{k} \sum_{i=1}^{k} (p_i) \tag{2.5}$$

to the centroid of $\mathcal{P}_q^k$. Then n_q is then found by taking the covariance matrix $C \in \mathbb{R}^{3\times3}$ of $\mathcal{P}_q^k$

$$C = \frac{1}{k} \sum_{i=1}^{k} (p_i - \bar{p}) \cdot (p_i - \bar{p})^\top, \tag{2.6}$$

finding its eigenvectors and eigenvalues

$$C \cdot v_j = \lambda_j \cdot v_j, \quad j \in \{0, 1, 2\}. \tag{2.7}$$

If $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$ then the smallest principal component v_0 corresponding to the smallest eigenvalue λ_0 is an approximation of n_q.

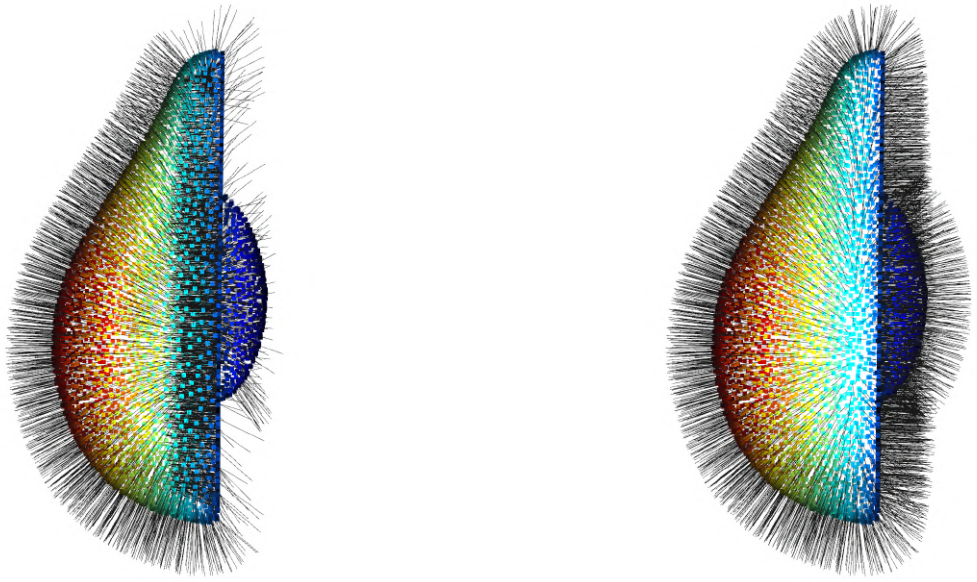
While this approach allows for normal estimation, it is not able to determine the appropriate orientation of the normal. The normal points either "into" the surface or "outside" the surface, the concrete choice is arbitrary, however, the choice should be consistent across the surface. One way of achieving that is using the viewpoint information. It stems from the assumption, that before registration, the point cloud is acquired from a single viewpoint, normals must be oriented consistently towards the viewpoint. More specifically,

$$\text{if } \frac{(v - p_i) \cdot n_{p_i}}{\|v - p_i\|} < 0, \text{ then } n_{p_i} \coloneqq -n_{p_i} \tag{2.8}$$

where v is the viewpoint. However, in the absence of information about the viewpoint the problem of orienting becomes more difficult. The PointCloudLibrary [40] chooses an arbitrary point $(0, 0, 0)$ as the viewpoint. A different, more computationally expensive, approach is proposed by [18]. The core idea is that two data points p_i, p_j that are geometrically close and therefore belong to the same part of the surface have the same orientation, meaning

$$n_{p_i} \cdot n_{p_j} \approx 1. \tag{2.9}$$

Given a starting point with a correctly oriented normal, the algorithm then reorients all normals to satisfy the local similarity of orientations. The algorithm uses an Euclidean Minimum Spanning Tree (EMST) to connect the neighboring points and weights based on Euclidean distances to determine orientation similarity. This approach is also useful when estimating normals for an already registered point cloud (one combined from multiple viewpoints). Figure 2.1 demonstrates the difference of normal orientation for such a point cloud.



(a) Inconsistent normal orientation

(b) Consistent normal orientation

Figure 2.1: Subfigure (a) is an example of a model with the normal orientation chosen based on the viewpoint resulting into inconsistent orientation across the model. On subfigure (b) all normals were reoriented to point outside the surface.

2.3. Point Feature Histogram

PFH represents a sophisticated and versatile method for capturing the spatial relationships between points in a point cloud, a crucial task for computer vision and robotics. At its core, PFH involves the computation of a high-dimensional feature space that encodes the local geometric properties of the point cloud and afterward a dimensional reduction using the histogram technique, to preserve the resolution capability of the point feature representation while minimizing the computational overhead caused by dimensionality. The main source for this section is the PFH author's thesis [37].

Normals and curvature are point feature representations that are easy to compute, but because they capture the neighborhood geometry $\mathcal{P}_q^k$ with only a few values, many points have similar values. In other words, the discrimination capability is comparatively

low. The main goal of PFH is to significantly improve the quality of the point feature representation. PFH achieves this by not only comparing the query point to its neighborhood, but by considering the relationship of all point pairs within the neighborhood $\mathcal{P}_q^k$.

When computing the Point Feature Histogram of a query point p_q, consider the set of its k nearest neighbors as defined in 2. Each point's surface normal must be known. The steps to compute the PFH are:

1. Establish a local coordinate system for each unique unordered pair of points p_i and p_j from $\mathcal{P}_q^k$.
2. Caclulate angular features describing the realtionship between the pair's normals.
3. Create a histogram based on all point pairs' angular features to reduce dimensionality.

Establishing local coordinate system

To calculate angular features for each point pair, a local coordinate system must be established, as depicted in Figure 2.2. To uniquely identify the origin for each point pair $\forall p_i, p_j \in \mathcal{P}_q^k, i \neq j$, define the point orientation vectors as

$$\begin{aligned} p_{ji} &= p_j - p_i, \\ p_{ij} &= p_i - p_j. \end{aligned} \tag{2.10}$$

Then label points p_i, p_j as the source point p_s and the target point p_t such that the source point is one with a smaller angle between its normal and a line connecting the source point with the target point.

$$\begin{aligned} &\text{if: } \arccos(n_i \cdot p_{ji}) \leq \arccos(n_j \cdot p_{ij}) \\ &\text{then } \begin{cases} p_s = p_i, & n_s = n_i, \\ p_t = p_j, & n_t = n_j, \end{cases} \\ &\text{else } \begin{cases} p_s = p_j, & n_s = n_j, \\ p_t = p_i, & n_t = n_i. \end{cases} \end{aligned} \tag{2.11}$$

Now the Darboux frame[1] uvw with origin at p_s is defined as:

$$\begin{aligned} u &= n_s \\ v &= u \times \frac{p_t - p_s}{\|p_t - p_s\|_2} \\ w &= u \times v. \end{aligned} \tag{2.12}$$

Calculating the angular features

With the Darboux frame established, the relationship between the two normals n_s and n_t is captured by

[1]Terminology used in consistency with [37].

$$\begin{aligned}
\alpha &= \arccos\left(v \cdot n_t\right), \\
\phi &= \arccos\left(u \cdot \frac{p_t - p_s}{\|p_t - p_s\|_2}\right), \\
\theta &= \arctan\left(\frac{w \cdot n_t}{u \cdot n_t}\right), \\
d &= \|p_t - p_s\|_2.
\end{aligned} \tag{2.13}$$

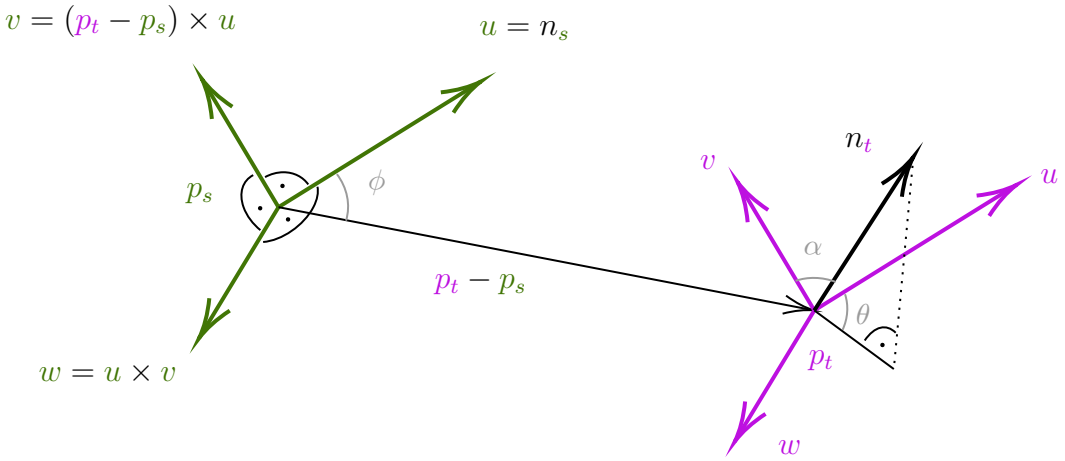


Figure 2.2: Darboux frame with highlighted angular features for a point pair.

Each point pair within the neighborhood $\mathcal{P}_q^k$ is represented with a quadruplet $(\alpha, \phi, \theta, d)$. The computational complexity for a single point is $\mathcal{O}(k^2)$ and for the entire point cloud with n points it is $\mathcal{O}(nk^2)$.

Histogram binning

The final step is to create a histogram representation from all the angular feature quadruplets. Normalizing the distances and the angles to the interval $[-1, 1]$ produces a transformed feature quadruplet (f_1, f_2, f_3, f_4). Splitting each feature at the middle of the interval generates a 16-bin histogram, where each bin represents the proportion of points with their features belonging to the appropriate interval. Point's histogram index idx is calculated as

$$idx = \sum_{i=1}^{4} step(s_i, f_i) \cdot 2^{i-1} \tag{2.14}$$

where the $step(s, f)$ function is defined as

$$step(s, f) = \begin{cases} 0 & f < s, \\ 1 & f \geq s. \end{cases} \tag{2.15}$$

Figure 2.3 illustrates how the Point Feature Histogram differs for points depending on which surface they are coming from.

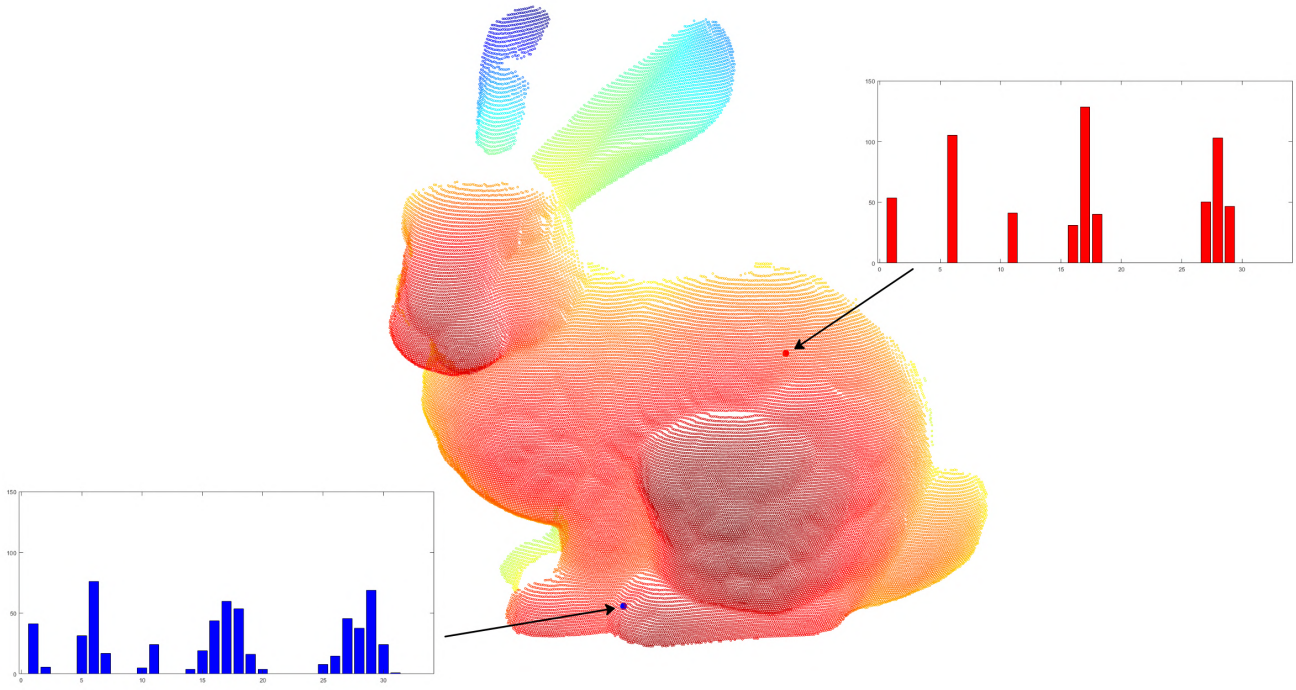


Figure 2.3: Examples of histograms sampled at different points on a model.

