UPPSALA
UNIVERSITET

UPTEC IT 20008
Examensarbete 30 hp
April 2020

Simple feature detection in indoor geometry scanned with the Microsoft Hololens

Nils Björk

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Simple feature detection in indoor geometry scanned with the Microsoft Hololens

Nils Björk

The aim of this work was to determine whether line-type features (straight lines found in geometry considered interesting by a user) could be identified in spatial map data of indoor environments produced by the Microsoft Hololens augmented reality headset. Five different data sets were used in this work on which the feature detection was performed, these data sets were provided as sample data representing the spatial map of five different rooms scanned using the Hololens headset which are available as part of the Hololens emulator. Related work on feature detection in point clouds and 3D meshes were investigated to try and find a suitable method to achieve line-type feature detection. The chosen detection method used LSQ-plane fitting and relevant cutoff variables to achieve this, which was inspired by related work on the subject of feature identification and mesh simplification. The method was evaluated using user-placed validation features and the distance between them and the detected features, defined using the midpoint diistance metric was used as a measure of quality for the detected measures. The resulting features were not accurate enough to reliably or consistently match the validation features inserted in the data and further improvements to the detection method would be necessary to achieve this. A local feature-edge detection using the SOD & ESOD operators was considered and tested but was found to not be suitable for the spatial data provided by the Hololens emulator. The results shows that finding these features using the provided data is possible, and the methods to produce them numerous. The choice of mehtod is however dependent on the ultimate application of these features, taking into account requirements for accuracy and performance.

Handledare: Stefan Seipel
Ämnesgranskare: Filip Malmberg
Examinator: Lars-Åke Nordén
UPTEC IT 20008
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

Augmented Reality(AR) is the concept of mixing physical elements such as real world geometry, weather, light etc. with virtual elements like 3D holograms, information overlays or other visual information thus 'augmenting' the real world around the user. The concept is similar to Virtual Reality(VR), the difference being that in VR all elements are virtual, usually with the aim to immerse the user in a completely virtual world without influences from the physical one. Using the Microsoft Hololens headset, 3D vertex and face data approximating the users surroundings can be obtained as a 'spatial map'. A spatial map can be used by applications that require information about the local space of an AR user. A couple of examples include occluding 2D holograms to emulate the effect of them existing behind real geometry and virutal physics interactions like bouncing a virtual ball on the physical floor and walls surrounding a user. The aim of this work was to ascertain whether this 3D data is suitable for simple feature detection in indoor geometry without guidance or input from the user. In the context of this work, straight lines present in the spatial map are considered as simple features. This work details an attempt at designing a detection algorithm inspired by related work on feature detection, mesh simplification and mesh saliency.

1.1 AR applications

Augmented reality has seen a resurgence in recent years with the development of technologies like the Microsoft Hololens, but is by no means a new concept. A large amount of work exists regarding AR applications. ARQuake [29] is an example of an earlier application of AR technology which details an attempt to simulate the first person shooter game Quake in real world environments. Although successful the technology was not mature enough to allow for a fluent AR experiecne.

Another early example of applying AR technology is the work by Kaufmann & Schmalstieg [20] who combined head-mounted devices and desktop computers to allow students and teachers to visualize vectors and geometric shapes in order to facilitate understanding of these concepts. Although they found promise in the application in the form of anecdotal evidence, they stated the need for further validation.

More recent work by A.Y.C. Nee, S.K. Ong, G. Chryssolouris and D. Mourtzis [25] examined how AR could be applied in a great number of situations with an emphasis on applications within manufacturing. Some examples of these applications included augmented modeling of the interior of a car on the assembly line, modifying and combining models extracted from real world objects with virtual elements and models as well as collaborative AR systems where multiple users might visualize and interact with virtual models.

Mekni & Lemieux [24] published a study on the applications, challenges and future trends of AR. The study covered a wide array of areas that AR had been applied to a few examples being urban planning, construction, entertainment,

military and medicine. They found AR, even though useful a bit too immature a technology which would need considerable improvement on the hardware level to prove useful for most applications. Some of the concerns were bulkiness, battery life, position accuracy and the lack of a recognition system.

Similarly Carmigniani, Furht, Anisetti, Ceravolo, Damiani & Ivkovic [5] surveyed how AR technologies and concepts could be applicable in both an academic and commercial settings. They also discussed what requirements AR applications would have in order to be useful in these settings, examples include social requirements, ways of interacting with an AR device and display methods. In their conclusion, AR was found to be in its infancy and the at the time current smartphone applications were "not used well" when it comes to AR, focusing mainly on entertainment and games. Future areas where AR could be useful included: Helping deaf individuals by visualizing audio cues in their environment, virtual dressing rooms and AR contact lenses making the displayed information more personal. Privacy concerns were also raised, when it comes to medical applications, sensitive information about a patient would have to be considered to sustain confidentiality. A point on the ethical aspects of AR warned that it is meant to 'augment' and improve a users experience of the physical world, not interfere with it.

Barsom, Graafland & Schijven [4] reviewed the efficiency of several AR applications for medical training. Three categories of applications were reviewed: Laparoscopic surgical training, mixed reality training of neurosurgical procedures and training echocardiography. They concluded that the interest for AR has increased in the recent years in both academia and the public view, even though the at the time current literature on the subject lacked in evidence proving them useful in medical training specifically.

