UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Karl - Walter Sillaots

Mobile AR Point Cloud Matching

Master's Thesis (30 ECTS)

Supervisors:
Raimond-Hendrik Tunnel, MSc
Timo Kallaste, BSc

Tartu 2021

Mobile AR Point Cloud Matching

Abstract:

This thesis explains the process of point cloud matching, to assess its viability in mobile markerless augmented reality solutions. Traditional point cloud matching algorithms like 4-Point Congruent Systems are described and considered. 2 Point Normal Sets was chosen for real-world experimentation due to being faster and easier to implement. Additional ideas to improve performance were implemented in addition to the chosen algorithm: A pair limiter that only uses up to 100 pairs each iteration for the matching process, changing the algorithm to only perform rotations across one axis and a point sampler to reduce the amount of points used. The best values for rotation error delta were also analyzed.

The first experiment done was with a point cloud in the shape of a rectangular box, to confirm if the additional ideas did improve performance. For the second experiment, various point cloud scans were made of a real-world room in Gallery Pallas. The algorithm was tested using these scans and the experiment results were documented. For the third experiment, an additional limiter was added to the point cloud scanner, so that it would only accept points closer than 2 meters from the scanner. This was done for more accurate point clouds. A second set of tests were done with this modification and the experiment results written.

The fourth and final experiment was done on a mobile device. While the previous experiments showed promise and improvements with each advancement, the algorithm had problems on a mobile device when matching against a unique point cloud for each run. Since there were still good matches, the possibility of using point cloud matching for markerless augmented reality is there, but would require additional work before it can be considered usable in applications.

Keywords:

Augmented reality, AR, mobile device, smart device, 3D, Unity, computer graphics, Android, iOS, mobile application, point cloud, point cloud matching, point cloud registration

CERCS: P170 - Computer science, numerical analysis, systems, control

Liitreaalsuse punktipilvede võrdlus mobiilseadmel

Lühikokkuvõte:

Antud magistritöös vaadatakse punktipilvede võrdlust ja selle potentsiaali alternatiivina teiste liitreaalsuse lahenduste jaoks. Osasid pakutud punktipilve võrdlusalgoritme nagu 4-Point Congruent Systems on võrreldud. 2 Point Normal Sets valiti eksperimentide jaoks välja, kuna selle võrdluskiirus on parem ja seda on kergem implementeerida. Valitud algoritmiga tehti veel lisamuudatusi: Piirangud, et iga iteratsiooni võrdluse ajal kasutatakse kuni 100 paari, pöördeid tehakse ainult ühe telje peal ja analüüsitud punktide vähendamine sämplimise kaudu. Samuti analüüsiti parimaid väärtuseid pöördevea delta jaoks.

Esimene eksperiment tehti risttahukakujulisel punktipilvel, et leida, kas lisamuudatused aitasid võrdluskiirust tõsta. Teise eksperimendi jaoks tehti mitmeid punktipilve skänne ruumist Pallase Kunstigaleriist. Algoritmi tulemused skännide põhjal pandi kirja. Kolmanda eksperimendi jaoks pandi punktipilve skänneerijale juurde piirang ainult lisada punktipilvele punkte, mis asuvad vähem kui kahe meetri kaugusel. Seda tehti täpsemate punktipilvede jaoks. Uued testid ja tulemused said lisatud selle muudatusega.

Neljas ja viimane eksperiment tehti läbi mobiilseadmel. Iga eelneva eksperimendiga läksid tulemused järjest paremaks, kuid mobiilseadmel tekkisid probleemid iga kord uue punktipilve vastu võrdlemisega. Heade tulemuste olemasolu tõttu oleks võimalik punktipilvede võrdlust kasutada alternatiivina liitreaalsuses, kuid vajab lisatööd enne, kui seda saaks rakendustes kasutada.

Võtmesõnad:

Liitreaalsus, AR, mobiilseade, nutiseade, 3D, Unity, arvutigraafika, Android, iOS, mobiilrakendus, punktipilv, punktipilve võrdlus, punktipilve registreerimine

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

Augmented reality (AR) is getting increasing support in business settings as a way to improve user experience and performance [1]. The devices most commonly used are mobile devices such as smartphones and tablets, or wearable goggles that offer more portability. One possible solution for environment-based AR problems[1] is real-time generation and tracking of arbitrary feature points in the environment. These points are used to generate and overlay virtual *planes* on horizontal or vertical surfaces, based on the detected real-world surfaces. The plane can then be interacted with by the user. A common use case would be placing an item on the plane, after which other interactions become available, like moving, resizing or rotating the object. This is known as *markerless* tracking. Its alternative, *marker-based* tracking, works by pointing towards a specific object to recognize, after which an action is performed.

Planes are a common use case available in existing AR application programming interfaces[2], but also limit AR usage in the environment to situations where the user can only add new objects at run-time to their chosen locations. It does not allow loading and adding objects in beforehand. The latter would need to have fixed locations in an environment for the objects and then detecting these locations in different environmental conditions afterwards. An idea offered by a mobile software development company Mobi Lab[3] was the exploration into such a solution, where a room would be prescanned with saved object locations. After a user has made a new scan of the room, the two different scans would be matched together by finding a suitable matching transformation between them. Any virtual objects that have been placed inside the original scan could then be transformed and placed in the correct position in the new scan.

Existing implementations that solve this problem in a similar manner either require monthly payments (Microsoft Azure Spatial Anchors[4]), sharing data (Google Cloud Anchors[5]) or are limited to a single platform (Apple ARWorldMap[6]). Other free implementations, like OpenGR[7], are set up in a way that it is difficult to port over to a project. Understanding how they work and being able to recreate them would be more beneficial for Mobi Lab. This means

[1] A problem where the surroundings play an important role in creating a solution (architectural problems versus overlapping objects on top of a person)
[2] https://developers.google.com/ar/reference/java/com/google/ar/core/Plane
https://developer.apple.com/documentation/arkit/arplanedetection
[3] https://lab.mobi/
[4] https://azure.microsoft.com/en-us/services/spatial-anchors/
[5] https://developers.google.com/ar/develop/java/cloud-anchors/overview-android
[6] https://developer.apple.com/documentation/arkit/arworldmap
[7] https://storm-irit.github.io/OpenGR/index.html

looking into algorithms that can solve point cloud matching and perform experiments that can be used to determine possible shortcomings.

Chapter 2 explores all the issues that come with solving this problem for mobile devices, how to decide what kind of algorithm to pick based on these issues, what was chosen at the end and what kind of optimizations were suggested to improve results. Chapter 3 describes the experiment setup and results of the given algorithm with optimizations considered using different test cases, looking for further improvements based on actual results and performing the tests on a mobile device. Finally, chapter 4 concludes the thesis. The appendix includes a glossary for some of the more important terms that are not explained in the thesis and the mobile hardware used for testing.

2. Point Cloud Matching

Point cloud matching is the process of finding a spatial transformation that best aligns two different point clouds. These point clouds can be identical copies with just changed rotations or completely different ones with only parts of either point cloud being similar. Such algorithms are used in various computer vision, pattern recognition and robotics fields[8].

For two different point sets, over a hundred possible algorithms to match them have been developed, with a dedicated GitHub page[9] to keep track of them. As more optimized matching algorithms are created for different point cloud matching cases, it is important to determine which of them are potentially suited for a mobile device-based application.

The first subchapter will go into detail on all the conditions that need to be taken into account to select a suitable point cloud matching algorithm for experimentation. The rest of the subchapters explain the road to how the chosen algorithm was developed, how it works and what further optimizations are suggested in the context of this thesis project.

2.1. Selecting an Algorithm

The first thing to assess for selecting an algorithm is what kind of point cloud is being created when using a mobile device to scan the room. The created application's point cloud scanning process (more details in chapter **3.1. Application**) simply requires the user to point their personal smartphone towards a location and it will try to overlay points on top of it. It is not completely accurate though, with the point cloud representations of walls, floors and objects being noisy, or incorrectly placing points within the scene causing outliers.

It is important to know what kind of issues this kind of point cloud can cause and determine what algorithms can match it well against another saved point cloud. Good runtime speed, accuracy and an implementation that is not too complex are all important for the selection process.

[8] Examples of use cases:
Pattern recognition - https://www.pointcloudtechnology.com/en/use-cases/
Robotics - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/
[9] https://github.com/gwang-cv/Point-Set-Matching-Registration-Material

2.1.1. Issues with Point Cloud Generation

Comparing point clouds to match them together is already a computationally complex task. Considering how the mobile devices generate point clouds with noise and can include incorrectly placed points, there are going to be additional issues, all of which need to be addressed:

- A point cloud consists of individual points, and objects represented through point clouds may need a lot of points for an accurate match. Expand this to an entire room, and the number can easily cross from tens- to hundreds of thousands. **Not only would comparing each point be a taxing task, the effects would be worse on a mobile device that does not have the amount of power that a desktop computer offers.** A balance would be needed that the point cloud remains dense enough for accuracy, but not too sparse that objects would no longer be easy to differentiate.
- **Two different scanned point clouds are not going to have an equal number of points, nor can individual points be directly matched with each other.** The initial scan is likely a thorough one of an entire room, while the second scan is likely just a part of the room. Points are not going to be placed at the exact same location either, with the room lighting, quality of the camera and its position affecting where a point is placed. These all add up to create a situation where it is impossible to match both point clouds fully one-to-one. This affects the accuracy of certain matching algorithms. One example is *iterative closest point*[10], which is a simple minimization algorithm between two different point clouds, but does not take inaccuracies into account.
- Another issue with the camera on a regular mobile device is the lack of depth perception, leading to outliers – points that are incorrectly placed on the scene. **Certain algorithms are susceptible to outliers and with a large enough set of them, will no longer be able to match properly.** While light detection and ranging (LiDAR) cameras could improve this [2], they are only used on some of the most modern smart devices, such as the newer Apple devices (iPad Pro 2020, iPhone 12 Pro) [3]. The likelihood of an AR specialized camera being adopted for the majority of regular smartphones is relatively unlikely and could take decades to happen.
- **The object being scanned can have an effect on the matching process based on its shape and material.** A room generally consists of sharp edges and corners, but can also include objects within it that are more rounded out. Glossy materials are reflective

[10] https://en.wikipedia.org/wiki/Iterative_Closest_Point

and can cause outliers, while matte monotone materials do not have any details to detect and place points on.

- **A point cloud being too symmetrical can cause mismatches.** One example is that with a rectangular object (a room), the algorithm might not be able to differentiate what is the floor, the ceiling, and where each wall is supposed to be. This increases the likelihood of matches where the point cloud is flipped over or upside down.

Knowing all of the potential issues that come with creating and matching point clouds, it is possible to start searching for suitable matching algorithms that address them.

2.1.2. Goal of the Algorithm

The end goal is to find a single *rigid* spatial *transformation* (see **Appendix I – Glossary**) that, by minimizing some error metric, aligns the first point cloud on top of the other. Doing so would require a rotation and translation that causes the first point cloud to overlay as much as possible on the second point cloud. A score metric should be used to determine if the found solution is better than previous ones.

Another relatively important aspect is to determine the ease of implementation and actual runtime speed, so algorithms that show better results should be preferred.

2.2. Super 4-Point Congruent System (S4PCS)

The S4PCS [4] algorithm, created in 2014, is based on the 4-Point Congruent System (4PCS) [5] process, which was made in 2008. On the same year as S4PCS, an alternative algorithm was made from 4PCS as well, called Generalized 4PCS (G4PCS). A year after, an algorithm that combines both S4PCS and G4PCS was made, called Super Generalized 4PCS. Finally, in 2017, a new algorithm was created, called 2-Points Normal Sets (2PNS). While all the other algorithms build on top of the original 4PCS, 2PNS uses parts of S4PCS, but otherwise has a different approach to finding a solution. Figure 1 summarizes all of these algorithms into one image.

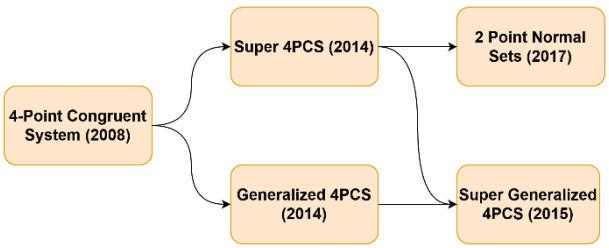

Figure 1. History of viewed algorithms.

The goal of these algorithms was to provide a relatively quick point cloud matching algorithm that is able to align two different 3D point sets, which would also be resilient to noise and outliers, even with small overlap. To do this, they find the best aligning transformation

according to the Largest Common Point set (LCP)[11] measure between a source point cloud P and a target point cloud Q. The problem is defined as follows:

Given two different point sets P and Q, LCP under δ-congruence solves for a subset P' of P, having the largest possible cardinality, such that the distance between $T(P')$ and Q is less than δ under an appropriate distance measure, T being a rigid transform.

In simpler terms, the goal is to find a transformation between two different point clouds, where the largest amount of points from the first point cloud are close enough to points from the second point cloud.

2.2.1. 4-Point Congruent System Process

The original 4PCS has a complexity of $O(n^2 + k)$, where n is the number of points in Q and k the number of reported 4-points sets. The algorithm is defined in several separate steps:

Selecting a Base

4PCS follows the RANSAC (see **Appendix I – Glossary**) approach of randomly selecting the initial base of elements from the first point cloud to compare against another base from the second point cloud. Figure 2 shows the end result. The first thing to do is to take a set of 4 coplanar points from source point cloud P as a base B. The first three points are randomly selected. The fourth point is selected so that they would form a wide base that is approximately coplanar. This approximation is due to there likely not existing a fully coplanar 4-point set, so a small amount of error is given as a limit when picking out the fourth point. The reason why a *wide base* is created by picking points far from each other is because it gives more stable alignments. However, for point clouds that only partially match together, if the points are too far apart, the selected points might not all lie in the overlap area. For this, an additional overlap fraction *f* can be used to estimate the maximum distance.

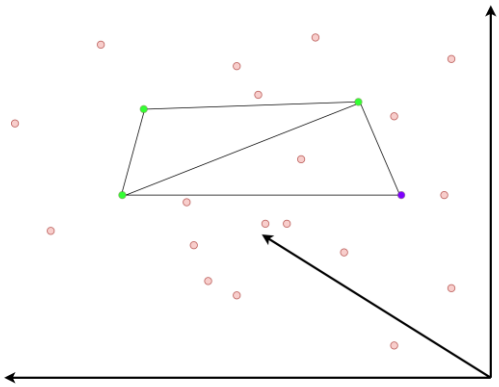

Figure 2. 4PCS plane in point cloud.