Histogram can be extended to any number of features with an arbitrary amount of interval splits. Furthermore, the splits are not required to be distributed equally along the interval. The generalized index function has a form

$$idx = \sum_{i=1}^{n} step_i(f_i) \cdot \frac{1}{div_1} \prod_{j=1}^{i} div_j \tag{2.16}$$

where div_i is a number of subintervals for a given feature f_i and $step_i$ is a generalized step function with arbitrary interval splits $s_j \in [f_{min}, f_{max}], j \in \{1, 2, \ldots, div_i - 1\}$.

$$step_i(f) = \begin{cases} 0, & f_i < s_1, \\ 1, & s_1 < f_i \geq s_2, \\ \vdots & \\ div_i - 1, & f_i \geq s_{div_i - 1}, \end{cases} \tag{2.17}$$

2.4. Fast Point Feature Histogram

In real time applications, the total processing time is the limiting factor for the feasibility of the algorithm, so it is especially important to consider algorithm optimizations. If the neighborhood $\mathcal{P}_q^k$ contains point p_i there is a high likelihood that a lot of point pairs are shared with $\mathcal{P}_i^k$. Rusu proposes storing calculated angular features computed in Eq. (2.13)

in a temporary cache. Caching is especially efficient when the dataset is ordered based on point proximity, allowing it to reduce the runtime to about 75 % [37].

Nonetheless, the theoretical computational complexity remains to be $\mathcal{O}(nk^2)$. Rusu et al. present a modification of PFH called Fast Point Feature Histogram (FPFH) [38] that achieves most of the discrimination accuracy while reducing the theoretical computational complexity to $\mathcal{O}(nk)$.

First, the notion of a Simplified Point Feature Histogram (SPFH) is introduced. Unlike $PFH(p_q)$ that computes angular features $(\alpha, \phi, \theta, d)$ for each point pair in the neighborhood $(p_i, p_j), \forall p_i, p_j \in \mathcal{P}_q^k, p_i \neq p_j$, the $SPFH(p_q)$ saves the required computations by only calculating angular features for pairs with the query point $(p_q, p_i), \forall p_i \in \mathcal{P}_q^k, p_i \neq p_q$. However, SPFH only captures the relationship of the query point with its neighborhood. To capture the local geometry, FPFH for the query point is created as a weighted average of SPFHs of points in the neighborhood

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\omega_i} \cdot SPFH(p_i) \tag{2.18}$$

where $m \leq k$ is the number of points in the neighborhood $\mathcal{P}_q^k$ and ω_i is a monotonically increasing function of distance of the point pair (p_q, p_i). Note that feature histograms are nothing but a b-dimensional point, where b is the number of histogram bins.

3. Color

Color is a complex phenomenon that not only adds beauty to the world around us but also constitutes a fundamental aspect of perception, communication an expression. The proper understanding of this phenomenon lies at the intersection of physics, biology and mathematical modeling. Due to the nature of point cloud data scanning, color values are often collected along Euclidean coordinates of points. Since color provides additional information to discriminate differing points, it presents an opportunity to further enhance point cloud registration.

This chapter aims to provide the necessary theoretical background about what color is from the physics point of view, how human eyes perceive it, and give examples of color spaces and different approaches to color representation, all to provide context, how color might be used as a point cloud descriptor.

3.1. What is color?

Fundamentally, color is not a physical property, but rather a subjective response to different wavelengths of light reflecting off surfaces and reaching our eyes. Color is a perception of light, part of the electromagnetic spectrum visible to the human eye, being reflected, transmitted, and emitted by objects. The visible electromagnetic spectrum[1] spans from wavelengths of 400 nanometers to 700 nanometers [13]. The shortest wavelengths of the visible portion of the spectrum are perceived as violet and the longest as red, with various colors residing in between, as shown in Figure 3.1.

Figure 3.1: Wavelengths of the visible spectrum in nanometers.

When light encounters an object, it's the object's inherent properties that determine which wavelengths are absorbed and which are reflected. The reflected wavelengths are then perceived by the eye. This phenomenon explains why a leaf appears green, it absorbs all colors of the spectrum except green, which is reflected. Similarly, an object appears white when it reflects all wavelengths and black when it absorbs them all without reflection.

3.2. Human perception of color

Vision is the most important human sense, responsible for 80 % of all perception [28]. The human visual system consists of the eye capturing the light, the optical nerve transferring the sensation to the brain, and the visual cortex, part of the cerebral cortex responsible for processing the visual information.

[1]The exact interval of visible light varies with sources. Formulas [4] used to calculate the tristimulus values used to represent color use the 380nm to 740nm range.

As light enters the eye through the pupil, it is projected by the lens onto the retina. Retina is covered with sensing cells, that are divided into two groups: rods and cones [13]. Rods are responsible for detecting light in low-light situations. However, these cells do not enable color sensing, this is the reason why vision in the dark is in grayscale. The other type of cells, cones, are responsible for vision when there is enough light. There are three types of cones, S, M, and L for short, medium, and long wavelengths. Human vision is therefore called trichromatic. The peaks in the response curves of the cones, as shown in Fig. 3.2, correspond to blue, green, and red respectively [11].

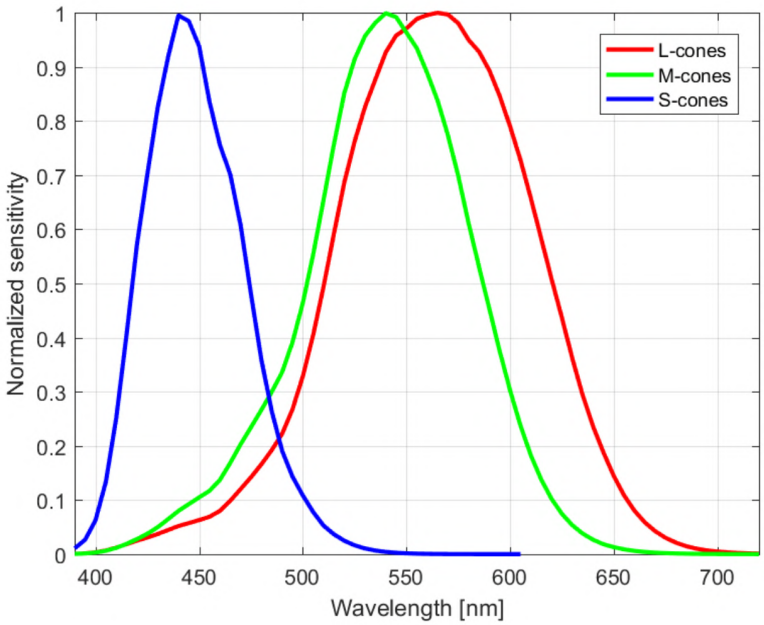


Figure 3.2: Response curves of the L, M, and S cones [7]. Response of each cone is normalized.

3.3. Measuring color

The task of measuring color is effectively a task of measuring light and its spectral characteristic. That can be measured using a spectrophotometer, a device, that measures the reflected light through a series of color filters, that separate the spectrum into narrow color bands, whose intensities are measured. Another approach, based on how the human eye perceives color, is to use a colorimeter, where the light is measured through a set of three filters: red, green, and blue. This is how most digital cameras operate as well, its sensor is covered in a color filter array, a pattern of red, green, and blue filters, producing a colorful image after a demosaicing process.

3D scanners use digital cameras and projection techniques to add color information to the point cloud data. The resulting point cloud is then described by Euclidean coordinates and RGB values.

Note, that this chapter omits the topic of camera calibration, color accuracy and processing of raw image data. There are multiple factors, that complicate the process of color reproduction. The data collected by the image sensor about the color varies by the

type of illuminant. The spectral power distribution of fluorescent light differs significantly from the sunlight at dawn. The human brain is able to compensate for the illumination change and perceive the color of a sheet of paper as white, however, to the camera, it appears as blue or orange. To compensate for the change of illuminant, the color balance needs to be adjusted. Another example of hidden complexity is the consistency of color across devices, i.e. ensuring the color is consistent across cameras, scanners, printers, and various display devices. The thesis focuses on point cloud processing rather than acquisition, therefore the following text assumes the color information in point clouds to be accurate. A detailed explanation of issues related to color accuracy can be found in [15, 21, 31].

3.4. Representing color - color spaces

As described at the beginning of the chapter, color is a perception of light. The light is described by its spectral power distribution (SPD), however, human eyes are not cannot perceive the SPD. This section outlines alternative approaches to representing color, that attempt to model the human eye and its trichromatic perception. These approaches are based on the observation, that any color can be recreated using an additive mix of three primary colors, thus representing color with a triplet of values of intensities of the primary colors. The system of primaries used to represent a color is called the color model or the color space. The following subsections describe some of the most commonly used color spaces, using [4] as the primary source.

3.4.1. RGB

The most widely adopted representation of color is using the red, green, and blue (RGB) primaries. Colors are represented by a triplet of intensities of the primary colors, creating the desired color through an additive mixture. A common misconception is that RGB is a color space, while in fact, it is a family of color spaces, where a concrete choice of the RGB primaries determines the specific color space. This is the reason why the RGB approach is called device dependent - each camera sensor and display have their own set of primaries.

The Commission Internationale de l'éclairage (CIE) in 1931 devised the CIE RGB color space. Monochromatic reference stimuli $\mathbf{R}, \mathbf{G}$, and $\mathbf{B}$ were selected as primaries, with respective wavelengths [31]

$$\begin{aligned} \lambda_R &= 700.0 \text{ nm}, \\ \lambda_G &= 546.1 \text{ nm}, \\ \lambda_B &= 435.8 \text{ nm}. \end{aligned} \tag{3.1}$$

Wright and Guild in a study asked observers to color match a set of lights with specified wavelength by varying the intensities of the primaries $\mathbf{R}, \mathbf{G}$,and $\mathbf{B}$. By averaging the responses the color matching functions $\bar{r}(\lambda), \bar{g}(\lambda)$, and $\bar{b}(\lambda)$ were devised for the standard observer [12]. See Figure 3.3 for a plot of the color matching functions.

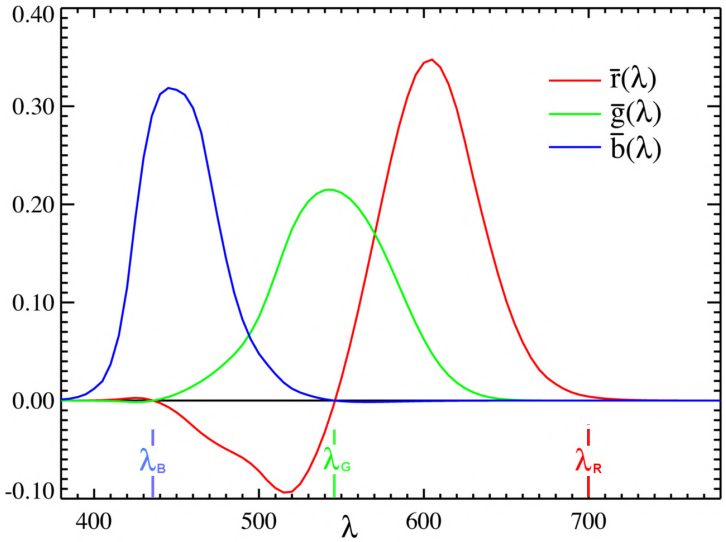


Figure 3.3: CIE 1931 RGB Color matching functions [9].

A monochromatic light E_{λ_i} with the wavelength λ_i is perceptually identical to the additive mixture using the color matching functions

$$E_{\lambda_i} \approx \bar{r}(\lambda_i)\mathbf{R} + \bar{g}(\lambda_i)\mathbf{G} + \bar{b}(\lambda_i)\mathbf{B}. \tag{3.2}$$

This provides an additive representation for any light stimulus Q. A light stimulus Q is a sum of monochromatic lights E_{λ_i} weighted by a factor $Q(\lambda_i)$. Then

$$Q(\lambda_i)E_{\lambda_i} \approx \bar{r}(\lambda_i)Q(\lambda_i)\mathbf{R} + \bar{g}(\lambda_i)Q(\lambda_i)\mathbf{G} + \bar{b}(\lambda_i)Q(\lambda_i)\mathbf{B}. \tag{3.3}$$

Using Grassman's linearity laws [4], the light stimulus Q has the same color as the additive mixture

$$Q \approx R\mathbf{R} + G\mathbf{G} + B\mathbf{B}, \tag{3.4}$$

where

$$\begin{aligned} R &= \int_{380}^{740} \bar{r}(\lambda)Q(\lambda)\mathrm{d}\lambda, \\ G &= \int_{380}^{740} \bar{g}(\lambda)Q(\lambda)\mathrm{d}\lambda, \\ B &= \int_{380}^{740} \bar{b}(\lambda)Q(\lambda)\mathrm{d}\lambda. \end{aligned} \tag{3.5}$$

The tristimulus (R, G, B) uniquely identifies any spectral power distribution, thus defining the CIE RGB color space.

3.4.2. XYZ

Although the RGB representation is ubiquitous, the CIE RGB color space has a flaw, that at the time of introduction severely limited its usability it permitted negative values. The International Commission on Illumination foresaw the issue and in 1931 also introduced a second color space, the CIE XYZ color space.

CIE XYZ was derived from CIE RGB with three set requirements. The first is for the color matching functions $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$, see Fig. 3.4, to be a linear combination of $\bar{r}(\lambda), \bar{g}(\lambda), \bar{b}(\lambda)$, such that $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$ are non-negative.

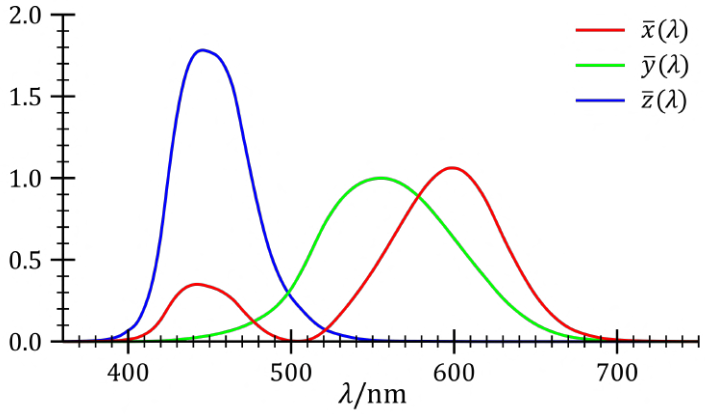


Figure 3.4: CIE 1931 XYZ color matching functions [8].

When CIE XYZ was developed the computational ability was a limiting factor and dealing with non-negative numbers allowed to simplify the calculation [31]. The second requirement was that the $\bar{y}(\lambda)$ be identical to the standard luminosity function so that the Y primary would correspond to the luminance of the color. The final requirement was to normalize the color matching functions so that the tristimulus values $X = Y = Z$ are equal for white light. The tristimulus (X, Y, Z) for light stimulus Q is obtained as

$$\begin{aligned} X &= \int_{380}^{740} \bar{x}(\lambda) Q(\lambda) \mathrm{d}\lambda, \\ Y &= \int_{380}^{740} \bar{y}(\lambda) Q(\lambda) \mathrm{d}\lambda, \\ Z &= \int_{380}^{740} \bar{z}(\lambda) Q(\lambda) \mathrm{d}\lambda. \end{aligned} \tag{3.6}$$

The primaries of CIE XYZ are sometimes called virtual primaries, as they do not correspond to a real color. The conversion between CIE RGB and CIE XYZ is possible via a simple linear transform

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.49000 & 0.31000 & 0.20000 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00000 & 0.01000 & 0.99000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{3.7}$$

3.4.3. HSV

The RGB representation and CIE XYZ are excellent for storing and transferring the color information, as displaying the color is just a matter of converting it to the device's color space by multiplying the color vector by the device's transformation matrix. However, without visualization, judging the properties of the color purely from the tristimulus values is difficult. The RGB and CIE XYZ representations do not align with how humans think about color. When working with color editing software, it is common to adjust

the brightness of a color, make it more intense or more pale, or perhaps shift the tone of color. To work with color in a more intuitive manner, RGB can be transformed into an HSV color space. HSV stands for hue, saturation, and value. Like RGB, HSV is device dependent, as it transforms device specific RGB tristimulus into an HSV representation.

When defining the HSV components, first a set of helper variables are defined, the maximum M, minimum m and the chroma C

$$\begin{aligned} M &= \max(R, G, B), \\ m &= \min(R, G, B), \\ C &= M - m. \end{aligned} \tag{3.8}$$

Hue component refers to the color's position on the color wheel, shown in Fig. 3.5, and is represented in degrees

$$\begin{aligned} H' &= \begin{cases} \text{undefined} & \text{if } C = 0, \\ \frac{G-B}{C} \bmod 6 & \text{if } M = R, \\ \frac{B-R}{C} + 2 & \text{if } M = G, \\ \frac{R-G}{C} + 4 & \text{if } M = B, \end{cases} \\ H &= 60^\circ * H'. \end{aligned} \tag{3.9}$$

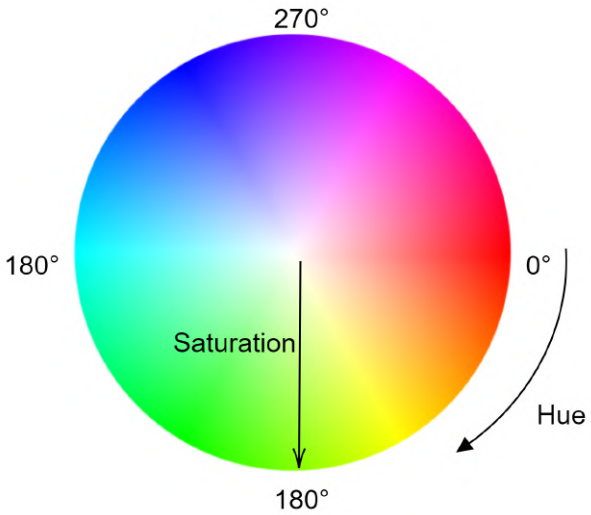


Figure 3.5: The color wheel. Obtained as a slice of the HSV space where $V = 1$

Value describes the intensity of color on a scale of 0 to 1. Value of 0 means the color is pure black, while value 1 can be either pure white or colorful, depending on the saturation value.

$$V = M = \max(R, G, B) \tag{3.10}$$

Finally saturation describes how pale or intense the color is on the scale from 0 to 1. Color with a saturation of 0 means is on the grayscale. Value of S is defined as

$$S = \begin{cases} 0 & \text{if } V = 0 \\ \frac{C}{V} & \text{otherwise.} \end{cases} \tag{3.11}$$

There are similar color representations to HSV, such as Hue, Saturation, Lightness (HSL) and Hue, Saturation, Intensity (HSI). These representations take a slightly different approach to defining the value and the saturation components, but the principle is the same, to separate color into more intuitive components.

3.4.4. LAB

A key functionality of point feature representations is the differentiation of points. To use color as a point feature representation, it is necessary to be able to reason about differences between colors. For that purpose, the HSV representation is much more useful than RGB since it allows to quickly compare whether the color is light or dark, pale or intense, or if it is more yellow or blue. However, the discriminative ability of HSV is limited because it is not a perceptually uniform color space.

A color space is called perceptually uniform when the distance between points in the given space is proportionate to the perceived difference between the colors corresponding to the points [4]. In 1976 CIE LAB (sometimes stylized as $L^\star a^\star b^\star$) was developed with perceptual uniformity as a goal.

The CIE LAB space is three dimensional color space. The $L^\star$ component represents lightness, matching the human perception of lightness, and ranges from 0 to 100. The $a^\star$ and $b^\star$ are chromaticity components, both ranging from -1 to 1 and when both are 0, the color is on the grayscale. The $a^\star$ component ranges from green to red and the $b^\star$ component ranges from blue to yellow. Figure 3.6 illustrates the axis of CIE LAB space.

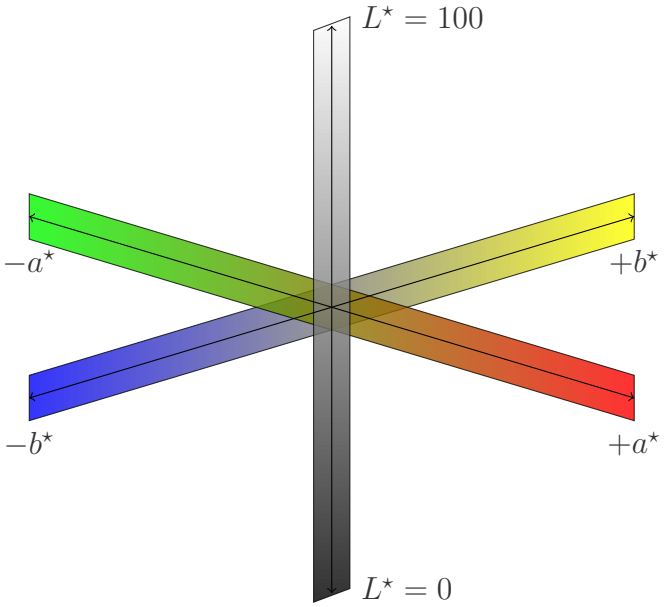


Figure 3.6: Axis of the CIE LAB color space.