[11] After two different point clouds have been matched, LCP is the ratio of points within the first point cloud that are near any point in the second point cloud, within a given distance.

Defining Invariant Ratios

Once an approximately coplanar base is selected, the next thing is to define *invariants* for the base. These are values that can be calculated with the 4 points of the base that do not change after a transformation has been performed on the point cloud. They can be used to further constrain searches for bases in later steps.

On Figure 3, for a 4-point base $B = \{a, b, c, d\}$, an additional intermediate point e can be defined, where the segment ab intersects segment cd. With these values, 2 ratios can be defined as:

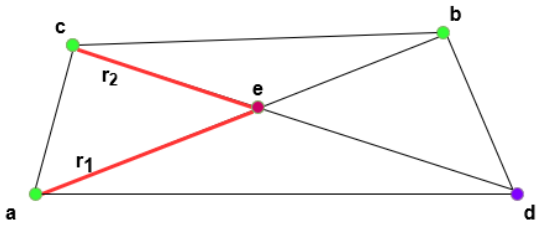

Figure 3. Base ratios.

$$r_1 = \frac{||a-e||}{||a-b||}, r_2 = \frac{||c-e||}{||c-d||}$$

Under rigid transformations, distances are also preserved. Therefore, the distances ab and cd are also used as invariants $(d_1,\ d_2)$.

Extracting Congruent Bases in Target Point Cloud Q

Now that all the necessary information is calculated (the distances d_1, d_2 and ratios r_1, r_2), the algorithm can extract all 4-point sets from the target point cloud Q that are approximately congruent with B, up to an approximation level δ. To do this, the algorithm goes through each pair of points $\{q_1, q_2\} \in Q$. For any pair there are 4 possible locations that an intermediate point, based on B, could be located at. Figure 4 shows these locations, which are calculated as:

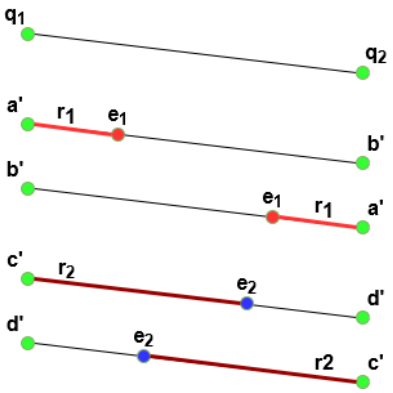

Figure 4. Intermediate points.

$$e_1 = q_1 \pm r_1(q_2 - q_1)$$

$$e_2 = q_1 \pm r_2(q_2 - q_1)$$

Figure 5 shows how any 2 pairs with their intermediate points match, one gotten with r_1 and the other with r_2 , are probably a 4-point set approximate copy of B. Because of noise though, intermediate points will likely not match up one-to-one, so there is a small amount of error for how far apart e_1 and e_2 can

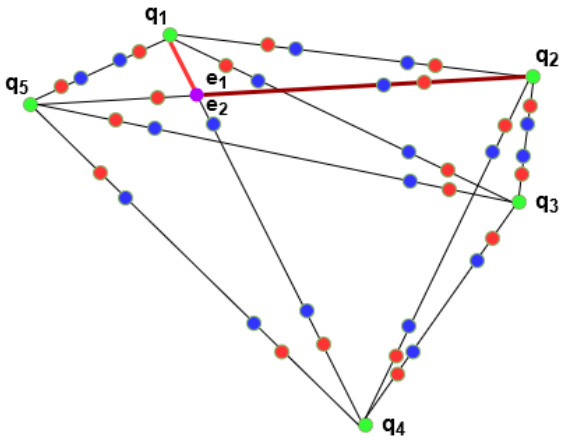

Figure 5. Matching intermediate points.

be. To allow quick queries of proximity points, an approximate range tree is built to put all intermediate points in it and query for all r_1 points that are in the δ-neighborhood of r_2 points. However, on its own, building and querying the neighborhood takes $O(n^2 \log n + k)$ time, where k is the total number of reported points. The entire extraction is converged to $O(n^2)$ by instead only searching for point pairs from Q that are d_1 or d_2 apart, up to a tolerance δ. Using the correspondence information between B and each found 4-point set, the best aligning transformation T_i (in least squares sense) is calculated.

Verification

The transformations still need to be verified if they are correct, so each transformation is separately used with source point cloud P, after which it finds how many points in the new alignment are closer than δ to some point in Q. This is performed in a randomization fashion, the algorithm used in this case is ANN (Approximate Nearest Neighbor). A constant number of points from P are selected, transformed using T_i, and for all the points, query for close neighbors in Q. If enough points matched, perform similar tests for the remaining points in P and assign an LCP score for T_i. The transformation with the best score is retained as T.

2.2.2. Improvements with Super 4-Point Congruent System

4PCS has 2 bottlenecks. First is having to exhaustively extract pairs from P in $O(n^2)$ time. The second is that a lot of unnecessary/invalid congruent sets are found, which later need to be removed in an additional verification step. This step takes $O(k_3)$ time, where k_3 is the number of actual 4-point bases from Q that are approximately congruent to the base B, assuming P does not have many copies in Q.

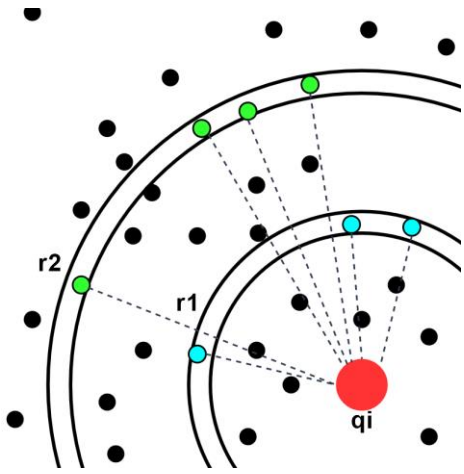

Figure 6. Ratio distances.

S4PCS was introduced as an improved algorithm that addresses these issues and improves the total runtime from quadratic time to a linear $O(n + k_1 + k_2)$, where k_1 is the number of pairs in Q at a given distance r and k_2 the number of congruent sets.

Extraction

To extract pairs more efficiently, a space partitioning strategy similar to an octree or k-d tree (see **Appendix I – Glossary**) was applied to the 3D grid of the point cloud. Instead of going through each point, for each point $q_i \in Q$, we can now define 2 distances r_1 and r_2. For all the points, it finds and indexes those points that are at distance $r_1 \pm \varepsilon$ and $r_2 \pm \varepsilon$, where ε is an error estimation (Figure 6). Both octree and k-d tree solutions would be subdivided whenever a new point is added. The extraction method used here would instead only start subdivision afterwards and pick areas that intersect between the sphere shells $r_1 \pm \varepsilon$ and $r_2 \pm \varepsilon$ (Figure 7 shows a 2D representation).

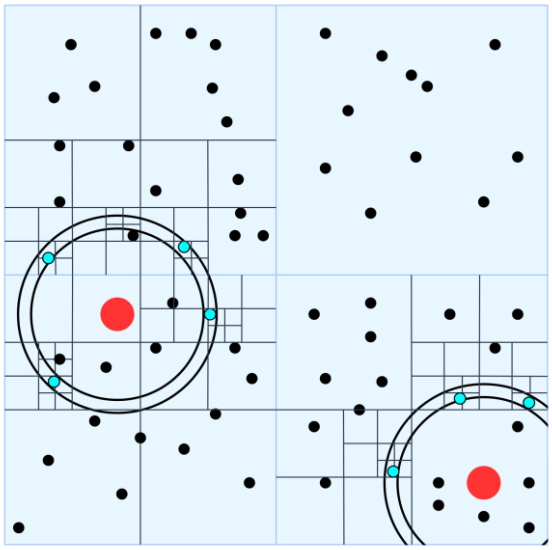

Figure 7. Finding suitable points (2D).

Verification

Many of the found congruent sets in 4PCS are congruent under affine transformations, but not under rigid transformations. Affine means that lines and parallelism are preserved, but not distance, when picking out a 4-point set. These would have to be removed in an additional step. For S4PCS, rigid transformations preserve the angle and distance between two intersecting pairs, which can be used as an additional constraint, looking only for source bases that share the same angle α. Because of this, an additional step is taken here to verify which sets have fulfilled this condition, while ignoring the rest.

2.2.3. Side Note – Generalized and Super Generalized 4PCS

Around the same time as S4PCS, G4PCS [6] was created with a different approach to improving results. Up until now, all 4-point bases are planar, meaning that in accordance to some error, all 4 points can be shown on a 2D plane. The generalized version of 4PCS introduces a more general type of 4-point base that removes this planarity constraint.

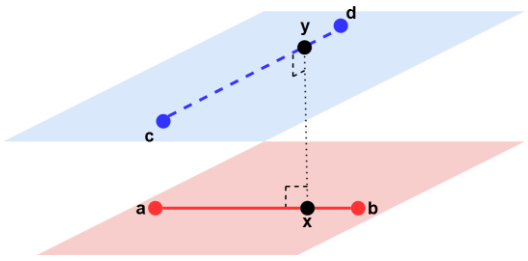

Figure 8. Generalized 4-point base.

The same concept of a 4-point set can be taken: $B = \{a, b, c, d\}$. However, this time, segments ab and cd do not lie on the same plane. They instead have a 3D view angle where they appear to overlap (Figure 8). The term used for this in the original paper is called a *3D intersection*. The two pairs are connected by an additional line (d_3), placed in such a way that its size would be as short as possible, while also remaining perpendicular to both pairs.

The time reduction on this comes from larger intersection distances resulting in less congruent bases being found, usually exponentially. This means less transformations to check, reducing runtime from the verification stage.

Ultimately, SG4PCS [7] takes both the elements of S4PCS and G4PCS, combining them together to further improve runtime. Due to S4PCS not taking non-planar pairs into account, additional conditions need to be addressed. One example is that the distance between the two pairs needs to be addressed differently than with S4PCS. Whenever $d_3 \neq 0$, the points for querying and points found do not lie on the same plane. The intermediate point of an overlapping d_1 segment, x, is somewhere on the circle that is centered at y, with distance d_3, as shown on Figure 9. An approach here would be to move in circular steps to find overlapping areas. This increases runtime, but through other optimizations, the overall runtime would still remain lower than before.

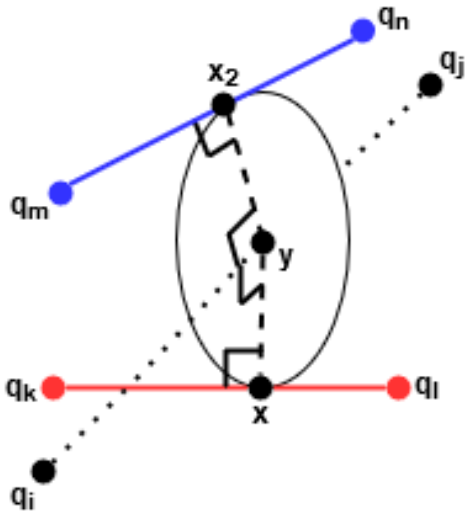

Figure 9. 2 pair intersection.

SG4PCS was considered as a possible algorithm to use for the point cloud matching process. The problem is that it consists of many parts that need to be taken into account during its creation. The complexity of understanding exactly how to recreate each part made it take more and more time to implement, until eventually, the algorithm within the next subchapter became a more preferable choice.

2.3. 2-Point Normal Sets (2PNS)

The 2-Point Normal Sets [8] algorithm was made in 2017. Like S4PCS, it is a RANSAC-based point cloud algorithm that tries to find the best LCP value to decide the best transformation. The main difference is that it instead relies on pairs of 2 points and their normal vectors to compare against. This leads to less combinations, because there is no need to match pairs of points together into 4-point sets like with S4PCS, making the algorithm implementation less

complex as well. Other areas, such as the pair extraction, are the same as S4PCS though, which is why 2PNS is seen as an advancement of S4PCS.

One thing to note is that point cloud scans do not have normals, which will need to be approximated with another method. In this thesis, the *PlanePCA* (see **Appendix I – Glossary**) method is used to calculate normals from point data. This was considered the most optimal solution according to *Klasing et al* [9]. A caveat of PlanePCA is that the direction of the normal is not determined, giving two possible solutions as the normal vector. For performance reasons, only one of the solutions is taken as the answer. 2PNS will not work when normals are not properly estimated, which can happen with a sparse point cloud or if the said point cloud is dominated by sharp edges and corners. The former was mentioned in chapter **2.1.1. Issues with Point Cloud Generation** and will not be a problem, since the generated point clouds allow points to be relatively close to each other. The latter may potentially be an issue when scanning rooms, but is determined in chapter **3. Experiments**.

For each RANSAC iteration, the algorithm takes a random pair of points A, B and their normals n_A, n_B from the source point cloud P. It is now important that other pairs taken from the target point cloud Q can be aligned with the source pair by a rigid transformation.

2PNS approaches pair extraction from S4PCS slightly differently. The latter extracts pairs with a distance $d = ||A - B||$ and their angle $\Theta = \angle(n_A, n_B)$ (variants of 4PCS can use point normals to improve performance). When combining pairs together to create planes to compare against the source point cloud base, they might not have normals that match closely enough. This leads to pairs that contain instances unalignable by rigid transformation. These bases would need to be removed through verification. 2PNS avoids this by searching for pairs that fulfill 3 additional angles in addition to d and Θ. Figure 10 shows these angles as γ_1, the angle between the line segments first point and normal, γ_2, which is the same but for the second normal, and γ_3, the angle between the normals projected onto the plane orthogonal to the line segment. This means that γ_3 is the angle between the *rejections* (see **Appendix I – Glossary**) of AB with n_A and BA with n_B. Angles are preserved under rigid transformations, so the given 5 invariants are used to extract congruent pairs in the target point cloud Q, similar to S4PCS. An additional tolerance is added for situations where noise and outliers are present, which can vary between 5 to 20 degrees for best results.

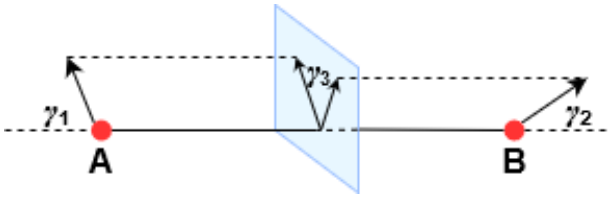

Figure 10. Gamma angles.