CIE LAB values can be obtained by converting the CIE XYZ tristimulus

$$\begin{aligned} L^\star &= 116 f(Y/Y_n) - 16, \\ a^\star &= 500 f(X/X_n) - f(Y/Y_n), \\ b^\star &= 200 f(Y/Y_n) - f(Z/Z_n), \end{aligned} \tag{3.12}$$

where the function $f(t)$ is defined as

$$f(t) = \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3, \\ \frac{1}{3}t\delta^2 + \frac{4}{29} & \text{otherwise,} \end{cases} \qquad \delta = \frac{6}{29}. \tag{3.13}$$

The X_n, Y_n, Z_n are the tristimulus values of the reference white, the illuminant to which the color is adapted. An example is the standard D65[2] illuminant with tristimulus values of

$$\begin{aligned} X_n &= 95.0489, \\ Y_n &= 100, \\ Z_n &= 108.8840. \end{aligned} \tag{3.14}$$

The inverse formula to obtain CIE XYZ tristimulus from CIE LAB is

$$\begin{aligned} X &= X_n f^{-1}\left(\frac{L^\star + 16}{116} + \frac{a^\star}{500}\right), \\ Y &= Y_n f^{-1}\left(\frac{L^\star + 16}{116}\right), \\ Z &= Z_n f^{-1}\left(\frac{L^\star + 16}{116} - \frac{b^\star}{200}\right), \end{aligned} \tag{3.15}$$

with $f^{-1}(t)$ being the inverse of $f(t)$ in Eq.(3.13) using the same value for δ

$$f^{-1}(t) = \begin{cases} t^3 & \text{if } t > \delta, \\ 3\delta^2(t - \frac{4}{29}) & \text{otherwise.} \end{cases} \tag{3.16}$$

3.5. Using color as a point feature representation

A key benefit of using a color space designed with perceptual uniformity in mind, such as CIE LAB, is that the perceptual difference between colors can be measured via the Euclidean metric. In in order to attempt to improve the discrimination between points in a point cloud, not only their distance is considered, but also their color similarity. When performing point set registration, the points from the source and the target point clouds might be close, but if they are both a different color, or one is significantly darker than the other, it indicates that they were sampled from a different area of the surface. Including color information in the CIE LAB color space is simple, it is sufficient to treat the tristimulus values as additional dimensions to the Cartesian coordinates and without changing the metric a distance can be measured.

For example, measuring the distance between points p and q can be done as

$$d(p, q, \alpha) = \left\| \begin{bmatrix} p_x \\ p_y \\ p_z \\ \alpha p_{L^\star} \\ \alpha p_{a^\star} \\ \alpha p_{b^\star} \end{bmatrix} - \begin{bmatrix} q_x \\ q_y \\ q_z \\ \alpha q_{L^\star} \\ \alpha q_{a^\star} \\ \alpha q_{b^\star} \end{bmatrix} \right\|_2^2 \tag{3.17}$$

[2]D65 is an artificial illuminant designed to represent average natural daylight. Its correlated color temperature is roughly 6500K [31].

where α is a scalar used to weigh the color information against the coordinates. This way it is possible to use color with already exiting point registration methods without the need to modify the implementation.

Note, that although CIE LAB was designed to be perceptually uniform, it didn't achieve the set out goal perfectly. Since its introduction in 1976 more advanced color spaces were introduced [49], offering better perceptional uniformity, however, for purposes of point set registration CIE LAB is sufficient.

4. Implementation

The final chapter of the thesis is dedicated to the implementation of the previously discussed concepts, aiming to extend FPFH with color for ICP registration. The chapter features excerpts of code to illustrate how the core concepts were implemented. Note that for the sake of conciseness, the featured code is modified and stripped of some comments. All code used is attached, with a file tree in Appendix 4.4.3.

4.1. Prerequisites

Python is the language of choice for the implementation part of the thesis. It is a high level, dynamically typed, interpreted, garbage collected language. These properties together with Python's design philosophy promote readability, so the source code is comprehensible for readers with little software engineering background. Another benefit is it is completely free to use[1] and comes with a mature ecosystem, that is available for the vast majority of systems. A wealth of developed libraries is accessible through pip[2], the package installer included with most Python distributions. These properties make Python a great choice for proof-of-concept projects and demo code bases.

While there are many benefits to using Python, it does not come without trade-offs. The main drawbacks are related to performance. Due to Python's approach to objects and memory handling it is slower than compiled languages. The performance difference is noticeable especially when dealing with loops and large amounts of computation. Furthermore, due to the Global Interpreter Lock (GIL), Python bytecode can be executed only by one thread, meaning multi-core systems cannot be efficiently utilized. Such trade-offs are acceptable considering the goal is to create a demonstration of point cloud registration and point feature representation principles rather than a production, real-time system. Nonetheless, there are ways of limiting the performance penalties of using Python, such as making use of libraries for matrix computations, that utilize optimized precompiled routines for demanding computations.

The implementation makes use of several libraries, the most noteworthy are:

1. NumPy[3], short for Numerical Python, is a fundamental library for projects performing extensive numerical data processing. The core feature of NumPy is `ndarray`, a performant class for representing multidimensional arrays, that comes with a plethora of operators.

2. Trimesh[4] is a library designed for working with triangular meshes. It offers a simple interface for working with point clouds, supports loading most common point cloud file types and provides interactive visualization.

[1] Most Linux distributions come with Python preinstalled, on Windows and Macs it is available through their respective package systems. Alternatively, Python is available at https://www.python.org/downloads/.
[2] https://pypi.org/.
[3] https://numpy.org/
[4] https://trimesh.org/index.html

3. Sklearn[5], otherwise known as scikit-learn, is a powerful machine learning suite, that comes with data processing routines. The implementation uses Sklearn for efficient nearest neighbor search.

4.1.1. Point cloud formats

File formats that store point cloud data are divided into two types, binary and ASCII. ASCII-based formats store point cloud data in a text form, the primary benefit being human legibility and simpler parsing. Binary formats require dedicated encoding and decoding software, but produce compact files, thus being useful for storage and transfer of larger models.

Considering that point clouds are a fundamental part of 3D modeling, a number of formats used for storing point clouds were primarily intended as 3D model formats.

XYZ and CSV

The simplest approach to storing point cloud data is to represent them in a tabular form, where each row represents a point and the columns are its coordinates and additional features. CSV (Comma Separated Values) format, being a staple format in data science, is suitable for this purpose. The XYZ is a nonstandardized file format, that stores each point on a separate line, with the x, y, z coordinates separated by whitespace. Some implementations choose to follow the x, y, z coordinates by the r, g, b values. Other implementations contain a few header lines describing the number of points and other metadata, often to allow storing multiple point clouds in a single file. However, the lack of standardization complicates parsing, so this format is not preferred.

PLY

PLY [45] (Polygon File Format), also known as the Stanford Triangle Format, is a versatile file format primarily designed to store polygon and point cloud data, that supports both the binary and ASCII representations. A PLY file begins with a header describing the present elements and their properties, along with counts. The header is then followed by lists of elements. PLY is popular and well-supported due to the standardized representation of vertices, faces, triangles and normals as well as the option to extend the elements with custom properties. See below for an example of a tetrahedron with colored faces in PLY.

```
1  ply
2  format ascii 1.0
3  comment tetrahedron
4  element vertex 4
5  property float x
6  property float y
7  property float z
8  element face 4
9  property list uchar int vertex_indices
10 property uchar red { start of vertex color }
11 property uchar green
```

[5] https://scikit-learn.org/stable/

```
12  property uchar blue
13  end_header
14  0 0 0
15  1 1 0
16  1 0 1
17  0 1 1
18  3 1 2 3 127 127 127
19  3 1 0 2 255 0 0
20  3 3 2 0 0 255 0
21  3 0 1 3 0 0 255
```

OBJ

Developed in the 1980s by Wavefront Technologies, the OBJ file format was intended for 3D modeling. Primarily an ASCII format, however, binary implementations exist as well. In addition to storing geometric vertices and their parameters, as well as facets, it has support for higher order surfaces and offers options for texture mapping. The OBJ format is widely adopted by the 3D printing community.

PCD

PCD [39], short for Point Cloud Data, is a file format developed under the Point Cloud Library[6] project. Unlike the previously mentioned formats, it was developed specifically for point clouds. For this reason, it does not support meshes and facets, instead, it focuses solely on point cloud data and optimizing the I/O. It stores the viewpoint, and has support for multidimensional feature histograms and more. Another differentiator is the support of organized point cloud data. Data coming stereo and Time-of-Flight cameras have an image-like structure. An organized dataset preserves the spatial relationship between points in the dataset, resulting in speedups in the nearest neighbor operations [39].

Loading and showing a point cloud

Before processing point clouds, it is necessary to be able to load them. Fortunately, trimesh makes it easy. The following snippet shows how to load a mesh, access the underlying point cloud with color, and finally how to visualize the mesh. An example script to load a model with trimesh, shown in Listing 1. The results of the script are shown in Fig. 4.1.

[6] https://pointclouds.org/

Listing 1 Code to read and display a .ply model.

```
import numpy as np
import trimesh

pathToMesh = 'model.ply'

# load and visualize a triangle mesh
mesh = trimesh.load(pathToMesh)
mesh.show()
# extract XYZ and RGB data as matrices
dataXYZ = np.asarray(mesh.vertices)
dataRGBA = np.asarray(mesh.visual.vertex_colors)
# create and visualize a pointcloud
pointcloud = trimesh.PointCloud(dataXYZ)
pointcloud.colors = dataRGBA
pointcloud.show()
```

(a) mesh visualization (b) point cloud visualization

Figure 4.1: Result of program in Listing 1. The viewport is interactive; the model can be rotated and scaled.

The triangle mesh provides a clearer visualization of the point cloud's underlying surface. For the remainder of the chapter, whenever a point cloud is visualized and a triangle mesh is available, it will be preferred over showing the points directly.

4.2. Implementing Point Feature Histograms

The class `FeatureHistogram`, in `PFH.py`, encapsulates the functions necessary to compute PFH and FPFH.

The constructor, shown in Listing 2, provides an overview of the member variables of the class. It requires three arguments: the point cloud, thresholds, size of the k-neighborhood, and as an additional optional argument, normals at the points, in case they are provided with the model. The point cloud is expected to be a 2D NumPy array, where rows correspond to single points and columns contain Euclidean coordinates. The same convention is held for normals and storing feature representations as well as colors. Histogram binning is written in a general manner to not only support angular feature binning, like in PFH and FPFH, but also to allow it to be extended with an arbitrary point feature representation. The shape, i.e. the number of bins and value distribution, of the resulting histogram is determined by the shape of thresholds. Thresholds are a list of lists. The length of the outer list corresponds to a number of point features used for binning, in this case three (this implementation considers only angles, not the distances, in alignment with [38], to reduce the dimensionality of the histogram), while the inner lists contain the splitting thresholds. The inner lists must be in ascending order and can be of arbitrary length. Therefore the more splits each inner list of thresholds has, the more bins will the final point feature histogram have.

Listing 2 Constructor of the FeatureHistogram class.

```
1  class FeatureHistogram:
2      def __init__(self, pointCloud, thresholds, kNeighborhood, normals=None):
3          assert kNeighborhood > 0,
4              'neighborhood size must be a positive number'
5          if normals is not None:
6              assert pointCloud.shape == normals.shape,
7                  'point cloud and normals must have the same shape'
8
9          self.pts = pointCloud
10         self.thresholds = thresholds
11         self.kNeighborhood = kNeighborhood
12         self.featureRepresentation = None
13         self.neighborIndices = None
14         self.normals = normals
15         self.numBins = 1
16         for i in range(len(thresholds)):
17             self.numBins *= 1+len(thresholds[i])
```

To initiate the calculation of point feature histograms, the `getFeatureRepresentation()`, shown in Listing 3, the function must be called with an optional argument specifying the type of algorithm used to calculate the histograms. Valid options are `'naivePFH'`, `'hashmapPFH'`, and `'FPFH'`, which is also the default setting. If available, the function returns

previously computed feature representation, otherwise, it computes feature representation according to the selected algorithm.

Before calculating the histograms, the function `getFeatureRepresentation()` first checks, if the indices of the k-neighborhood and normals at each point are available. If not, k-neighborhood is calculated using `KDTree` from the Sklearn library and normals are estimated effectively by fitting a plane through the k-neighborhood and taking the normal of the plane as an estimate of the normal at the given point.