Figure 11 shows the process of finding the rotation between the pair AB taken from P and an arbitrary found pair $A'B'$ from Q.

(a) The initial states of both the source pair (v_1) and some target pair (v_2).

(b) The vectors of both pairs (AB, $A'B'$) are initially normalized and centered to the origin, using their centroids.

(c) The rotation R is found as $R_\alpha \cdot R_\beta$. The first half R_α estimates the alignment of the vector pairs from P and Q around the perpendicular rotation axis (cross product between v_1 and v_2, pitch and yaw) with the rotation angle ($\cos^{-1}(v_1 \cdot v_2)$).

(d) R_β determines the alignment of the normal vectors and can be found by rotating an angle beta around the axis v_2 (roll). This angle is found between the rejections (see **Appendix I – Glossary**) of the normal vectors n_A^* and n_A'. against v_2. The first normal n_A^* is the normal from n_A after it is rotated by R_α. The second one n_A' is the normal for the target pair. Once the rotation is known, the translation is estimated from the distance between the centers of both pairs. The rotation and translation are used to compute the LCP.

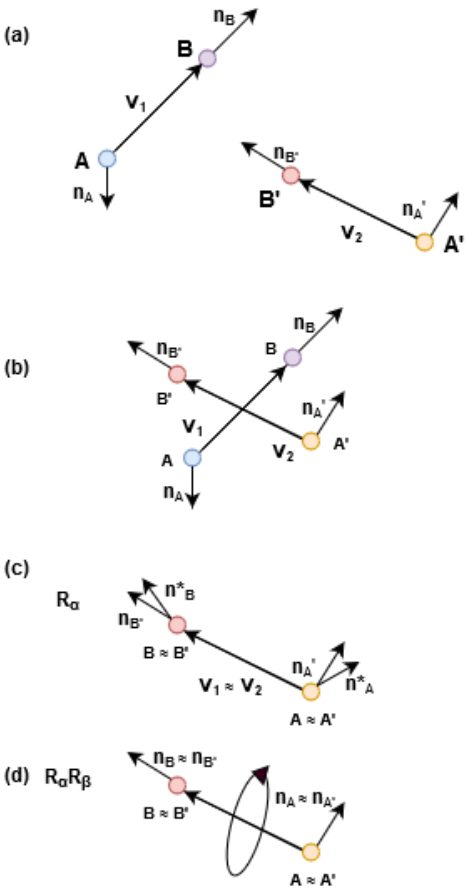

Figure 11. Pair matching.

2.4. Modifying 2PNS Pair Extraction

When using and modifying the sample code used for the project[12], something unique was noticed when saving point clouds. Regardless of the position of the mobile device scanning the environment, the forward and right axes of the 3D environment are set based on the initial rotation of the device itself. However, the up (y) axis always remains consistent with the real world "up axis". This means that the scans will always be upright, so only a rotation around the y-axis and a translation T is needed to correctly match the two different point clouds.

Figure 12 shows the general process on how the modified pair extraction would differ from the original algorithm:

[12] https://github.com/Unity-Technologies/arfoundation-samples

(a) The initial pair of points from the source point cloud is still taken the same way. Additionally, the height between the 2 points y_d and the width r are taken as separate values. For consistency in the next step, point A should have a higher y-axis value than point B.

(b) During pair extraction, 2PNS would originally find all points within sphere radius distance to the chosen point Q_i, where the radius is about the length of AB. Because of the y-axis constraint, the location at which matching points can be found consists of two cylindrical shells (instead of one spherical shell), since the algorithm should look for pairs that only require a rotation across the y-axis.

(c) Like the original pair extraction, a small amount of error will need to be taken into account, as it is unlikely to find points that are an exact amount of distance away. It is extended to the annulus[13] of two cylindrical shells.

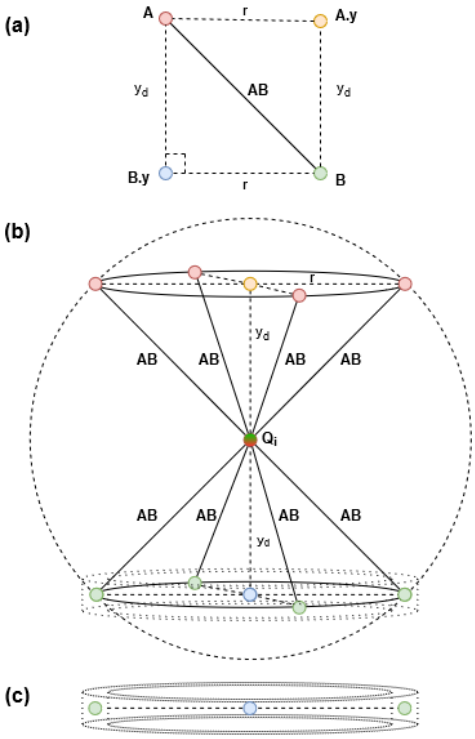

Figure 12. Optimized pair extraction.

2.5. Adding a Pair Limiter

Due to the random nature of a RANSAC iteration, there is a non-trivial chance that a pair of source points will be picked in such a way, that a lot of potential matching pairs will be found. One iteration could have from a few dozen to over 5000 potential pairs. Having to go through a large number of pairs is time consuming and causes the performance to suffer a lot. This is why a limiter was added as a possible solution, where for each RANSAC iteration, only a maximum of 100 candidates will be visited. These visits are spread out across the entire list of candidates to prevent picking out potential pairs found from only the first few pairs.

[13] The region lying between two concentric circles

2.6. Sampling Point Clouds

The final key element to reduce runtime is by resampling both the source and target point clouds whenever the matching process is started. This is done by creating uniform point clouds of the originals (Figure 13). A preset value is used to determine how far the points in the uniform cloud should be from each other. After that, each point in the original point cloud is visited and a small subset of points far enough from each other is extracted. This reduces the amount of points needed to be checked from around 10,000 points to less than a thousand, improving runtime and preserving the general appearance of the point cloud.

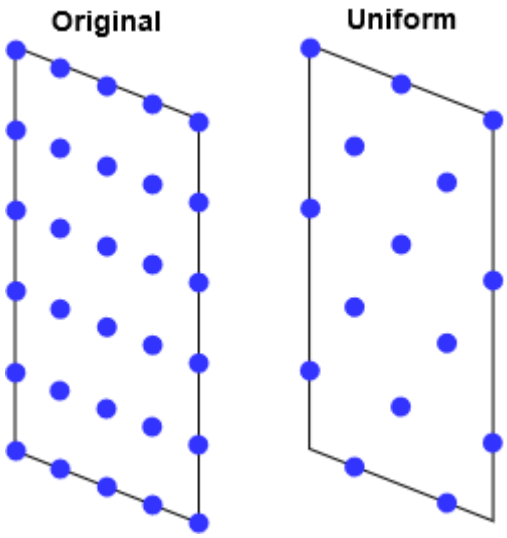

Figure 13. Uniform point cloud.

For the target point cloud, an additional random sampling method is applied to further slightly reduce point count. This also means that the matching process will have slight randomizations on how much can be matched against each time the algorithm is run.

3. Experiments

A point matching algorithm may work well in theory, but it is also important to make sure how well it performs in a variety of different situations. The context that Mobi Lab has offered is using these algorithms to match point clouds of interior spaces / rooms of museums. The first user would be the app administrator (e.g. museum curator) who maps the entire area as a point cloud and saves it. The second user will likely scan only a part of the room before it is matched against the original point cloud. The second user can end up in situations, where the chosen algorithm might not be able to match two different point clouds together correctly, which is why this needs to be tested.

Chapter 3.1 will briefly explain the application that was created and used for testing. All of the following subchapters will then go into detail on the types of tests done, with their runtime, LCP values and match quality. After that, suggested and implemented improvements with further experiments are described, until lastly, the thesis presents the results when the final version of the algorithm was used with a mobile device.

3.1. Application

The application (folder "Build" in the accompanying files) was created using Unity ARFoundation[14], a framework that uses core AR features of different platforms. It can release separate AR application builds for all of them without having to create the entire code from scratch for each platform. In the case of mobile devices, it uses ARCore for Android devices and ARKit for iOS devices.

The developed application initially starts scanning the environment to find *feature points*[15] and makes them persist even when the device is no longer pointing towards them. An octree has been added to prevent the addition of points too close to existing points. The number should be reduced whenever possible, as it has direct impact on overall performance. It does this by iterating through every point that has been found on the current frame and using their location to find any nearby ones within the octree. If a point is found based on a preset range, it is not accepted. Otherwise, it is stored and also added to the octree. This method was chosen over a grid, since the latter can technically still place 2 points very close to each other, as long as they are in separate grid cells.

[14] https://unity.com/unity/features/arfoundation

[15] Points added to areas where sharp edges or sudden color changes are detected. E.g., the edge of a table.

Once the initial admin user has scanned the room, the point cloud can be saved as a binary file, where all the point coordinates are converted into a byte stream. The main test case is to use this saved text file and match its contents against the newly scanned point cloud.

3.2. Experiment 1 - Rectangular Box

The first test was to create some form of point cloud data with no noise or outliers on it, that could be used to verify if the algorithm is functioning. For this, two identical rectangular boxes with different y-axis rotations and shared x-axis rotations (the box is tilted) were created as a test case. The rectangular point cloud has an outline made of roughly 15,000 points, where each point is away from a neighboring point by 0.1 units.

A note here is that all these tests were done on a computer. Mobile tests were done after all improvements were implemented to ensure that it is ready to use (chapter **3.5. Experiment 4 – Mobile Tests**).

Test 1 – Pair Limit

The first test was to determine the need for having a pair limiter, as mentioned in chapter **2.5. Adding a Pair Limiter**. By verifying if finding a large number of pairs can drop performance, several tests were done with the pair limiter enabled and disabled.

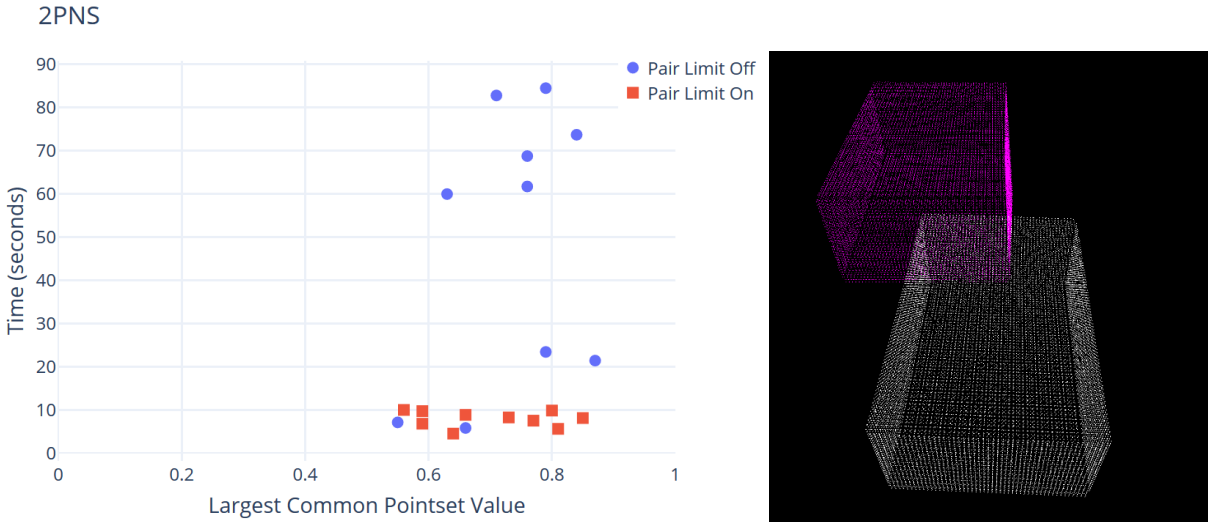

Figure 14. Rectangular box matching.

On Figure 14, the x-axis of the graph shows the LCP values, which fluctuated from 0.5 to 0.9. Most of the results visually looked good though, differing by a few degrees of rotation or translation. The reason behind it is likely the way point clouds are being sampled, as mentioned in chapter

2.6. Sampling Point Clouds. By randomly picking out a small subset of points, the maximum possible LCP value with the best match will fluctuate on each separate iteration.

As for time performance, it can be pretty clearly seen that while the pair limiter is on, the matching time remains relatively stable, never going over 10 seconds. When the pair limiter is off, a few cases have about the same performance, but there are also cases where the time taken increases multiple times. In these situations, some source pairs that were picked out had over 5000 potential pairs within the target point cloud, when under normal cases this number was below 100 pairs. Considering that the LCP and actual match quality is not noticeably affected by having the pair limiter on, all following tests will be leaving it on by default.

There is another issue with this algorithm that does not show itself within the graph. For all of these results, the rotation was wrong. While the two rectangle point clouds match relatively well, the source point cloud is turned upside down in every case. This means that every time the algorithm is run, any objects would be nowhere near their intended locations. The symmetrical nature of the current point cloud was a potential issue directly mentioned in chapter **2.1.1. Issues with Point Cloud Generation**, which is what the next test will address.

Test 2 - Y-Optimization

The optimized solution can only do rotations along the y-axis. This in theory should prevent a point cloud from being turned upside down, which happened during each run with the previous test. Figure 15 shows the performance between the 2PNS algorithm and the y-axis rotation optimization (Y-Opt).

In terms of match quality, it can be seen that all the LCP values obtained from Y-Opt are around the same LCP range as the original algorithm, but at the same time, have a much smaller average time. This is likely because the search range of the optimization for potential pairs is significantly smaller compared to the original.

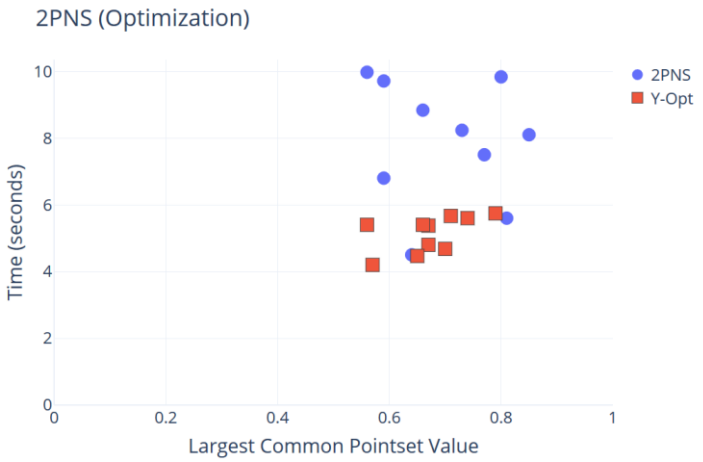

Figure 15. 2PNS Y-optimization.