Listing 3 Main entry point for computation of variants of feature histograms.

```
def getFeatureRepresentation(self, histogramAlgorithm = 'FPFH'):
    if (self.featureRepresentation is None
        or self.histogramAlgorithm != histogramAlgorithm):
        if self.neighborIndices is None:
            self.calculateNeighbors()

        if self.normals is None:
            self.calcNormals()

        self.histogramAlgorithm = histogramAlgorithm
        match(histogramAlgorithm):
            case 'naivePFH':
                self.featureRepresentation = self.calcPFH_naive(
                    self.normals, self.neighborIndices)
            case 'hashmapPFH':
                self.featureRepresentation = self.calcPFH_hashmap(
                    self.normals, self.neighborIndices)
            case 'FPFH':
                self.featureRepresentation = self.calcFPFH(
                    self.normals, self.neighborIndices)
            case _: # use FPFH as default
                warnings.warn(
                    'Invalid histogram algorithm string. FPFH used instead.')
                self.featureRepresentation = self.calcFPFH(
                    self.normals, self.neighborIndices)

    return self.featureRepresentation
```

When the k-neighborhood is established and normals are available, feature histograms can be calculated. The option `'naivePFH'` and `'hashmapPFH'` are equivalent implementations of PFH in Sec. 2.3. The `'hashmapPFH'` variant stores computed angular features for point pairs in a cache, achieving a significant speedup. The `'FPFH'` variant implements FPFH in Sec. 2.4. These variants use the same functions to calculate the angular features and the bin index of the angular features. The calculation of angular features follows Eq. (2.13) directly. Similarly, the function for calculating the index of a histogram bin follows Eq. (2.16)-(2.17).

The `calcFPFH()` function, shown in Listing 4, consists of two main loops. The first loop iterates through all points, and computes angular features for its k-neighborhood and the SPFH, as in Sec. 2.4, based on the angular features. The second loop then creates the final FPFH for each point when the SPFH for all points is known. To further speed up the calculation, the function uses hashmap similarly to `calcPFH_hashmap()` to cache angular features for point pairs that appeared previously.

Listing 4 Function for computing Fast Point Feature Histograms.

```
def calcFPFH(self, normals, indNeigh):
    SPFH_Array = np.zeros((len(self.pts), self.numBins))
    # combination number "N choose 2"
    N_features = comb(self.kNeighborhood+1, 2)
    # dict(hashmap) representing the Darboux frame for each pointpair
    # key: tuple of pointpair indices
    # value: np.array containing a histogram index and point distance
    DarbouxDict = {}
    minimumDistance = np.inf
    for i in range(len(self.pts)):
        #indices of all unique point pairs in neighborhood
        numNeighbors = len(indNeigh[i])
        indices = np.zeros(numNeighbors, dtype=int)
        for j in range(numNeighbors):
            tup = tuple([i, indNeigh[i][j]])
            if tup not in DarbouxDict:
                feature = self.angularFeatures(
                    self.pts[i], normals[i],
                    self.pts[indNeigh[i][j]], normals[indNeigh[i][j]])
                index = self.calcBinIndex(feature)
                distance = np.linalg.norm(self.pts[i] - self.pts[indNeigh[i][j]])
                minimumDistance = min(minimumDistance, distance)
                DarbouxDict[tup] = np.array([index, distance])
                indices[j] = index
            else:
                indices[j] = DarbouxDict[tup][0]

        # clean SPFH for each point
        SPFH = np.zeros(SPFH_Array.shape[1])
        unique, counts = np.unique(indices, return_counts=True)
        SPFH[unique] = counts
        SPFH_Array[i,:] = SPFH / N_features

    # calculate the FPFH itself
    FPFH_Array = SPFH_Array.copy()
    for i in range(len(self.pts)):
        SPFH_sum = np.zeros(FPFH_Array.shape[1])
        for ind in indNeigh[i]:
            pointDistance = DarbouxDict[tuple([i, ind])][1]
            # closest point has a weight 1
            distanceWeight = pointDistance / minimumDistance
            SPFH_sum += SPFH_Array[ind] / distanceWeight

        FPFH_Array[i] += SPFH_sum / len(indNeigh[i])

    return FPFH_Array
```

4.3. Implementing ICP

Functionality related to point cloud registration is encapsulated in the file `ICP.py`. The main entry point is the function `ICP_features()`, shown in Listing 5, which produces a transform matrix to fit `srcPointCloud` onto `dstPointCloud`. The function iterates until a

maximum number of iterations is reached, or when the change of error between iterations is smaller than `stopDelta`. For ease of testing, it allows to set an initial alignment and offers the option of running ICP on a subsample of the input data. `ICP_features()` is general enough to support point clouds of arbitrary dimension, so it works for 2D and 3D point clouds alike, as long as the source and the target point cloud's dimension match.

Listing 5 ICP using Euclidean coordinates and point feature representations.

```
def ICP_features(srcPointCloud, srcFeatures, dstPointCloud, dstFeatures,
        initialAlignment=None, maxIterations=20, stopDelta=0.001, samplerate=1.0):
    assert srcPointCloud.shape[1] == dstPointCloud.shape[1],
    'Point cloud dimensions dont match'
    if (samplerate > 1) or (samplerate <= 0):
        samplerate = 1
        warnings.warn('Sample rate should be between 0 and 1')
    # subsampling
    if samplerate == 1:
        srcP = np.copy(srcPointCloud)
        srcF = np.copy(srcFeatures)
        dstP = np.copy(dstPointCloud)
        dstF = np.copy(dstFeatures)
    else:
        ids = np.random.uniform(0, 1, size=srcPointCloud.shape[0])
        srcP = srcPointCloud[ids <= samplerate, :]
        srcF = srcFeatures[ids <= samplerate, :]
        ids = np.random.uniform(0, 1, size=dstPointCloud.shape[0])
        dstP = dstPointCloud[ids <= samplerate, :]
        dstF = dstFeatures[ids <= samplerate, :]
    srcCopy = np.copy(srcP)

    # initial estimated alignment
    if initialAlignment is not None:
        srcP = applyAffineTransform(srcP, initialAlignment)

    lastError = 0
    # main ICP loop
    for currentIter in range(maxIterations):
        distances, indices = pairMatching(np.concatenate((srcP, srcF), axis=1),
                                          np.concatenate((dstP, dstF), axis=1))
        transform,_,_ = estimateTransformation(srcP, dstP[indices])
        # apply the estimated transform
        srcP = applyAffineTransform(srcP, transform)
        # stop condition
        error = np.mean(distances)
        if np.abs(lastError - error) < stopDelta:
            break

        lastError = error
    # get resulting transformation
    transform,_,_ = estimateTransformation(srcCopy, srcP)
    srcP = applyAffineTransform(srcCopy, transform)
    distances,_ = pairMatching(srcP, dstP)

    return transform, distances, currentIter
```

A small note on the resulting transform. In Eq. (1.3) the transformation of an n dimensional point cloud is described by $n \times n$ rotation matrix R and an n dimensional vector t. The above function opts for an alternative approach, performing the transformation in $n+1$ dimensional projective space using $(n+1) \times (n+1)$ affine transformation matrix T [10]. The points are projected by taking the proper points and adding ones as the last coordinate. The transform of a 3D point q onto p in projective space would be described by the following equation

$$\overset{p}{\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}} = \overset{T}{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}} \overset{q}{\begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix}} \tag{4.1}$$

where r_{ij}, t_i are elements of R and t respectively.

The `estimateTransformation()`, shown in Listing 6, uses the point-to-point algorithm described in Sec. 1.4.2 approach to estimating the best fit.

Listing 6 Point-to-point ICP approach to transform estimation.

```
def estimateTransformation(src, dst):
    assert src.shape == dst.shape, 'Point cloud dimensions dont match'

    # center point clouds
    srcCentroid = np.mean(src, axis=0)
    dstCentroid = np.mean(dst, axis=0)
    srcCentered = src - srcCentroid
    dstCentered = dst - dstCentroid
    # least squares fit using Singular Value Decomposition
    cov = np.dot(srcCentered.T, dstCentered)
    U, S, Vt = np.linalg.svd(cov)
    rotation = np.dot(Vt.T, U.T)
    # get number of dimensions
    m = src.shape[1]

    # special reflection case
    if np.linalg.det(rotation) < 0:
        Vt[m-1,:] *= -1
        rotation = np.dot(Vt.T, U.T)

    translation = dstCentroid.T - np.dot(rotation, srcCentroid.T)
    transform = np.identity(m+1)
    transform[:m, :m] = rotation
    transform[:m, m] = translation

    return transform, rotation, translation
```

Note that the ”special reflection case” comment refers to Eq.(1.14).

4.4. Comparing registration using feature histograms and color

The final section of the implementation chapter evaluates the utility of color as a point feature representation in comparison to FPFH.

4.4.1. Methodology

The evaluation of color consists of a series of experiments on varying data. Each experiment entails performing point cloud registration using ICP with varying combinations of point feature representations. To ensure consistency in nearest neighbor search, the point clouds are normalized to the range $[0, 1]$, likewise, the point feature representations are also scaled to the interval $[0, 1]$. Experiments are performed on multiple point clouds varying in size as well as shape, and can be divided into three types:

- Synthetic control. Target point cloud is obtained by rotating and translating the source and then registering it onto itself. Serves as a baseline for compared algorithms.
- Synthetic target. Target point cloud is obtained by adding Gaussian noise to the source and then rotating and translating it. Evaluates the sensitivity to noise.
- Realistic target. Source and target point clouds are overlapping scans from different viewpoints. The ground truth transform is not known in advance but provides insight into performance in a realistic scenario.

The parameters of FPFH and ICP stay the same between all experiments. FPFH uses a single threshold of 0 for each of the three angular features to create the histograms. Feature histograms are created from the 30 closest points. ICP is limited to 1000 iterations (never reached the limit) and uses 10^{-11} as the error delta for the stop criterion.

The point set registration function returns the estimation of optimal fit, list of distances for each matched point pair and number of ICP iterations the algorithm needed to reach the stop criterion. Mean and variance of the distances provide an insight into the closeness of fit. Mean alone describes the closeness of fit, however, ICP can get stuck at a local minimum which is visually obvious, but not easily detected from mean alone. Increased variance is a good indicator of a fit in a local minimum, alerting to adjust the ICP parameters. ICP iterations indicate the effect of the given choice of point feature representation on the convergence of ICP and serve as a proxy for computational intensity. As mentioned at the beginning of the chapter, due to Python's approach to memory management, higher dimensional point feature representations, which take up more memory, cause more cache misses and are disproportionately disadvantaged in computation time. For this reason time measurements are not considered as a part of the experiment. When two approaches produce comparable quality of fit, the one that took fewer iterations is preferred.

The advantage of experiments with synthetic targets is that the ground truth transform is known in advance. The difference in translation is evaluated as the Euclidean distance between translation vectors, while the difference between $R_{initial}$ and $R_{estimated}$ is evaluated using a norm ϕ defined as

$$\phi : SO(3) \times SO(3) \to \mathbb{R}^+, \quad \phi(R_{initial}, R_{estimated}) = \|\mathrm{I} - R_{initial} R_{estimated}^{\top}\|_F. \tag{4.2}$$

The intuition behind this norm uses the fact that an orthogonal rotational matrix $R \in SO(3)$ multiplied by its transpose yields an identity matrix and ϕ measures the Frobenius distance from the identity matrix. Further analysis of the ϕ metric including proof of satisfaction of metric definition in [22, 25].

4.4.2. Data

To test registration, the implementation uses data captured by the Microsoft Kinect sensor, used in a study developing Signature of Histograms of Orientations (SHOT) [41, 43]. Data, available in [44], consists of 4 datasets, Duck, PeterRabbit, Squirrel, and SuperMario, each dataset consists of objects scanned from different views.

4.4.3. Evaluation of experiments

The synthetic control experiment provides a baseline for measured values. In an ideal scenario, the mean and the variance of the pairwise distances would be zero, however, as is the case for iterative numerical methods, results are very small floating numbers, which are effectively considered to be zero. Visualization shows a perfect fit regardless of the point feature representation used. Plain ICP took 40 iterations to converge, ICP with FPFH took 16, and variants using any representation of color took around 8 iterations.

Synthetic control experiment					
Target Source		'SuperMario/SuperMarioPly002.ply' - 41417 points target point cloud rotated 30 degrees			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	40	2.4160e-15	2.7334e-33	1.4142	2.5900e-15
ICP & FPFH	16	2.4832e-15	4.4776e-33	1.4142	2.8489e-15
ICP & RGB	10	2.4637e-15	2.2721e-33	1.4142	2.8119e-15
ICP & HSV	8	2.3831e-15	3.5447e-33	1.4142	2.4230e-15
ICP & LAB	9	2.4456e-15	1.5085e-33	1.4142	2.6197e-15
ICP & FPFH & RGB	8	2.7547e-15	4.4718e-33	1.4142	2.6558e-15
ICP & FPFH & HSV	7	2.4145e-15	3.6235e-33	1.4142	2.3415e-15
ICP & FPFH & LAB	8	2.4445e-15	2.2009e-33	1.4142	2.5474e-15

Table 4.1: Synthetic control source is created by rotating the target point cloud 30 degrees.

The synthetic target experiments are performed with three levels of additive noise. The added noise is Gaussian with zero mean and variances of 0.005, 0.01 and 0.05. Across noise levels, the mean and the variance of pairwise distances are consistent across variants. A comparison of the estimated transform to the ground truth transform indicates using color without FPFH yields a smaller translation norm. With increased noise levels the advantage of color disappears. In the highest noise scenario use of FPFH seems to improve the estimation of the rotation, however upon visual inspection of the resulting registration it is clear, that none of the variants is able to overcome the high level of noise and produce an acceptable transformation.

The number of iterations before convergence was consistently lower for the variants using color.

Synthetic source experiment					
Target		'SuperMario/SuperMarioPly002.ply' - 41417 points			
Source		target is rotated 30 degrees - added noise with var. 0.005			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	88	0.0041791	8.5658e-06	1.4176	0.0028922
ICP & FPFH	54	0.0043715	9.2524e-06	1.4204	0.0086201
ICP & RGB	16	0.0041815	8.5899e-06	1.4146	0.0005566
ICP & HSV	14	0.0041815	8.5903e-06	1.4147	0.0006278
ICP & LAB	17	0.0041815	8.5898e-06	1.4146	0.0005310
ICP & FPFH & RGB	18	0.0041994	8.6331e-06	1.4197	0.0035228
ICP & FPFH & HSV	16	0.0041942	8.6189e-06	1.4189	0.0029271
ICP & FPFH & LAB	17	0.0041965	8.6244e-06	1.4179	0.0022897
Target		'SuperMario/SuperMarioPly002.ply' - 41417 points			
Source		target is rotated 30 degrees - added noise with var. 0.01			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	68	0.0082116	3.1876e-05	1.4257	0.010572
ICP & FPFH	80	0.0083029	3.2302e-05	1.4061	0.005293
ICP & RGB	19	0.0082221	3.2118e-05	1.4208	0.005577
ICP & HSV	15	0.0082239	3.2144e-05	1.4196	0.004449
ICP & LAB	17	0.0082228	3.2129e-05	1.4200	0.004860
ICP & FPFH & RGB	18	0.0082202	3.2077e-05	1.4213	0.004962
ICP & FPFH & HSV	16	0.0082222	3.2116e-05	1.4178	0.003084
ICP & FPFH & LAB	19	0.0082210	3.2092e-05	1.4180	0.002846
Target		'SuperMario/SuperMarioPly002.ply' - 41417 points			
Source		target is rotated 30 degrees - added noise with var. 0.05			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	56	0.040828	0.00084968	1.4183	0.019018
ICP & FPFH	49	0.040833	0.00084966	1.3932	0.020614
ICP & RGB	32	0.041037	0.00085697	1.4305	0.022025
ICP & HSV	22	0.041075	0.00085762	1.4295	0.015447
ICP & LAB	30	0.041082	0.00085854	1.4373	0.024815
ICP & FPFH & RGB	23	0.041035	0.00085637	1.4165	0.018070
ICP & FPFH & HSV	23	0.041072	0.00085702	1.4119	0.006534
ICP & FPFH & LAB	28	0.041062	0.00085741	1.4144	0.015668

Table 4.2: Synthetic source is created by rotating the target point cloud 30 degrees and adding varying degrees of noise.

Registration with realistic targets is performed with models "as is", using models from the dataset scanned from different viewpoints without any additional preprocessing, although pairs of point clouds for registration were selected from close viewpoints, ensuring there was enough overlap to make registration even possible. Across datasets, the use of point feature representations resulted in fewer iterations, especially when using color. The addition of color provided a slightly worse quality of fit than using only FPFH or plain ICP.

Realistic target experiment			
Target	'SuperMario/SuperMarioPly002.ply' - 41417 points		
Source	'SuperMario/SuperMarioPly001.ply' - 44219 points		
algorithm	Iters	dist. mean	dist. var.
ICP	94	0.0015479	4.7882e-06
ICP & FPFH	34	0.0016503	4.8626e-06
ICP & RGB	16	0.0022364	5.8409e-06
ICP & HSV	14	0.0024121	6.1474e-06
ICP & LAB	16	0.0022229	5.8396e-06
ICP & FPFH & RGB	14	0.0023133	5.9090e-06
ICP & FPFH & HSV	11	0.0024358	6.0984e-06
ICP & FPFH & LAB	13	0.0023636	5.9409e-06

Table 4.3: Point cloud registration on two views from the SuperMario dataset.

Tables with experiment results of other datasets are available in Appendix 4.4.3, as well as figures of the tested point clouds 4.4.3.

Conclusion

The objectives of this thesis were to provide an overview of Fast Point Feature Histograms, color spaces for point clouds, and to propose a new extension of FPFH using color spaces for point cloud registration. Then to provide an implementation of the aforementioned concepts together with an evaluation of color's usefulness in point cloud registration. Chapter 1 introduced point clouds, the methods of scanning, the problem of nearest neighbor search, and the concept of point set registration using ICP. Chapter 2 discussed point feature representations in general and focused on Point Feature Histograms and Fast Point Feature Histograms. The following Chapter 3 provided an overview of color and its representation in order to present color space candidates to be considered as a point feature representation in order to improve point cloud registration. Finally, Chapter 4 presented the implementation of ICP and Feature Histograms in Python, and proposed experiments to evaluate registration using various combinations of point feature representations with a primary focus on color.

The experiments proved color to be a valuable asset in the toolbox for point cloud registration, bringing down significantly the total number of ICP iterations required for convergence across combinations of point feature representations utilizing color while maintaining the accuracy of the registration.

The focus of the experiments was on using point feature representations for fine registration using ICP. An alternative approach is to use point feature representations to select key points of the point cloud and perform coarse registration based on the key points. The implementation provided with the thesis was developed with flexibility and extensibility in mind, making it easy to swap feature representations or registration algorithms using the same experiment measuring framework, providing a base for further research.

Bibliography

[1] Apple. *Spatial and logical arrangement of an example octree [graph]*. [cit. 2024-02-11]. Available at: https://developer.apple.com/documentation/gameplaykit/gkoctree.

[2] Arun, K. S., Huang, T. S. and Blostein, S. D. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1987, PAMI-9, no. 5, p. 698–700. DOI: 10.1109/TPAMI.1987.4767965. ISSN 0162-8828. Available at: https://ieeexplore.ieee.org/document/4767965.

[3] Beek, M., Small, C. F., Ellis, R. E., Sellens, R. W. and Pichora, D. R. Bone Alignment Using the Iterative Closest Point Algorithm. *Journal of Applied Biomechanics*. 2010, vol. 26, no. 4, p. 526–530. DOI: 10.1123/jab.26.4.526. ISSN 1065-8483. Available at: https://journals.humankinetics.com/view/journals/jab/26/4/article-p526.xml.

[4] Bertalmío, M. *Vision Models for High Dynamic Range and Wide Colour Gamut Imaging*. Academic Press, 2020. 131-155 p. Computer Vision and Pattern Recognition. ISBN 978-0-12-813894-6. Available at: https://www.sciencedirect.com/science/article/pii/B9780128138946000119.

[5] Besl, P. and McKay, N. D. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992, vol. 14, no. 2, p. 239–256. DOI: 10.1109/34.121791. ISSN 0162-8828. Available at: http://ieeexplore.ieee.org/document/121791/.

[6] Chen, Y. and Medioni, G. Object modelling by registration of multiple range images. *Image and Vision Computing*. 1992, vol. 10, no. 3, p. 145–155. DOI: 10.1016/0262-8856(92)90066-C. ISSN 02628856. Available at: https://linkinghub.elsevier.com/retrieve/pii/026288569290066C.

[7] Coliban, R.-M., Marincaş, M., Hatfaludi, C. and Ivanovici, M. Linear and Non-Linear Models for Remotely-Sensed Hyperspectral Image Visualization. *Remote Sensing*. august 2020, vol. 12, p. 2479. DOI: 10.3390/rs12152479.

[8] Commons, W. *The CIE 1931 XYZ color matching functions*. 2009. Available at: https://upload.wikimedia.org/wikipedia/commons/thumb/8/8f/CIE_1931_XYZ_Color_Matching_Functions.svg/2560px-CIE_1931_XYZ_Color_Matching_Functions.svg.png.

[9] Commons, W. *CIE 1931 RGB color matching functions with primary wavelengths shown*. 2023. Available at: https://commons.wikimedia.org/wiki/File:CIE1931_RGBCMF2.png.

[10] Craig, J. *Introduction to Robotics: Mechanics and Control*. Pearson/Prentice Hall, 2005. Addison-Wesley series in electrical and computer engineering: control engineering. ISBN 9780201543612.

[11] Deering, M. F. A photon accurate model of the human eye. In: *ACM SIGGRAPH 2005 Papers*. New York, NY, USA: Association for Computing Machinery, 2005, p. 649–658. SIGGRAPH '05. DOI: 10.1145/1186822.1073243. ISBN 9781450378253.

[12] Fairman, H. S., Brill, M. H. and Hemmendinger, H. How the cie 1931 color-matching functions were derived from wright-guild data. *Color Research & Application*. 1997, vol. 22, no. 1, p. 11–23. DOI: 10.1002/(SICI)1520-6378(199702)22:1<11::AID-COL4>3.0.CO;2-7.

[13] Foley, J. D., Dam, A. van, Feiner, S. and Hughes, J. *Computer Graphics: Principles and Practice*. 2.th ed. Reading, MA: Addison-Wesley, 1990. ISBN 978-0-201-12110-0. Available at: https://students.aiu.edu/submissions/profiles/resources/onlineBook/a6A8H5_computer%20graphics.pdf.

[14] Forbes, A., Oliveira, M. de and Dennis, M. R. *Structured light*. Springer Science and Business Media LLC, march 2021. DOI: 10.1038/s41566-021-00780-4.

[15] Fraser, B., Murphy, C. and Bunting, F. *Real World Color Management: Industrial-strength Production Techniques*. Peachpit Press, 2003. Real World Series. ISBN 9780201773408.

[16] Friedman, J. H., Bentley, J. L. and Finkel, R. A. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*. 1977, vol. 3, no. 3, p. 214–216. DOI: 10.1145/355744.355745. ISSN 0098-3500. Available at: https://dl.acm.org/doi/10.1145/355744.355745.