As for the actual rotation, all of the results still had visually good results. Since both point clouds are equally tilted across one axis and only have different y-axis rotations, it can be concluded that the rotations are actually correct.

One final thing is that while all the original 2PNS results are “wrong”, these are done on a perfectly symmetrical point cloud. This might not happen to such an extent with real world data. At the same time, however, since it can allow rotations across the x- and z-axes and takes more time to perform, it is a better choice to stick to the optimized version of the algorithm.

Test 3 - Delta Value

For every 2PNS pairs, 3 gamma rotation values are calculated. To decide if these gamma values in a target pair are suitable enough against the chosen source pair, delta error is used. When comparing gammas from either pair, if the deviation is more than delta, a pair is not accepted. With the rectangular box, point cloud surfaces are straight, meaning generating normals through PlanePCA will give relatively stable results. However, this might not be the case with actual room results, plus there is room for improvement with the rectangular cube. Figure 16 shows the different results, when Y-Opt is given a delta value of 5, 10, 15, 20, and completely off.

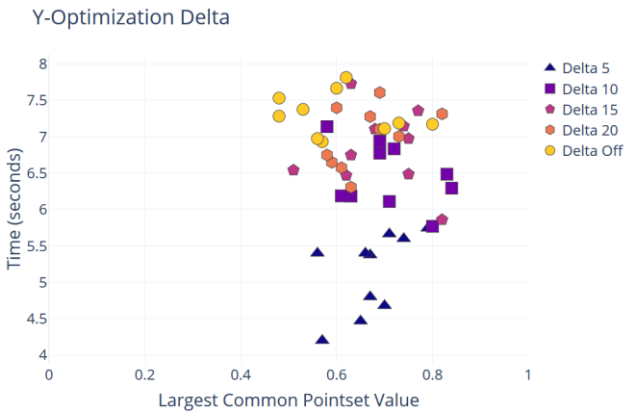

Figure 16. Delta angles.

As can be seen from the results, the LCP values vary quite a bit. The average time taken for the algorithm to finish tends to increase as the delta value gets larger. This makes sense, as the more lenient this error checking method becomes, the more possible pairs are accepted within the pair extraction phase. If it were completely disabled, then the only check done is to make sure that 2 points are at a suitable distance away from each other.

One other noteworthy thing is that when delta was disabled, there was a significant chance (around 1 in 4) that the translation was no longer good. This is also indicated in the figure by “Delta Off” having more lower LCP results than any of the other results. The possible reason behind this is the increasingly larger number of pairs being selected, causing the pair limiter to have a far smaller likelihood at picking out a good result.

Apart from this though, the other delta choices had relatively similar LCP matches, but cannot be fully determined if they are suitable based on one perfectly generated point cloud alone. Unlike the previous 2 tests, this requires more testing with a real-world dataset to determine if LCP is affected.

3.3. Experiment 2 – Gallery Pallas

Once the Rectangular Box test case experiment had been done, the next step was to create point cloud scans of a real-world environment to see how the algorithm holds up with noisy data. Gallery Pallas[16] was chosen as the location due to having relatively large and distinguishable areas when creating a point cloud, which helps in the matching process. Two locations were selected to make point cloud scans out of: the main exhibit on the first floor and a much smaller exhibition room in the building's basement.

3.3.1. First Floor Exhibition

The first-floor exhibition room (Figure 17) is a relatively rectangular area with around 28 meters in length and 8 meters in width. There are pillars within the room in non-symmetric locations, as well as an art exhibit set up at the time of the point cloud experiment. This would suggest that the scan would be relatively unique and help during the matching process.

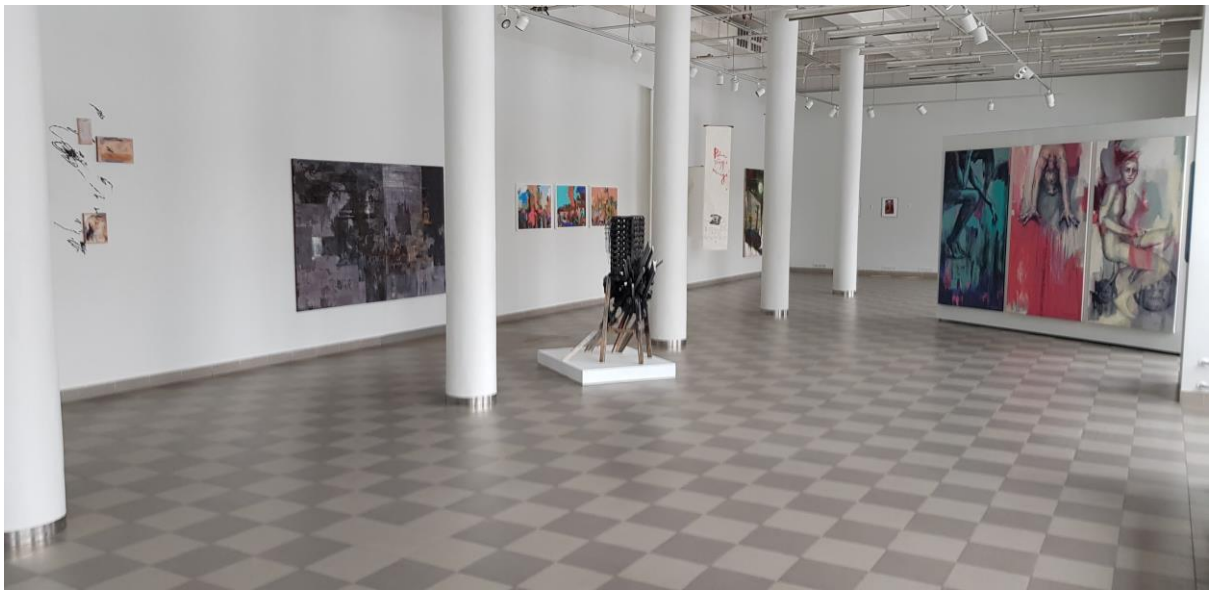

Figure 17. Gallery Pallas, first floor.

However, an unexpected limitation with the point cloud detection came up. When walking for long distances, the point cloud itself started to move along with it, causing the results to compress together. Trying to get a proper full scan of the room required too much careful movement, which would definitely end up being ignored in a real-world use case.

[16] https://pallasart.ee/en/pallas-gallery-and-exhibits/

There are two possible reasons as to why this could have occurred:

- The room itself is too large – There is a possibility that the point cloud scanning has non-modifiable hard limits on how wide a point cloud can be. This could be due to internal accuracy issues.
- The floor could have caused the point cloud to shift its position – The floor consisted of black and white square tiles in a symmetric pattern. It is possible that, through walking a long distance and looking at the floor, the application interpreted that the point cloud was no longer accurately positioned. Therefore, it offset it and "fixed" the issue.

A second issue that came up is the amount of points for the point cloud itself. Preliminary tests have been primarily done with around 10,000-15,000 points, but this particular one took over 30,000 to create. In addition to taking a long amount of time to simply load the point cloud in, the used mobile device (see **Appendix II – Mobile Device Hardware**) used took a heavier performance hit as well. This can be rectified by simply not visualizing the point cloud, but then the user would not get any feedback on whether or not they are scanning the environment.

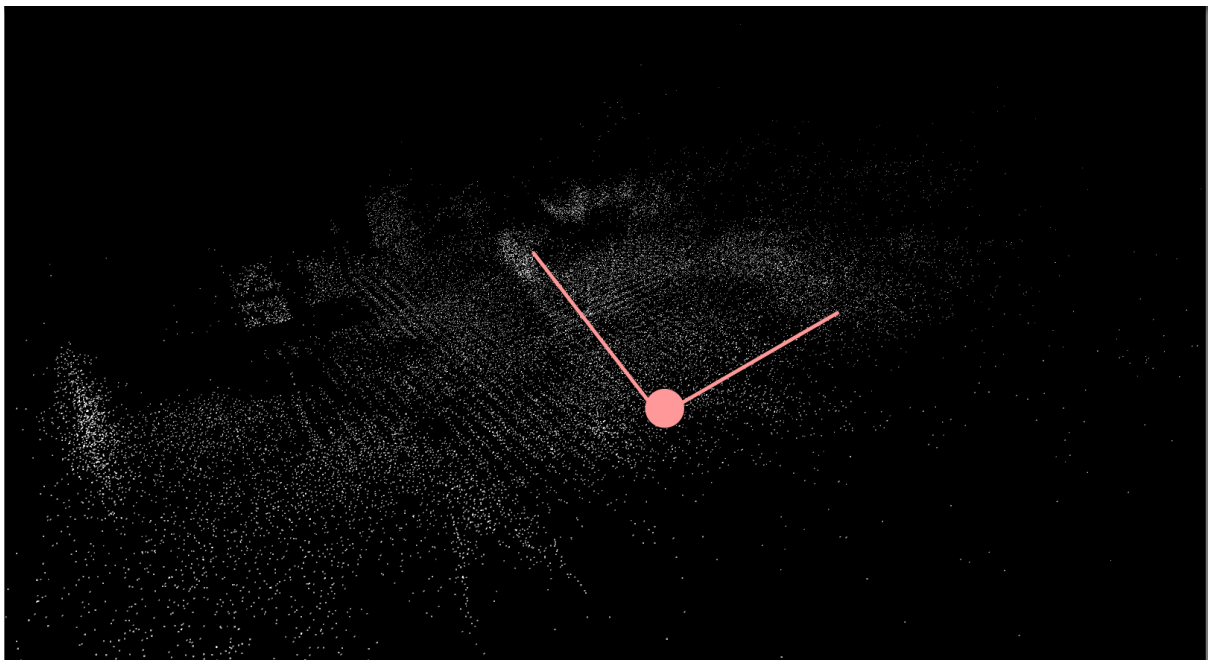

Figure 18. Point cloud of Gallery Pallas, first floor.

Figure 18 shows the point cloud representation of the first-floor exhibition, with the camera location of Figure 17 outlined in pink. The area outside of the camera view looks relatively stable, with several areas where the point cloud scanner detected paintings. Everything after that is no longer clear though, as the point cloud started moving alongside the mobile device. There should be a noticeable location for a large painting, with three smaller ones on the right side, but is not clear within the point cloud image at the outline location.

3.3.2. Basement Exhibition

The basement exhibition area of Gallery Pallas is considerably smaller in size, roughly 10 meters in length and 4 meters in width. The flooring is uneven, the walls are all white with no patterns (meaning they will not be detected), while the pictures placed around the room are not symmetrically aligned. These conditions should also theoretically allow for good point cloud matching. The resulting point cloud (Figure 19) looks considerably better compared to the first floor point cloud. While the detected points against walls / floors are still noisy, it is also much easier to recognize where the pictures are located based on them.

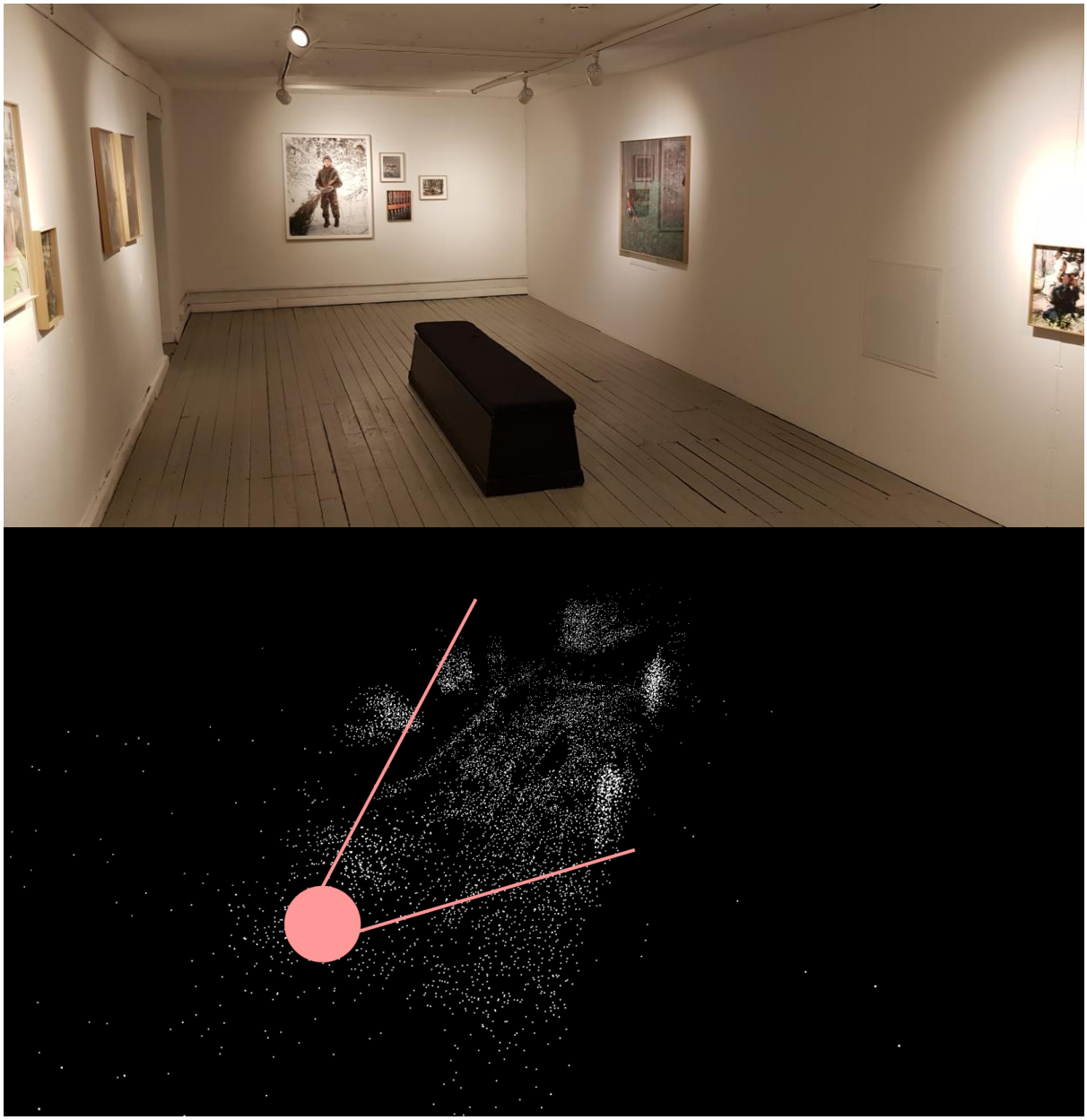

Figure 19. Gallery Pallas basement floor exhibition.