[17] Han, X., Jin, J. S., Xie, J., Wang, M. and Jiang, W. A comprehensive review of 3D point cloud descriptors. *CoRR*. 2018, abs/1802.02297. Available at: http://arxiv.org/abs/1802.02297.

[18] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. Surface reconstruction from unorganized points. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1992, p. 71–78. SIGGRAPH '92. DOI: 10.1145/133994.134011. ISBN 0897914791.

[19] Hu, L. and Nooshabadi, S. Massive parallelization of approximate nearest neighbor search on KD-tree for high-dimensional image descriptor matching. *Journal of Visual Communication and Image Representation*. 2017, vol. 44, p. 106–115. DOI: 10.1016/j.jvcir.2017.01.013. ISSN 10473203. Available at: https://linkinghub.elsevier.com/retrieve/pii/S1047320317300135.

[20] Huang, R., Yao, W., Ye, Z., Xu, Y. and Stilla, U. RIDF: A ROBUST ROTATION-INVARIANT DESCRIPTOR FOR 3D POINT CLOUD REGISTRATION IN THE FREQUENCY DOMAIN. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2020, V-2-2020, p. 235–242. DOI: 10.5194/isprs-annals-V-2-2020-235-2020. Available at: https://isprs-annals.copernicus.org/articles/V-2-2020/235/2020/.

[21] Hunt, R. *The Reproduction of Colour*. 2.th ed. Wiley, 2004-09-24. ISBN 9780470024256.

[22] Huynh, D. Metrics for 3D Rotations: Comparison and Analysis. *Journal of Mathematical Imaging and Vision*. october 2009, vol. 35, p. 155–164. DOI: 10.1007/s10851-009-0161-2. Available at: https://doi.org/10.1007/s10851-009-0161-2.

[23] Karatas, O. H. and Toy, E. Cone beam computerized tomography for craniofacial imaging [photo]. *European journal of dentistry: Three-dimensional imaging techniques: A literature review*. 2014, vol. 2014, no. 01, p. 132–140.

[24] Kazmi, I., You, L. and Zhang, J. A Survey of 2D and 3D Shape Descriptors. In: *International Conference Computer Graphics, Imaging, and Visualization*. August 2013, no. 10, p. 1–10. DOI: 10.1109/CGIV.2013.11.

[25] Larochelle, P. M., Murray, A. P. and Angeles, J. A Distance Metric for Finite Sets of Rigid-Body Displacements via the Polar Decomposition. *Journal of Mechanical Design*. july 2006, vol. 129, no. 8, p. 883–886. DOI: 10.1115/1.2735640. ISSN 1050-0472. Available at: https://doi.org/10.1115/1.2735640.

[26] Li, D., Wei, Y. and Zhu, R. A comparative study on point cloud down-sampling strategies for deep learning-based crop organ segmentation. *Plant Methods*. 2023, vol. 19, no. 1. DOI: 10.1186/s13007-023-01099-7. ISSN 1746-4811. Available at: https://plantmethods.biomedcentral.com/articles/10.1186/s13007-023-01099-7.

[27] Liu, Q. A Survey of Recent View-based 3D Model Retrieval Methods. *CoRR*. 2012, abs/1208.3670. Available at: http://arxiv.org/abs/1208.3670.

[28] Man, D. and Olchawa, R. In: Wojciech P. Hunek, S. P., ed. *Brain Biophysics: Perception, Consciousness, Creativity. Brain Computer Interface (BCI)*. 1.th ed. February 2018, p. 38–44. ISBN 978-3-319-75024-8.

[29] Marton, Z., Pangercic, D., Blodow, N. and Beetz, M. Combined 2D–3D categorization and classification for multimodal perception systems. *International Journal of Robotic Research - IJRR*. october 2011, vol. 30, p. 1378–1402. DOI: 10.1177/0278364911415897.

[30] Mason, A. *Making 3D Models with Photogrammetry [graph]*. [cit. 2024-02-10]. Available at: https://thehaskinssociety.wildapricot.org/resources/Pictures/Tutorials/Photogrammetry/photogrammetry.jpg.

[31] Ohta, N. and Robertson, A. R. *Colorimetry*. Wiley, 2005-11-11. ISBN 9780470094723.

[32] OpenDSA. *Example of a kd tree [graph]*. 2011 [cit. 2024-02-11]. Available at: https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/KDtree.html.

[33] OpenDSA. *Example of a PR quadtree [graph]*. 2011 [cit. 2024-02-11]. Available at: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html.

[34] Pomerleau, F., Colas, F. and Siegwart, R. A Review of Point Cloud Registration Algorithms for Mobile Robotics. *Foundations and Trends in Robotics*. 2013, vol. 4, no. 1, p. 1–104. DOI: 10.1561/2300000035. ISSN 1935-8253. Available at: http://www.nowpublishers.com/article/Details/ROB-035.

[35] Ram, P. and Sinha, K. Revisiting kd-tree for Nearest Neighbor Search. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: ACM, 2019-07-25, p. 1378–1388.

DOI: 10.1145/3292500.3330875. ISBN 9781450362016. Available at: https://dl.acm.org/doi/10.1145/3292500.3330875.

[36] Raychev, J., Hristov, G., Kyuchukova, D. and Zahariev, P. Workflow for development of a virtual museum that will provide better way for learning the cultural heritage. In: *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE, 2017, p. 1–5. DOI: 10.1109/ITHET.2017.8067815. ISBN 978-1-5386-3968-9. Available at: http://ieeexplore.ieee.org/document/8067815/.

[37] Rusu, R. B. *Semantic 3D object maps for everyday manipulation in human living environments*. Munich, 2009. PhD thesis. Technical University of Munich. Available at: https://mediatum.ub.tum.de/doc/800632/941254.pdf.

[38] Rusu, R. B., Blodow, N. and Beetz, M. Fast Point Feature Histograms (FPFH) for 3D registration. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, p. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[39] Rusu, R. B. and Cousins, S. *The PCD (Point Cloud Data) file format*. 2010. Available at: https://pcl.readthedocs.io/projects/tutorials/en/latest/pcd_file_format.html.

[40] Rusu, R. B. and Cousins, S. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9-13 2011.

[41] Salti, S., Tombari, F. and Di Stefano, L. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*. 2014, vol. 125, p. 251–264. DOI: 10.1016/j.cviu.2014.04.011. ISSN 1077-3142. Available at: https://www.sciencedirect.com/science/article/pii/S1077314214000988.

[42] Teledyne. *Laser Triangulation [graph]*. [cit. 2024-02-10]. Available at: https://imaging.teledyne-e2v.com/products/applications/3d-imaging/laser-triangulation/.

[43] Tombari, F., Salti, S. and Di Stefano, L. A combined texture-shape descriptor for enhanced 3D feature matching. In: *2011 18th IEEE International Conference on Image Processing*. 2011, p. 809–812. DOI: 10.1109/ICIP.2011.6116679. Available at: http://www.vision.deis.unibo.it/fede/papers/icip11.pdf.

[44] Tombari, F., Salti, S. and Stefano, L. D. *SHOT: Unique Signatures of Histograms for Local Surface Description*. 2013. Available at: http://www.vision.deis.unibo.it/research/80-shot.

[45] Turk, G. *The PLY Polygon File Format*. 1994. Available at: https://gamma.cs.unc.edu/POWERPLANT/papers/ply.pdf.

[46] Wald, I. and Havran, V. On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N). In: *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE, 2006, p. 61–69. DOI: 10.1109/RT.2006.280216. ISBN 1-4244-0693-5. Available at: http://ieeexplore.ieee.org/document/4061547/.

[47] Wang, Q., Tan, Y. and Mei, Z. Computational Methods of Acquisition and Processing of 3D Point Cloud Data for Construction Applications. *Archives of Computational Methods in Engineering*. 2020, vol. 27, no. 2, p. 479–499. DOI: 10.1007/s11831-019-09320-4. ISSN 1134-3060. Available at: http://link.springer.com/10.1007/s11831-019-09320-4.

[48] Wang, W., Zhang, Y., Ge, G., Jiang, Q., Wang, Y. et al. Multi-Dimensional Indexing Structure for PointCloud Data. In: *2021 13th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*. IEEE, 2021, p. 172–176. DOI: 10.1109/IHMSC52134.2021.00047. ISBN 978-1-6654-2836-1. Available at: https://ieeexplore.ieee.org/document/9556109/.

[49] Xue, Y. *Uniform color spaces based on CIECAM02 and IPT color difference equations*. Rochester, NY, 2008. Dissertation. Rochester Institute of Technology.

List of symbols

$\mathbb{R}$	Real numbers
$\mathcal{O}()$	Theoretical computational complexity
$\|a\|$	vector norm
$\|a\|_2$	Euclidean vector norm
$A^\top$	transpose operator
$a \cdot b$	dot product of a and b
$a \times b$	cross product of a and b
$\mathcal{P}_q^k$	neighborhood of a point p_q limited to k closest points
$\|R\|_F$	Frobenius matrix norm defined as $\|R\|_F = \left(\mathrm{Tr}(R^\top R)\right)^{\frac{1}{2}}$
I	Identity matrix
$SO(3)$	Special orthogonal group, also known as the 3D rotation group

Appendix

File tree of the attached sources

```
/
├── data ..... subdirectories contain .ply point cloud models
│   ├── Duck
│   ├── PeterRabbit
│   ├── Squirrel
│   ├── SuperMario
│   └── sourceOfModels.txt
├── PFH.py
├── ICP.py
├── descriptorEvaluator.py
├── evaluateColor.py
└── requirements.txt
```

Additional experiment results

Synthetic control experiment					
Target		'Duck/Duck_010.ply' - 26453 points			
Source		target point cloud rotated 30 degrees			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	95	4.5250e-15	3.2210e-33	1.4142	4.4668e-15
ICP & FPFH	29	5.7312e-08	7.0384e-16	1.4142	3.9493e-07
ICP & RGB	17	4.3135e-15	3.6305e-33	1.4142	3.9475e-15
ICP & HSV	15	4.2013e-15	2.4941e-33	1.4142	4.1130e-15
ICP & LAB	17	4.0836e-15	5.0805e-33	1.4142	3.5149e-15
ICP & FPFH & RGB	15	4.0934e-15	2.0028e-33	1.4142	3.8882e-15
ICP & FPFH & HSV	13	4.2644e-15	2.8455e-33	1.4142	4.2334e-15
ICP & FPFH & LAB	14	4.2055e-15	2.1931e-33	1.4142	4.1182e-15

Table 4.4: Synthetic control source is created by rotating the target point cloud 30 degrees.

Synthetic control experiment					
Target		'Squirrel/Scrat6.ply' - 56337 points			
Source		target point cloud rotated 30 degrees			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	65	0.00048427	5.2633e-08	1.4165	0.0039593
ICP & FPFH	20	8.3303e-15	4.0889e-33	1.4142	7.9600e-15
ICP & RGB	18	8.5255e-15	6.9030e-33	1.4142	8.6578e-15
ICP & HSV	15	8.4033e-15	5.4290e-33	1.4142	8.1863e-15
ICP & LAB	17	8.3797e-15	4.5236e-33	1.4142	7.7926e-15
ICP & FPFH & RGB	13	8.3345e-15	4.3178e-33	1.4142	7.8648e-15
ICP & FPFH & HSV	12	8.2912e-15	5.1033e-33	1.4142	7.8206e-15
ICP & FPFH & LAB	13	8.5787e-15	8.8105e-33	1.4142	8.2078e-15

Table 4.5: Synthetic control source is created by rotating the target point cloud 30 degrees.