To confirm if the algorithm is working as intended, two different point clouds of the same room were scanned for testing purposes. As determined with the Rectangular Box experiment, the pair limiter will be enabled since it reduces runtime without noticeable penalties on accuracy. Only the optimized version of 2PNS will be used, as the original algorithm can do incorrect rotations with the x- and z-axes. As for the delta value, this initial test will still use the same delta values as with the Rectangular Box, to see if LCP values are affected with a different point cloud.

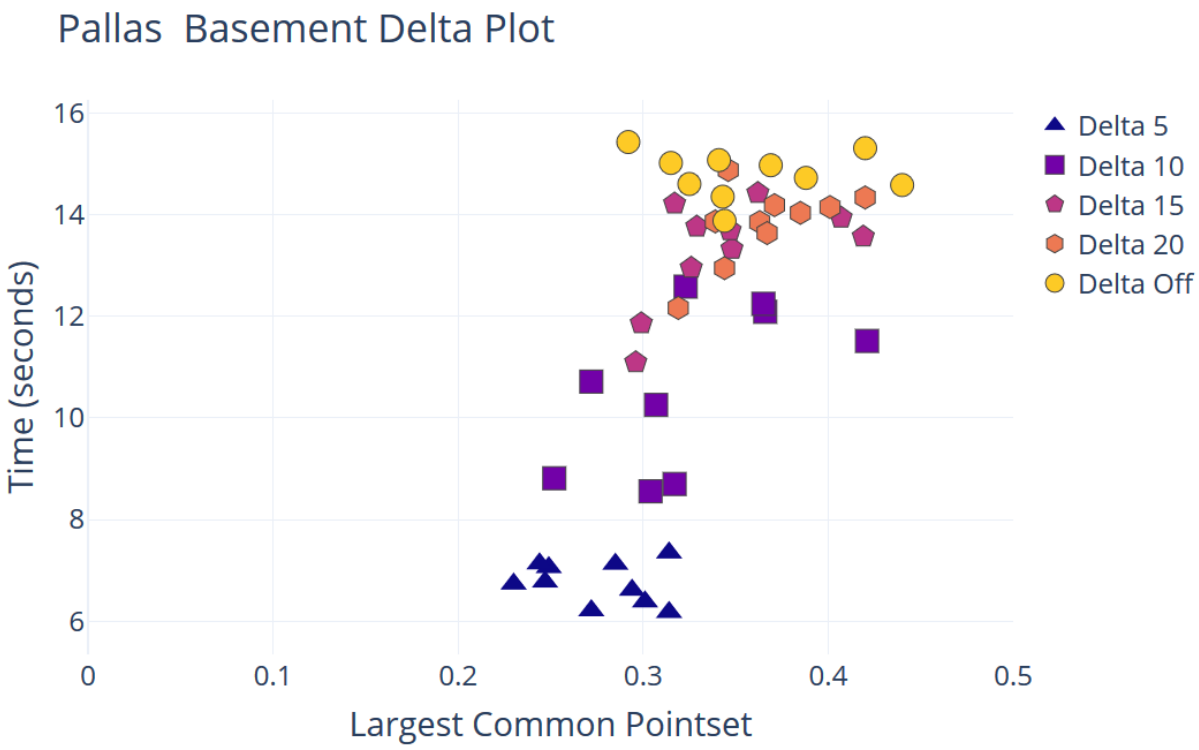

Figure 20. Basement full scan results.

As can be seen from Figure 20, a correlation can be generally drawn that when the delta error is increased, LCP values increase on average as well as the time taken. With a delta of just 5 degrees, the algorithm might perform faster at an average of 7 seconds, but the noisiness of the graph also means that better points with worse normals likely never end up being picked out. A delta of 10 degrees further illustrates this, as it now has more both cases that are still generally low in LCP with worse performance, as well as cases where it managed to find good results at the cost of more time taken. Having the delta at 15 degrees, 20 degrees or disabled all give relatively similar LCP values, with the latter ones taking slightly more time.

However, these results cannot be taken at face value. Even if LCP values might be larger on average, there is a possibility that the matching process can find an incorrect rotation and translation to have a better LCP value. For this, Table 1 has been defined with 3 types of

matches: a **close match** means that either the rotation is off around 6-9 degrees, the translation by a noticeable amount, or a combination of both. A **great match** is stricter at around 2-5 degrees of rotation and a smaller amount of translation error. A **perfect match** has a rotation error of up to 2 degrees, with any differences between two different point clouds unnoticeable.

Table 1. Pallas basement full scan match quality.

Delta (10 runs)	**Average Time (sec)**	**Average LCP**	**Bad Matches**	**Close Matches**	**Great Matches**	**Perfect Matches**
Delta 5	6.76	0.275	8	2	0	0
Delta 10	9.55	0.293	6	2	1	1
Delta 15	13.29	0.345	3	4	2	1
Delta 20	13.81	0.366	4	2	4	0
Delta Off	14.79	0.358	4	2	2	2

With a delta of 5 degrees, most of the tests did not have good results. Either the rotation was off by 20+ degrees / flipped by 180 degrees, or the translation was too far off. A delta of 10 degrees gave better results, but any **bad matches** had similar results to the previous test case. Because of their relatively low **good match** rate, later test cases will no longer be using these degrees.

With the delta at 15 degrees, 20 degrees or disabled, the matching quality is generally around the same region. All great matches share slight errors in either the rotation or translation with close matches having higher error values. In terms of bad results though, they all shared consistent issues – either the translation was way off or the rotation was over 180 degrees flipped. The LCP values were also comparable to close and great matches as well, making it really difficult to extract good and bad cases based on this value alone.

There are 2 potential issues that can cause this to happen: The noise of the point cloud or the scanned area still having too much symmetry. The noise of a point cloud can be an issue because the original algorithm can have hard limits on noise resiliency. This combined with PlanePCA not being able to accurately generate normals for a noisy point cloud makes it more difficult to find good pairs to match against each other. For symmetry issues, while the exhibition pictures were placed in various locations around the room, the two opposing short walls have photos relatively placed in the middle. Combined with noise, the matching process may find it better to flip the room around instead.

3.3.3. Basement Exhibition Extended Tests

To verify if point cloud symmetry is really the issue for the optimized algorithm, several additional scans needed to be done. The room has 2 long walls and 2 short walls. By not scanning specific walls, an interpretation of a real-world case would be made, where a wall would not have anything to scan and therefore would not have any points placed on it. Three additional point clouds were made: one of the longer walls not being scanned, one of the shorter walls not being scanned, and one where both a short and long wall are not scanned.

Test 1 – Short Wall Not Scanned

The following image (Figure 21) shows one of the scans made of the basement again, with the wall in the frontmost area not having been scanned. Figure 22 and Table 2 afterwards show the results with all algorithms.

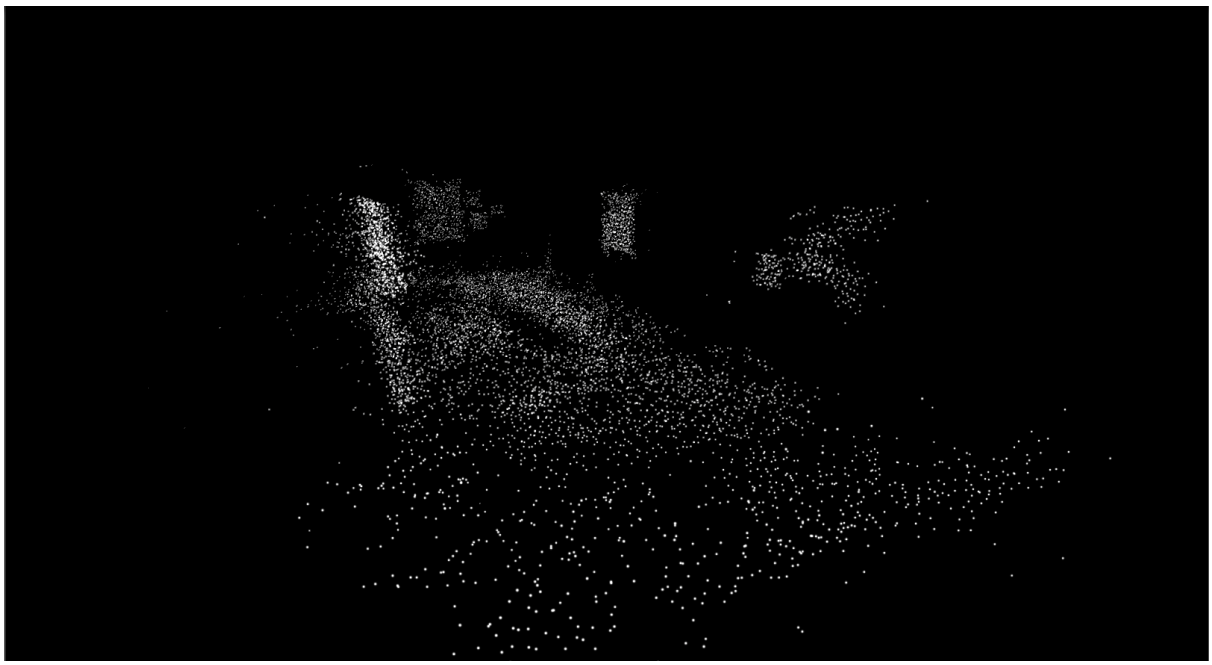

Figure 21. 2 long, 1 short wall scan.

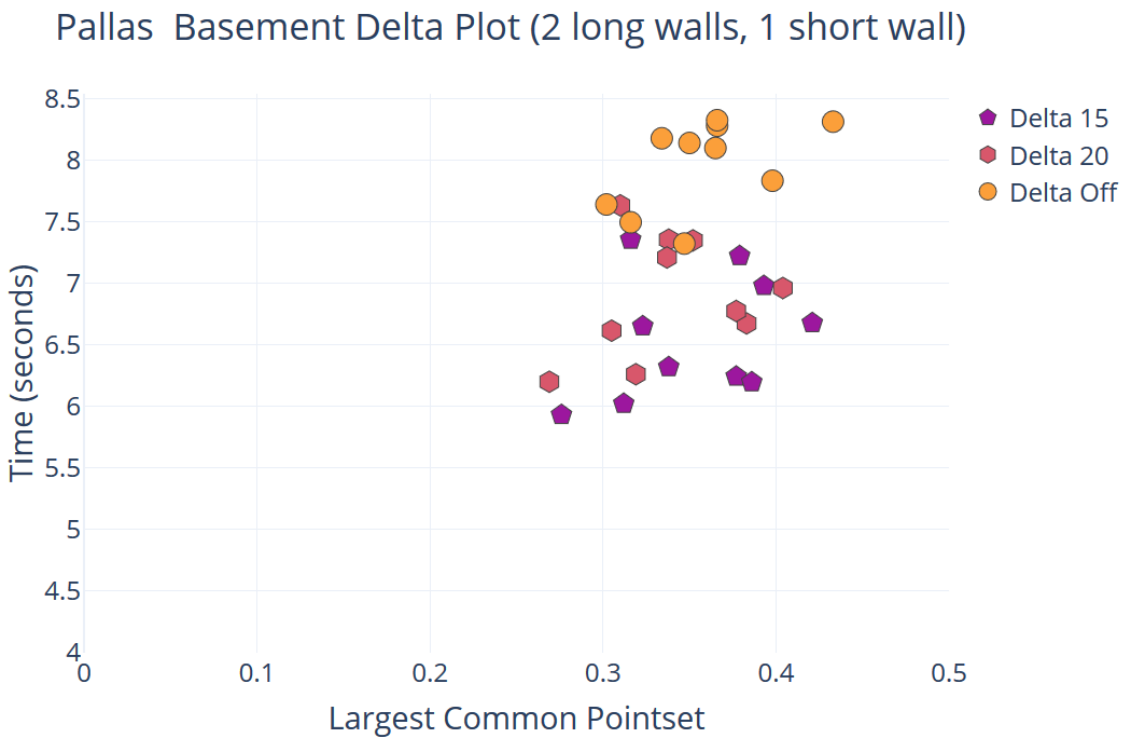

Figure 22. 2 long, 1 short wall LCP results.

Table 2. 2 long, 1 short wall match quality.

Delta (10 runs)	**Average Time (sec)**	**Average LCP**	**Bad Matches**	**Close Matches**	**Great Matches**	**Perfect Matches**
Delta 15	6.563	0.352	3	1	4	2
Delta 20	6.907	0.339	5	1	3	1
Delta Off	7.967	0.358	0	2	8	0

The results have a lot of variance. In terms of LCP, all 3 deltas showed general consistency, where the higher the value, the more accurate the match. In terms of actual matching quality, the delta being enabled or disabled gave rather different results.

With delta at 15 and 20 degrees, great matches usually had issues with translation being slightly out of place. Close matches had a combination of rotation and translation error. Bad matches sometimes were caused by a really bad rotation or translation, but also had a lot of times when the rotation was flipped over again, showing that having one short wall less to scan may not necessarily be enough to prevent such cases from happening.

Delta being completely disabled gave the most interesting results though. While there were no perfect cases, neither were there any bad ones. All cases shared the exact same issue, the translation was moving off towards one specific direction, with it being worse on close matches. This could be an implication that the uniform and sampled point clouds were generated in such a way that it might not be able to find a perfect match, although since delta 15 and 20 were able to find such cases, it is more likely that the issue is not being able to randomly pick out the best results due to too many pairs to pick from.

Test 2 - Long Wall Not Scanned

With this test case, one of the longer walls was not scanned (Figure 23). Considering the larger amount of area that is not covered, it could theoretically provide better results.

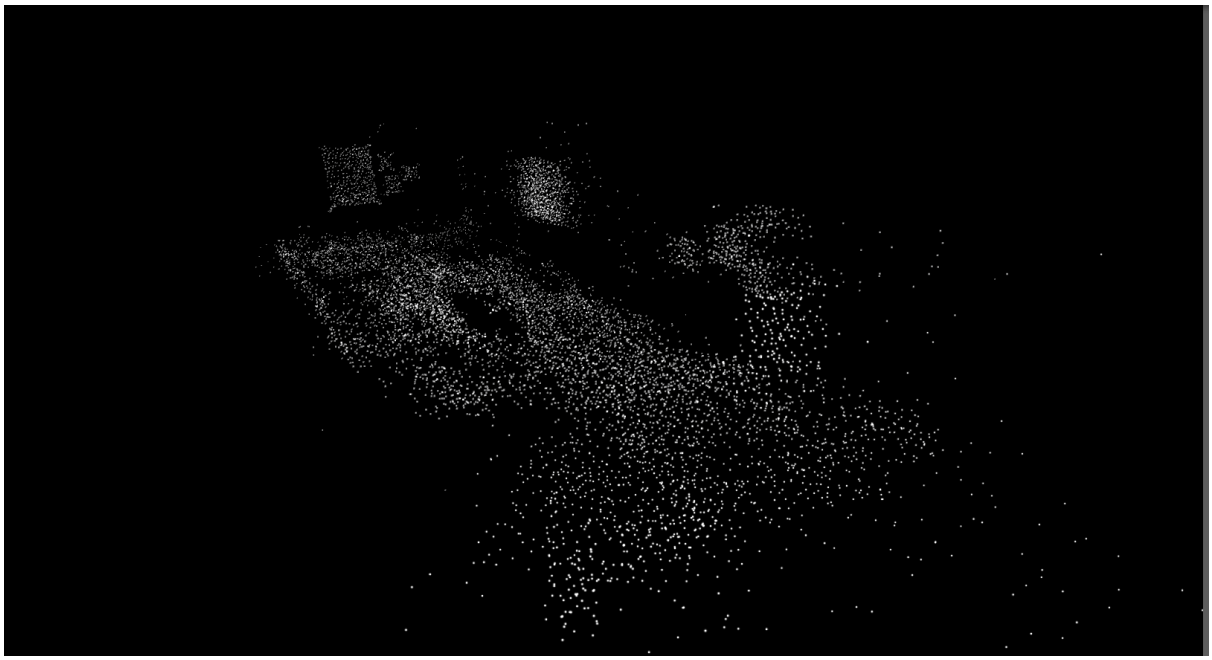

Figure 23. 1 long, 2 short wall scan.

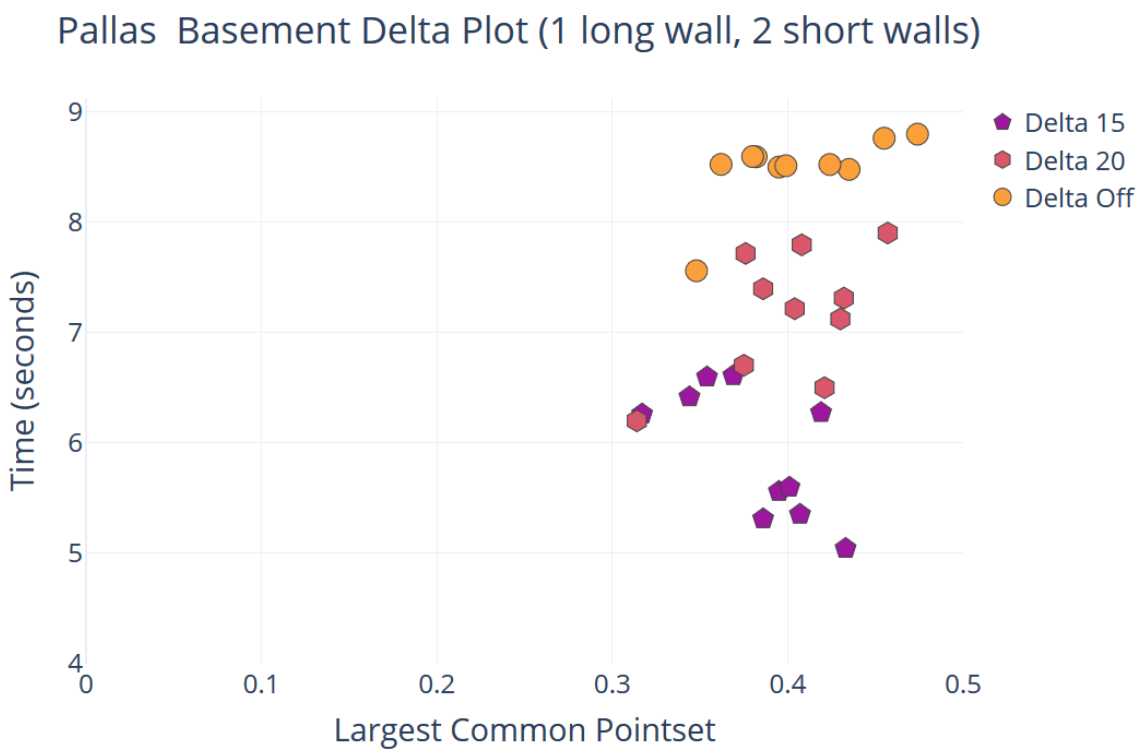

Figure 24. 1 long, 2 short wall LCP results.

Table 3. 1 long, 2 short wall match quality.

Delta (10 runs)	Average Time (sec)	Average LCP	Bad Matches	Close Matches	Great Matches	Perfect Matches
Delta 15	5.902	0.383	2	6	2	0
Delta 20	7.184	0.401	4	1	2	3
Delta Off	8.481	0.405	4	3	2	1

At first glance, it appears that the results based on LCP are far better than with the previous test (Figure 24), but these are very deceptive results. In terms of actual matches (Table 3), the results are similar Great matches have minor rotation or translation issues, close matches more so, and bad matches even worse, with some 180-degree rotations as well.

The problem is that some of the bad cases have LCP values comparable to great or perfect matches, making it really hard to differentiate them from the graph. Not to mention, finding a bad match with a high LCP value also means that a good match with a lower value is never going to be picked out. Noise can again be seen as a potential issue here. The short wall removed during the previous test has a very noisy scan of the picture, which is why removing it may have improved results overall. In this case, that wall remains and a significantly larger

wall is removed, making it more likely that this scan is far noisier and results in higher LCP values. The LCP check keeps finding outliers as “good” matches.

Test 3 - Two Walls Not Scanned

Finally, a third scan (Figure 25) was used where both the long and short walls were ignored.

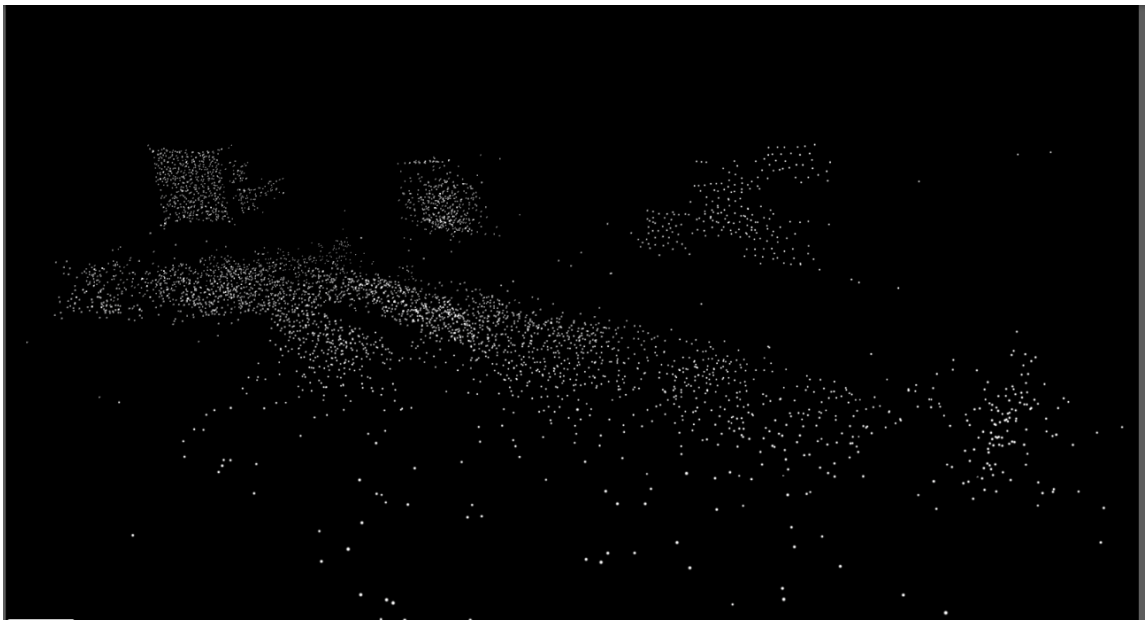

Figure 25. 1 long, 1 short wall scan.

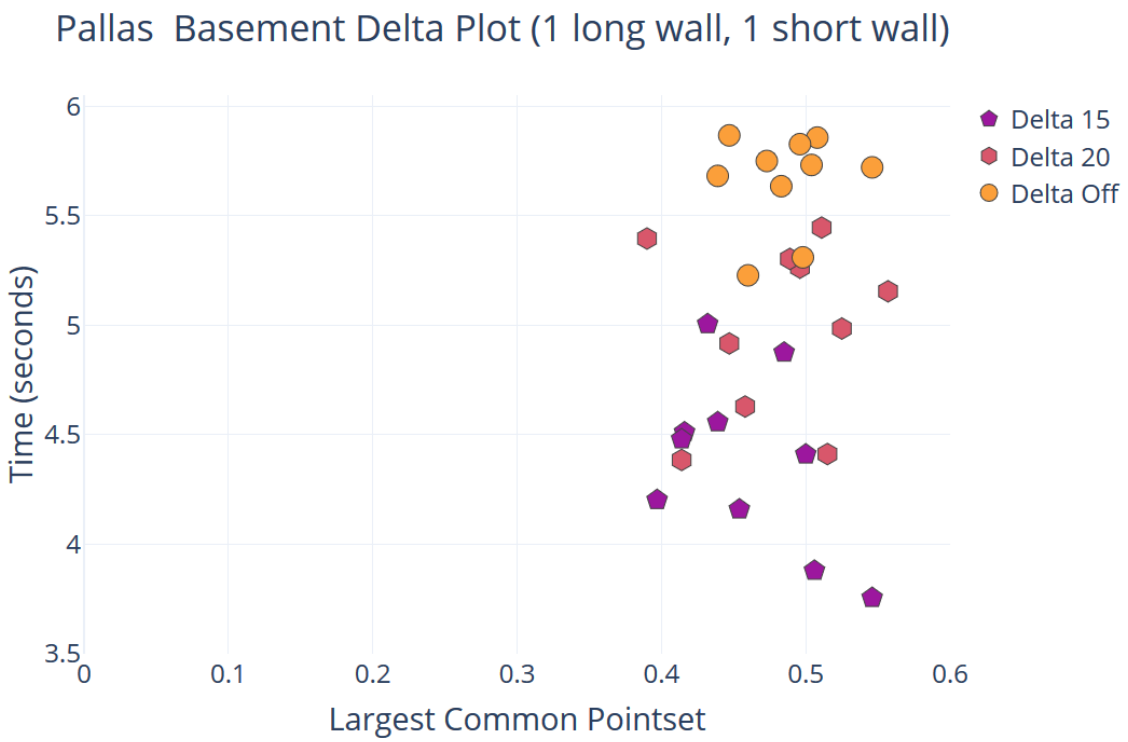

Figure 26. 1 long, 1 short wall LCP results.

Table 4. 1 long, 1 short wall match quality.

Delta (10 runs)	Average Time (sec)	Average LCP	Bad Matches	Close Matches	Great Matches	Perfect Matches
Delta 15	4.384	0.459	0	4	3	3
Delta 20	4.989	0.48	0	0	4	6
Delta Off	5.66	0.485	0	4	4	2

Out of all the test cases, this one (Figure 26, Table 4) gave the best results, as none of the provided delta values got a bad result during initial testing. At the same time, due to the overall decreased size, it was simultaneously much more difficult to determine the quality of the match itself. The main issue for close matches was usually the rotation. Apart from that, most of the matches followed the expected situation, where a higher LCP value means a better match.

3.3.4. Improving Results

Taking all the previous results into account, the next step was to add potential solutions to reduce the frequency for any errors that came out during the process.

The first thing done was adding an additional limiter for the point cloud scanner to only remember points closer than two meters from the device at the time of the point being detected. The reasoning behind this is that the further away a user is from a wall or area of interest, the more likely that the depth calculation for the feature point will fail and end up creating an outlier instead. Or, the scanned area becomes far noisier as a result of not being able to accurately gauge the distance. By having a set limit on how far a point can be, the user will still be moving around the room, pointing their camera at points of interest without any badly detected points being saved.

The second idea was to alter the approach on what areas should be scanned for the source point cloud. Noise can have a large impact on the bad results for point clouds, but there is no confirmation on the point clouds themselves so far. The floor was near-fully scanned for each test, and considering that it covers a large amount of area, this can theoretically cause the sampling process to pick out points from it in an unfavorable way. For example, the sampling could favor one side of the floor and group points there, causing the translation to lean towards that direction. This can also explain why 180-degree rotations were picked out on certain scans far more often – the algorithm found more “good" matches with these sampled results. Looking

for alternate ways to compare point clouds, such as only scanning the walls for the source point cloud, could also improve results.