Synthetic control experiment					
Target		'PeterRabbit/PeterRabbit001.ply' - 13944 points			
Source		target point cloud rotated 30 degrees			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	73	3.8460e-15	3.0363e-33	1.4142	4.0045e-15
ICP & FPFH	19	3.6239e-15	3.5330e-33	1.4142	3.7374e-15
ICP & RGB	9	3.9168e-15	8.5104e-34	1.4142	3.9987e-15
ICP & HSV	8	3.9654e-15	3.1444e-33	1.4142	4.1629e-15
ICP & LAB	8	3.7971e-15	1.6535e-33	1.4142	3.6972e-15
ICP & FPFH & RGB	7	3.9085e-15	1.6048e-33	1.4142	3.9995e-15
ICP & FPFH & HSV	7	3.8741e-15	2.9334e-33	1.4142	4.1180e-15
ICP & FPFH & LAB	7	3.6431e-15	3.2636e-33	1.4142	3.9546e-15

Table 4.6: Synthetic control source is created by rotating the target point cloud 30 degrees.

Synthetic source experiment					
Target		'Duck/Duck_010.ply' - 26453 points			
Source		target is rotated 30 degrees - added noise with var. 0.005			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	159	0.0040527	8.5361e-06	1.4138	0.0011291
ICP & FPFH	85	0.0045703	1.0808e-05	1.4128	0.0125461
ICP & RGB	21	0.0040568	8.5516e-06	1.4143	0.0002382
ICP & HSV	18	0.0040568	8.5515e-06	1.4144	0.0002276
ICP & LAB	19	0.0040567	8.5512e-06	1.4144	0.0002298
ICP & FPFH & RGB	28	0.0040697	8.5959e-06	1.4130	0.0035629
ICP & FPFH & HSV	20	0.0040649	8.5782e-06	1.4125	0.0037377
ICP & FPFH & LAB	23	0.0040696	8.5935e-06	1.4127	0.0034252
Target		'Duck/Duck_010.ply' - 26453 points			
Source		target is rotated 30 degrees - added noise with var. 0.01			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	169	0.0079951	3.3859e-05	1.4101	0.0040455
ICP & FPFH	95	0.0082179	3.5903e-05	1.4151	0.0152312
ICP & RGB	25	0.0080157	3.4130e-05	1.4149	0.0009005
ICP & HSV	21	0.0080176	3.4148e-05	1.4149	0.0005565
ICP & LAB	23	0.0080169	3.4143e-05	1.4149	0.0006974
ICP & FPFH & RGB	26	0.0079977	3.3924e-05	1.4133	0.0032430
ICP & FPFH & HSV	23	0.0079956	3.3898e-05	1.4127	0.0044227
ICP & FPFH & LAB	24	0.0079965	3.3904e-05	1.4127	0.0035430
Target		'Duck/Duck_010.ply' - 26453 points			
Source		target is rotated 30 degrees - added noise with var. 0.05			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	90	0.042924	0.00086462	1.4226	0.025865
ICP & FPFH	73	0.042923	0.00086805	1.4243	0.026432
ICP & RGB	46	0.043146	0.00088418	1.4252	0.010483
ICP & HSV	38	0.043237	0.00088883	1.4234	0.012042
ICP & LAB	48	0.043169	0.00088544	1.4226	0.010401
ICP & FPFH & RGB	51	0.043172	0.00088510	1.4195	0.004885
ICP & FPFH & HSV	36	0.043251	0.00088912	1.4181	0.003293
ICP & FPFH & LAB	63	0.043185	0.00088595	1.4189	0.005299

Table 4.7: Synthetic source is created by rotating the target point cloud 30 degrees and adding varying degrees of noise.

Synthetic source experiment					
Target		'Squirrel/Scrat6.ply' - 56337 points			
Source		target is rotated 30 degrees - added noise with var. 0.005			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	86	0.0040852	8.7495e-06	1.4147	0.001292
ICP & FPFH	79	0.0043955	1.0111e-05	1.4306	0.019791
ICP & RGB	43	0.0040853	8.7513e-06	1.4147	0.000928
ICP & HSV	27	0.0040854	8.7530e-06	1.4144	0.000501
ICP & LAB	28	0.0040854	8.7521e-06	1.4146	0.000705
ICP & FPFH & RGB	33	0.0042259	9.3933e-06	1.4115	0.011968
ICP & FPFH & HSV	29	0.0042090	9.2941e-06	1.4119	0.005836
ICP & FPFH & LAB	31	0.0042155	9.3374e-06	1.4129	0.009235
Target		'Squirrel/Scrat6.ply' - 56337 points			
Source		target is rotated 30 degrees - added noise with var. 0.01			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	135	0.0080583	3.4576e-05	1.4156	0.0038705
ICP & FPFH	102	0.0082199	3.5820e-05	1.4229	0.0082931
ICP & RGB	50	0.0080593	3.4606e-05	1.4159	0.0018720
ICP & HSV	42	0.0080600	3.4616e-05	1.4160	0.0017137
ICP & LAB	47	0.0080595	3.4609e-05	1.4160	0.0016956
ICP & FPFH & RGB	41	0.0081380	3.5243e-05	1.4121	0.0119470
ICP & FPFH & HSV	28	0.0081227	3.5091e-05	1.4130	0.0059822
ICP & FPFH & LAB	31	0.0081311	3.5168e-05	1.4128	0.0092223
Target		'Squirrel/Scrat6.ply' - 56337 points			
Source		target is rotated 30 degrees - added noise with var. 0.05			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	77	0.041034	0.00084829	1.3569	0.062082
ICP & FPFH	59	0.041048	0.00084791	1.3491	0.062372
ICP & RGB	67	0.041038	0.00085011	1.3648	0.064482
ICP & HSV	63	0.041061	0.00085171	1.3800	0.056576
ICP & LAB	72	0.041047	0.00085081	1.3703	0.062021
ICP & FPFH & RGB	62	0.041031	0.00084918	1.3528	0.068512
ICP & FPFH & HSV	53	0.041043	0.00084992	1.3694	0.054993
ICP & FPFH & LAB	67	0.041033	0.00084943	1.3589	0.063152

Table 4.8: Synthetic source is created by rotating the target point cloud 30 degrees and adding varying degrees of noise.

Synthetic source experiment					
Target		'PeterRabbit/PeterRabbit001.ply' - 13944 points			
Source		target is rotated 30 degrees - added noise with var. 0.005			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	97	0.0040033	8.0718e-06	1.4067	0.0063612
ICP & FPFH	79	0.0049428	1.2271e-05	1.2520	0.1404545
ICP & RGB	13	0.0040082	8.0951e-06	1.4139	0.0004668
ICP & HSV	12	0.0040083	8.0953e-06	1.4138	0.0004657
ICP & LAB	11	0.0040083	8.0955e-06	1.4141	0.0002673
ICP & FPFH & RGB	14	0.0040132	8.0923e-06	1.4038	0.0109234
ICP & FPFH & HSV	15	0.0040110	8.0910e-06	1.4078	0.0073182
ICP & FPFH & LAB	16	0.0040124	8.0959e-06	1.4057	0.0094434
Target		'PeterRabbit/PeterRabbit001.ply' - 13944 points			
Source		target is rotated 30 degrees - added noise with var. 0.01			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	119	0.0076510	3.1097e-05	1.3907	0.019426
ICP & FPFH	98	0.0082140	3.5451e-05	1.2277	0.158932
ICP & RGB	15	0.0076738	3.1489e-05	1.4127	0.001596
ICP & HSV	13	0.0076755	3.1510e-05	1.4135	0.000775
ICP & LAB	15	0.0076747	3.1501e-05	1.4141	0.000534
ICP & FPFH & RGB	16	0.0076679	3.1354e-05	1.4001	0.013749
ICP & FPFH & HSV	13	0.0076714	3.1400e-05	1.4045	0.010113
ICP & FPFH & LAB	14	0.0076704	3.1385e-05	1.4045	0.010124
Target		'PeterRabbit/PeterRabbit001.ply' - 13944 points			
Source		target is rotated 30 degrees - added noise with var. 0.05			
algorithm	Iters	dist. mean	dist. var.	rot. norm ϕ	transl. norm
ICP	51	0.042594	0.00087529	1.3088	0.090438
ICP & FPFH	43	0.042605	0.00087676	1.2562	0.132361
ICP & RGB	19	0.042756	0.00089273	1.3774	0.039531
ICP & HSV	25	0.042798	0.00089512	1.3741	0.040052
ICP & LAB	26	0.042782	0.00089432	1.3772	0.038495
ICP & FPFH & RGB	23	0.042777	0.00089379	1.3697	0.041496
ICP & FPFH & HSV	24	0.042817	0.00089612	1.3803	0.031474
ICP & FPFH & LAB	31	0.042804	0.00089530	1.3665	0.043309

Table 4.9: Synthetic source is created by rotating the target point cloud 30 degrees and adding varying degrees of noise.

Realistic target experiment			
Target	'Duck/Duck_010.ply' - 26453 points		
Source	'Duck/Duck_011.ply' - 27114 points		
algorithm	Iters	dist. mean	dist. var.
ICP	82	0.0019030	6.1206e-06
ICP & FPFH	37	0.0019465	6.0267e-06
ICP & RGB	19	0.0021143	6.4834e-06
ICP & HSV	24	0.0021704	6.4599e-06
ICP & LAB	26	0.0021266	6.4200e-06
ICP & FPFH & RGB	22	0.0020944	6.2300e-06
ICP & FPFH & HSV	13	0.0021594	6.2472e-06
ICP & FPFH & LAB	18	0.0020853	6.3023e-06

Table 4.10: Point cloud registration on two views from the Duck dataset.

Realistic target experiment			
Target	'Squirrel/Scrat6.ply' - 56337 points		
Source	'Squirrel/Scrat5.ply' - 57395 points		
algorithm	Iters	dist. mean	dist. var.
ICP	89	0.0022085	8.7790e-06
ICP & FPFH	52	0.0025715	8.0141e-06
ICP & RGB	32	0.0029784	7.0152e-06
ICP & HSV	28	0.0029863	7.1137e-06
ICP & LAB	38	0.0028891	7.0627e-06
ICP & FPFH & RGB	23	0.0029531	7.4570e-06
ICP & FPFH & HSV	22	0.0030395	7.3121e-06
ICP & FPFH & LAB	25	0.0029097	7.3368e-06

Table 4.11: Point cloud registration on two views from the Squirrel dataset.

Realistic target experiment			
Target	'PeterRabbit/PeterRabbit001.ply' - 13944 points		
Source	'PeterRabbit/PeterRabbit000.ply' - 13357 points		
algorithm	Iters	dist. mean	dist. var.
ICP	98	0.0026003	7.5680e-06
ICP & FPFH	77	0.0028571	7.5086e-06
ICP & RGB	21	0.0026633	8.0981e-06
ICP & HSV	17	0.0027389	8.0929e-06
ICP & LAB	16	0.0026746	8.1034e-06
ICP & FPFH & RGB	14	0.0029608	8.2826e-06
ICP & FPFH & HSV	14	0.0030176	8.4232e-06
ICP & FPFH & LAB	12	0.0029675	8.2860e-06

Table 4.12: Point cloud registration on two views from the PeterRabbit dataset.

Point clouds used in the experiments

(a) `SuperMarioPly001.ply`

(b) `SuperMarioPly002.ply`

Figure 4.2: Used point clouds from the SuperMario dataset.

(a) `Duck_011.ply`

(b) `Duck_010.ply`

Figure 4.3: Used point clouds from the Duck dataset.

(a) `Scrat6.ply` (b) `Scrat5.ply`

Figure 4.4: Used point clouds from the Squirrel dataset.

(a) `PeterRabbit000.ply` (b) `PeterRabbit001.ply`

Figure 4.5: Used point clouds from the PeterRabbit dataset.