3.4. Experiment 3 – Gallery Pallas Second Run

The first thing done after the limiter was implemented was going through the Gallery Pallas experiment again. This is to make sure if results actually improved when it was applied. The only test case not redone was the one where one of the longer walls was not scanned in, mainly because it did not significantly improve results compared to a full scan. For the rest of the test cases, the results can be seen below:

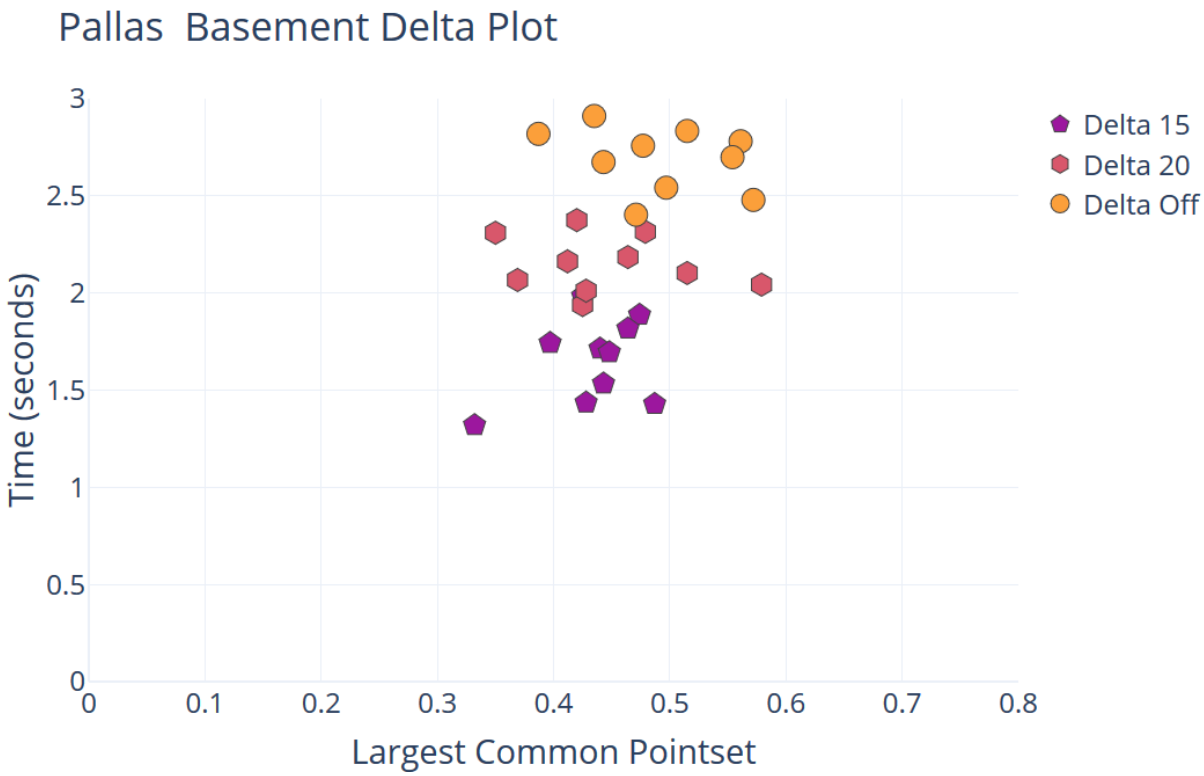

Figure 27. Updated full scan LCP results.

Table 5. Updated full scan match quality.

Delta (10 runs)	Average Time (sec)	Average LCP	Bad Matches	Close Matches	Great Matches	Perfect Matches
Delta 15	1.656	0.4338	3	3	4	0
Delta 20	2.151	0.4441	4	0	3	3
Delta Off	2.690	0.4912	2	1	3	4

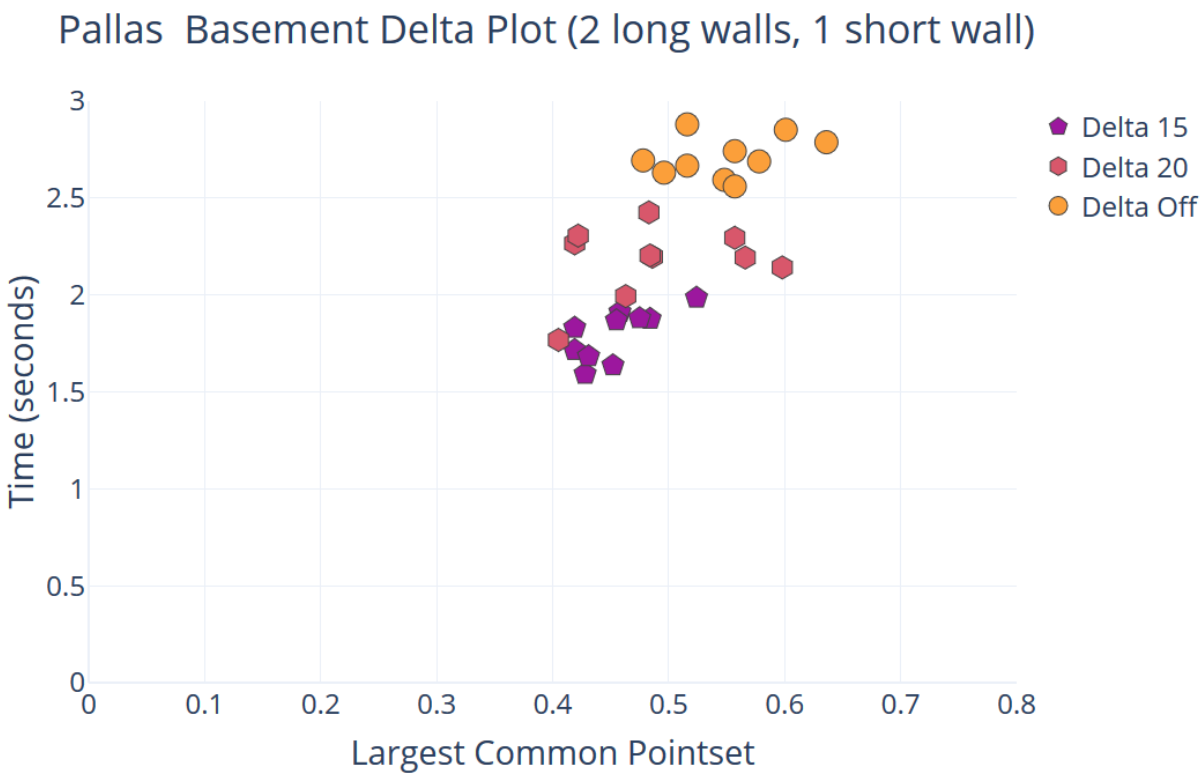

Figure 28. Updated 2 long, 1 short wall LCP results.

Table 6. Updated 2 long, 1 short wall match quality.

Delta (10 runs)	**Average Time (sec)**	**Average LCP**	**Bad Matches**	**Close Matches**	**Great Matches**	**Perfect Matches**
Delta 15	1.798	0.4545	5	2	3	0
Delta 20	2.179	0.4883	4	3	2	1
Delta Off	2.709	0.5483	1	0	6	3

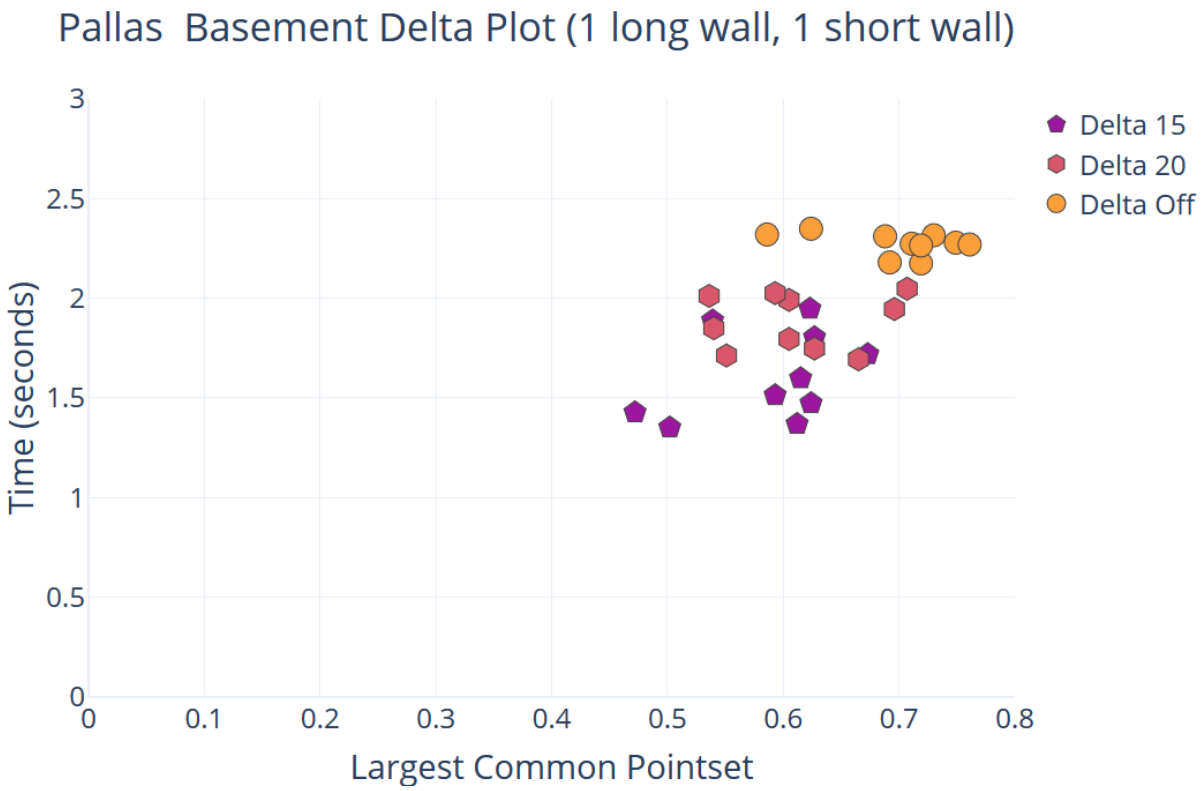

Figure 29. Updated 1 long, 1 short wall LCP results.

Table 7. Updated 1 long, 1 short wall match quality.

Delta (10 runs)	**Average Time (sec)**	**Average LCP**	**Bad Matches**	**Close Matches**	**Great Matches**	**Perfect Matches**
Delta 15	1.609	0.588	3	2	5	0
Delta 20	1.882	0.6125	0	5	3	2
Delta Off	2.274	0.6979	0	2	2	6

The first thing of significant note is the performance. With the extended tests, the average time was around 4-8 seconds across all test cases, and in the case of the full scan, over 13 seconds on average. However now, none of the new tests (Figure 27, Figure 28, Figure 29) went over 3 seconds. While the number of outliers is clearly reduced due to the new optimization, the total amount of points for some of these cases did not differ that much from their original counterparts. It is far more likely that the point cloud sampling and octree performance were improved through this optimization, as now there are no longer any points floating a few meters away from the core point cloud itself. This means less points being taken out during the sampling process and octrees having to work on a far smaller area.

The second thing is the LCP values. Again, on average, these values are much higher, since they are now in more concentrated areas, making it far more likely to pick out a good match during the LCP match process.

For the actual match quality (Table 5, Table 6, Table 7), with delta at either 15 or 20 degrees, the results were mostly similar or slightly worse than before. At the same time though, most of these tests can still return great to perfect matches. The amount of iterations can be increased to improve the likelihood of finding a better match, while still being faster than the original tests. When delta was off, the results appeared to be better than before. Since there are far less outliers, the likelihood of picking a good match out of a pool of potential pairs would end up being further improved, while also taking less time than before. One other thing to note is that during the test with only 2 walls, the perfect match cases were the most consistently good cases compared to all other tests done so far. All of them were scoring over 0.7 with the LCP algorithm, the highest value so far with real world test cases.

When it comes to bad matches, the same issues were present. Either the rotation or translation were really off, or the rotation still turned itself around 180 degrees. A better way to measure accuracy compared to LCP might be necessary to fix these cases from happening.

Test 4 – Only Scanning the Pictures

The final suggestion was to only scan the pictures hanging around the room while ignoring the rest of the room. The idea is that if there is no floor to pick points from, it is far likelier to pick out good pairs between just the pictures. All points within the sampled point cloud would only be picked out from images. The results are shown on Figure 30 and Table 8:

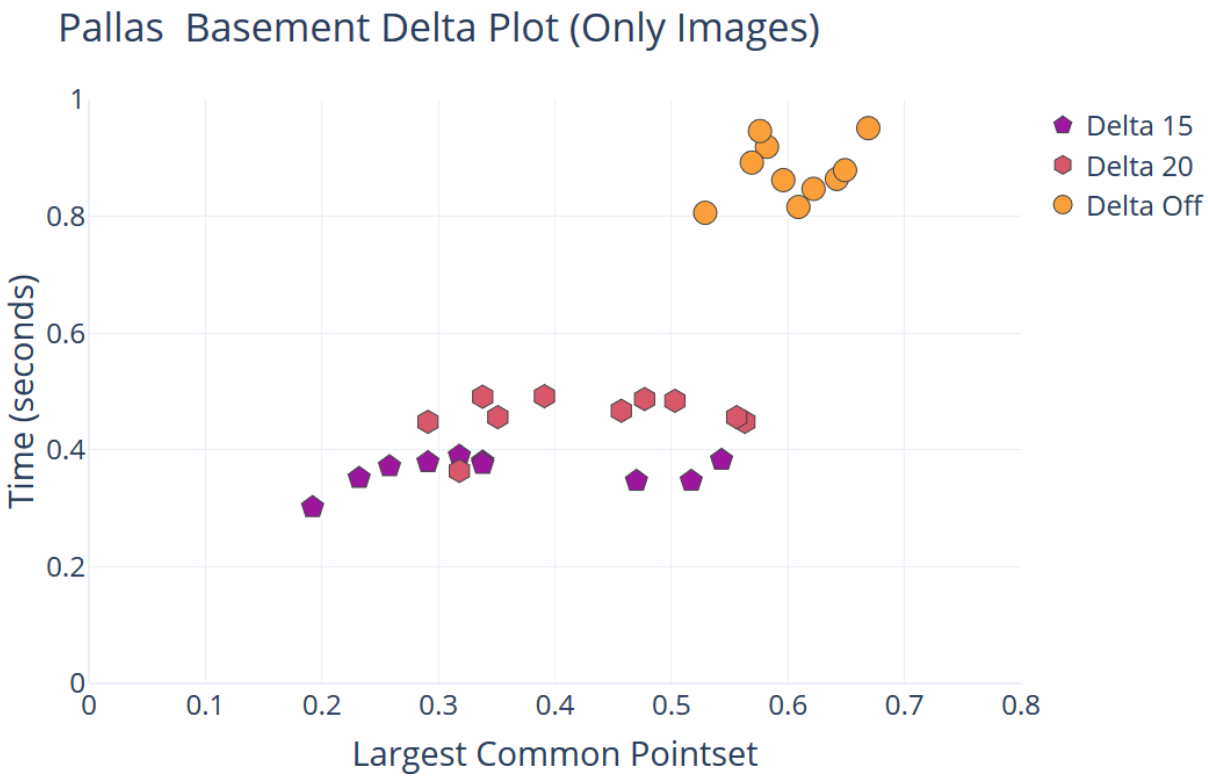

Figure 30. Only images LCP results.

Table 8. Only images match quality.

Delta (10 runs)	**Average Time (sec)**	**Average LCP**	**Bad Matches**	**Close Matches**	**Great Matches**	**Perfect Matches**
Delta 15	0.362	0.349	4	1	5	0
Delta 20	0.459	0.425	0	4	5	1
Delta Off	0.878	0.604	0	0	5	5

Based on these LCP and match quality results, 15- and 20-degree deltas are still somewhat questionable on their viability. They can give good matches, but also still show signs of bad rotations or even a full 180-degree rotation in a few cases, like all the other previous tests. A consistent issue was the translation hovering more upwards than normal, likely due to there being no floor to ground matches against.

When delta was disabled though, results became a lot more consistent. While they still shared similar issues, the quality of the match itself was always pretty good. The range at which the rotation was determined did not deviate more than 6-7 degrees from the ideal, and usually any rotation issues were more emphasized due to a bad pivot being selected. Furthermore, with the

average time being less than a second, it would be possible to increase the amount of iterations and get even better, consistent results.

3.5. Experiment 4 – Mobile Tests

With the results verified, the best parameter sets from experiments 2 and 3 were used to experiment on a mobile device (see **Appendix II – Mobile Device Hardware**). The same tests as the previous subchapter were done, matching:

1. Two different point clouds together using full scans,
2. Scans with one short wall not included,
3. Scans with both one long and one short wall not included,
4. Scans with only pictures on the wall.

The difference here is that for each test case, the tests are no longer done with just 2 different point clouds. For each test case the source point cloud is shared, but the target point cloud is rescanned differently each run. This gives a more realistic estimate on the algorithm's performance when a regular user would be using it. Each new target point cloud was started at a slightly different location and scanned slightly differently (some parts scanned more lazily than others).

Figure 31. 3D vase.

To verify the accuracy of matches on mobile, a simple 3D vase model (Figure 31) was created. All source point clouds origin points are at the middle of the basement room of Gallery Pallas. The vase would then be automatically set at the origin point (Figure 32 shows the before and after when the algorithm is run).

Due to the randomness with each test case being different, the results of such experiments are much more unpredictable. In this experiment, the results were assessed by looking at the placement of the vase instead of rotation and LCP. This is because each test now has a different best-case rotation and maximum LCP. All of them can be viewed in the accompanying PDF file (MobileResults.pdf).

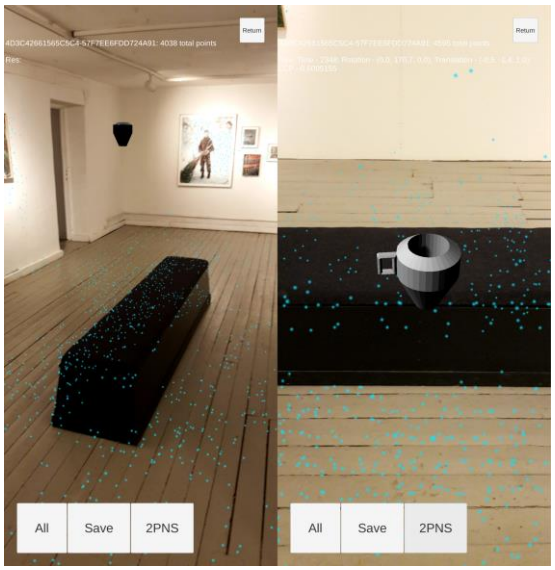

Figure 32. Positioning the virtual vase.

In summary, while some of these tests gave results that would be acceptable in the given error range, most tests showed noticeable issues:

- In the **full point cloud scan** tests, for all delta values, there were a few good cases, but most of them had noticeable translation issues, where the vase model started to move too far away from the center of the room. The rotation was also inverted in a few cases.
- In the tests where **one short wall was not scanned** in, the 15- and 20-degree delta value tests had some of the worst translation matches across all tests. When delta was disabled, results were far more consistent, but also had a tendency to hover slightly towards one specific direction.
- In the tests where **only one long and one short wall were scanned** in, the 15- and 20-degree delta value tests still had inconsistently bad matches, making it hard to determine the cause. With delta disabled, it had some of the best matches across all matches. At the same time, around half of the matches had the translation sway slightly towards one side, or upwards.
- In the tests where **only the pictures on the walls were scanned**, all delta cases had issues in all sorts of ways, whether the rotation had flipped around, or the translation had swayed off a lot in one direction. In some cases, the vase even went into the floor.

With all the improvements, the best results came with delta being completely disabled, which also struggled with translation or rotation at times. Even then, there is always room for further improvement. For one thing, the actual time taken for the matching process to finish is around the same as it being done on a computer. This means that increasing the iteration count could increase match quality without sacrificing too much performance on a mobile platform. The current result of the algorithm might not work well on a mobile device right now, but can be improved on in the future.

In summary, the 2PNS algorithm had issues with runtime and matching quality. Both of these were significantly improved on by changing the algorithm to only rotate across the y-axis, adding a pair limiter to use up to 100 pairs per iteration and sampling point clouds to reduce the amount of points checked. Afterwards, changing the point cloud scanner to only accept points less than 2 meters helped reduce point cloud noise, which further reduced runtime to below 3 seconds and slightly improved matching quality. Each one of these additions increased the amount of good quality matches across all experiments, but when it came to mobile tests, additional work would be needed for more stable results across random point clouds.

4. Conclusion

This thesis set out to find and experiment the possibility of using point cloud matching algorithms for AR purposes. Different types of potential issues with the matching process were listed. Finding a suitable algorithm for the given point cloud matching problem went through multiple iterations.

4PCS is one of the earlier examples for aligning point clouds that have noisy data, but its implementation is rather complex and its performance is not that great compared to future advancements. S4PCS and G4PCS, made several improvements to reduce time complexity. SG4PCS ended up combining the two together for even better performance. All of these algorithms ended up being rather difficult to implement.

2PNS ended up being chosen for point cloud matching. Compared to S4PCS, it simplified certain aspects on matching points, which helped it perform better and made it much easier to implement. For this thesis, additional features were also added to it to improve matching quality and reduce runtime: a pair limiter, sampling point clouds and only rotating point clouds across the y-axis.

The first experiment was done with a premade rectangular point cloud box. It was mainly used to justify the need for the implemented additional features. The pair limiter and point cloud sampler helped reduce runtime. The y-axis rotation helped fix an issue where the first point cloud always rotated upside down when matching against the other point cloud.

All of the following experiments were done in the Gallery Pallas basement exhibition. For the second experiment, the main goals were to find what rotation error delta gave the best results when matching and what scanning conditions gave the best results overall. For the first half, having the error at 15 degrees, 20 degrees or completely disabled had the best results, which is why these values were chosen for all subsequent tests. For scanning conditions, most tests had issues that caused several bad matches and large amount of variance in good matches. The reason was either the translation or rotation being significantly off or a combination of both of them. In a few cases, the point cloud was also rotated 180 degrees, implying issues with symmetry. The best results were when only 2 walls next to each other were scanned with the floor. A full scan of the entire room gave the worst results.

For the third experiment, an additional distance limiter was added to the point cloud scanner, where it would not add any points scanned further than 2 meters. An additional test case was also suggested, where only the pictures on the wall were scanned, but not the floor. Matching quality for all tests redone from the second experiment were mostly the same except when delta

error was off, in which case it was very slightly better. Matching runtime was reduced by even larger amounts. None of the tests took more than 3 seconds. The point cloud with only pictures scanned gave the best results of all cases with delta off. Half of the results were near-perfectly matched and the other half had only minor issues. When looking at other test cases though, the possibility of finding a better replacement for LCP was brought up, as the bad results in some test cases had LCP values comparable to good results.

The fourth experiment was redoing the tests from the third experiment on a mobile device, as the main goal was to find out the viability of a point cloud matching system on smartphones. The results were not great. Delta being disabled was the only instance with more acceptable results (around half), although it still had plenty of troubles. Scans with only 2 walls had the most stable results, while the scan with only images was considerably worse off than before.

For Mobi Lab, it would still be possible to design a mobile application with the current algorithm, but would require additional work. 2PNS is easier to understand and implement compared to 4PCS algorithms and also easier to modify, as it uses 2-point pairs.

I would like to thank Mobi Lab for offering this thesis idea to work on, as well as Gallery Pallas for a place to test the created algorithm. Understanding point cloud detection in-depth and implementing a point cloud matching system was a challenging task, not to mention getting it to work on a mobile device. Given that my previous thesis was also about AR, I found it valuable to understand how it works even more now.

Appendix I – Glossary

Random Sample Consensus (RANSAC) - An algorithm that is widely used for robust fitting of models to data that has noise and outliers present. The general procedure is to first pick 3 random points from point clouds P and Q and form a pair of bases in correspondence. Then, compute the candidate transformation T_i that aligns the pairs and count the number of points k_i from P that are within δ-distance from points in Q. If k_i is large enough, T_i is accepted as a good answer, otherwise the entire process is repeated to get a better result. This is repeated a set amount of times and stopped early if a sufficiently good enough solution has been gotten.

Octree - A tree data structure in which each internal node has exactly eight children. Used in 3D spaces to subdivide areas into eight octants. When there is a point within one subspace, that one is further divided into eight octants, up to a minimum size. The intent in this thesis is to have a method to efficiently query points in a point cloud in relation to another point. When searching for neighbors within a range to the source point, instead of having to compare against every point, it only has to check the points that are in nodes that are inside the range or intersect with it.

K-d Tree - A binary tree where every leaf node is a k-dimensional point. Every non-leaf node can be interpreted as a hyperplane that splits the current subspace into two equal halves. Depending on the dataset, there are different methods to construct a k-d tree, but the general process is to loop and add a hyperplane through each axis until reaching a minimum size. The splits are decided based on the location of points.

Singular Value Decomposition (SVD) - A factorization of an $m \times n$ complex matrix M to the form $U\Sigma V*$, where U is an $m \times m$ complex unitary matrix, Σ a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V a $n \times n$ complex unitary matrix. If the matrix M is real, U and $V^T = V*$ are real orthogonal matrices.

It is essentially a method to decompose a transformation matrix into 2 separate rotations (U, $V*$) and scaling (Σ) that can be used for various purposes. In the context of this thesis, to estimate normals for all points in a point cloud.

Plane Principal Component Analysis (PlanePCA) - An algorithm used to assign normals for every point in a point cloud. The problem can be defined as a set of n points represented as a data matrix $\mathrm{P} = [\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \dots, \mathrm{p}_\mathrm{n}]^\mathrm{T}$, where each point is a 3D vector position $\mathrm{p_i} = [\mathrm{p_{ix}}, \mathrm{P_{iy}}, \mathrm{p_{iz}}]^\mathrm{T}$. For each point, a normal vector $\mathrm{n_i} = [\mathrm{n_{ix}}, \mathrm{n_{iy}}, \mathrm{n_{iz}}]^\mathrm{T}$ is estimated from a set of k points in its neighborhood $\mathrm{Q_i} = [\mathrm{q_{i1}}, \dots, \mathrm{q_{ik}}]^\mathrm{T}$. An additional $\mathrm{Q_i^+} = [\mathrm{p_i}, \mathrm{q_{i1}}, \dots, \mathrm{q_{ik}}]^\mathrm{T}$

is defined, with the difference being the very first point being the chosen point p_i in addition to its neighbors.

For finding neighbors k nearest neighbors (kNN) is used, as an octree is already implemented and can easily be expanded for picking out the closest points to the chosen point as well.

PlanePCA, in summary, minimizes variance, removing the empirical mean from the data matrix Q_i^+. An SVD is then performed on the modified data matrix to extract Σ and V. The smallest diagonal value in Σ is used as the index for V, which would be the estimated normal for the given point and its neighbors. The full calculation is $\min_{n_i} \left|\left|[Q_i^+ - \overline{Q_i^+}]n_i\right|\right|_2$.

Rigid Registration (Transformation) - The process of matching two separate point clouds without the distance between two points of a point cloud changing.

Vector Rejection - While a vector projection is the orthogonal projection of vector **a** on a *straight line parallel to vector* ***b***, the rejection is the orthogonal projection of vector **a** onto the *plane orthogonal to vector* ***b***.

Appendix II – Mobile Device Hardware

Name: Samsung Galaxy Note 8

Display resolution:	1080 x 2220 pixels
Camera resolution:	12MP
Camera pixel size:	1,4µm
Camera sensor size:	1/2.55’’
Camera sensor ratio:	4:3
Camera FOV:	77°
Camera aperture:	F1.7

Processor: Samsung Exynos Octa 8895
2,31 GHz (8 cores)

Figure 33. Mobile device used for experiments.

Appendix III – License

Non-exclusive licence to reproduce thesis and make thesis public

I,
Karl - Walter Sillaots,
(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Mobile AR Point Cloud Matching,
(*title of thesis*)

supervised by Raimond-Hendrik Tunnel, Timo Kallaste.
(*supervisor's name*)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Karl - Walter Sillaots
12/05/2021

References

[1] How Augmented Reality Can Improve Employee Performance (Lana Gates, 2018) https://www.insight.com/en_US/content-and-resources/2018/08062018-how-augmented-reality-can-improve-employee-performance.html

[2] LIDAR vs. Camera — Which Is the Best for Self-Driving Cars? (Vince Tabora, 2020) https://medium.com/0xmachina/lidar-vs-camera-which-is-the-best-for-self-driving-cars-9335b684f8d

[3] Why Apple's LiDAR Scanner Opens Up a Brave New World of Indoor Mapping (Zuzanna Villa, 2020) https://blog.magicplan.app/why-apples-lidar-scanner-opens-up-a-brave-new-world-of-indoor-mapping

[4] Super4PCS: Fast Global Pointcloud Registration via Smart Indexing (Nicolas Mellado, Niloy Mitra, Dror Aiger, 2014)

[5] 4-Points Congruent Sets for Robust Pairwise Surface Registration (Dror Aiger, Niloy J. Mitra, Daniel Cohen-Or, 2008)

[6] Generalized 4-Points Congruent Sets for 3D Registration (Mustafa Mohamad, David Rappaport, Michael Greenspan, 2014)

[7] Super Generalized 4PCS for 3D Registration (Mustafa Mohamad, Mirza Tahir Ahmed, David Rappaport, Michael Greenspan, 2015)

[8] Using 2 Point+Normal Sets for Fast Registration of Point Clouds with Small Overlap (Carolina Raposo, João P. Barreto, 2017)

[9] Comparison of Surface Normal Estimation Methods for Range Sensing Applications (Klaas Klasing, Daniel Althoff, Dirk Wollherr, Martin Buss, 2009)