The work by Wojciechowski, Walczak, White & Cellary [31] details a relatively early example of using physical AR markers and a head-mounted display to build virtual museum exhibits both in a physical museum space as well as online. They found the technology useful to enrich the museum visiting experience with more possibilities for learning and interacting with the virtual exhibits.

A study by Antonioli, Blake & Sparks [2] examined the usability of AR in an educational environment. They concluded that AR can be a very useful tool to help both students and educators alike perform better by increasing the focus of students undertaking a certain task or giving educators access to more elaborate examples like a realistic cross-section view of the digestive system. They also raise potential problems with the current AR applications such as tunnel vision and technical problems.

Another study by Chi, Kang & Wang [7] discusses current(2013) and future trends in different areas of AR applications, specifically within the areas of architecture, engineering, construction and facility management. Four technologies applicable to AR applications were discussed: localization, natural user interface, cloud computing environment, and portable and mobile devices. They found these technologies and their potential applications promising which they predicted could increase safety, productivity and efficiency in these areas.

1.2 Microsoft Hololens

Figure 1: The Microsoft Hololens headset

The Microsoft Hololens is a type of AR headset known as a 'Head Mounted Display' featuring a clear display on which 3D and 2D holograms can be displayed (Figure 1). Using a set of sensors the Hololens continuously maps the surroundings of the user which produces a corresponding 3D spatial map. With information about the users position and viewing angle relative to the spatial map, the Hololens is capable of displaying both occluded and non-occluded holograms, creating the illusion of their existence as a part of the physical world surrounding the user [3] [9]. It is also equipped with network capabilites allowing for access to central servers for data streaming purposes as an example. This also allows for multiple Hololens units to be interconnected allowing for collaborative setups where by using spatial anchors, multiple users can interact and percieve the same holograms. The spatial mapping makes the occlusion of holograms possible, but it also allows for features such as collision detection between virtual objects and real geometry, placing virtual items on top of a physical table or bouncing a virtual ball off of a physical wall is made possible because of this feature. The Hololens also features headphones as well as voice and gesture recognition support which alleviates the problem of allowing for user input without a dedicated peripheral.

Quite a few applications have been implemented on the Hololens. This project focuses on utilizing the spatial mapping features of the Hololens which could be considered one of its main features, however many of the works described in the following section also utilize its voice and gesture detection features as well.

1.3 Hololens applications

An example of a medical application is the work by Hoffman & Provance [16] used the Hololens headset to render a 3D model of aspirin and HLA molecules and their structures. They found that the AR rendering of these molecules illustrated the three-dimensional aspects of these molecules better than a traditional 2D image, which in turn led to a better understanding of their structure.

In the field of reconstructive surgery Pratt, Ives, Lawton, Simmons, Radev, Spyropoulou & Amiras [27] used the Hololens for overlaying 3D models aquired from a CT-scan of a patient detailing an area in need of surgery. The aim was to find out if the technology was suitable for this type of surgery. They found that compared to previous methods of locating veins and arteries before an incision the Hololens proved more reliable, reducing anaesthetic time and making remote support available for the operating doctor. A drawback however was that a technician was required to be present to assist with the technical

aspects of setting up and using the headset itself.

Another study by Hanna, Ahmed, Nine, Prajapati & Pantanowitz examined the usability of the Hololens headset in Anatomic Pathology use cases [14]. Using the internet capabilities of the Hololens personnel in training could be instructed by an expert remotely when performing an autopsy of a specimen. They also allowed for users to view 3D models of organs and tissue samples as they were examining a physical specimen.

Zhang, Chen, Dong & El Saddik visualized Toronto city using the Hololens headset as a 3D map augmented onto a flat surface with user interaction [32]. The implementation allowed users to view the city layout in an immersive way as well as interacting with points of interest and receiving information about these as 2D informative holograms.

As an example of collaboration using the Hololens, Lee, Swift, Tang & Chen [6] proposed an application where multiple collaborators could take part of and annotate upon a live feed of the Hololens users view as well as the spatial data associated with it.

The Work by Hockett & Ingleby [15] utilized the spatial map to allow the placing of 3D models of architecture in the real world in order to review suitability of housing placements. Similarly Ahn, Ko & Gim [1] presented a collaborative augmented reality system where users could place and interact with holograms synchronized using a central server housing models and keeping the different AR views synched up and consistent. They also proposed using a spectator setup using a camera in conjunction with a Hololens headset to allow for a view of the holograms outside of the headset users.

The above literature using the Hololens headset presents its suitability for feature detection in a promising light. Considering the complexity of some of the projects, identifying line-type features seems quite possible if not simple in comparison. There are however a few problems preventing a naive approach to this problem, the following section describes the most prominent one, the quality of the spatial map itself.

1.4 Spatial map data set

The Hololens headset provides a spatial map of the users surroundings. The data provided consists of triangle mesh patches with accompanying vertex positions, triangle indices and vertex normals. Note that due to the lack of an actual headset to generate these spatial maps, all data utilized in this project is based on the provided sample data sets provided along with the Hololens emulator.

The provided mesh is quite rough and segmented into patches being 'scanned in' in real time. The patches are not connected by any shared vertices and slightly overlap each other to hide this fact. This means that for this project as it would be necessary to examine features across multiple patches, the feature identification must be done on the collection of patches to be able to find these(Figure 2). The mesh also contains non-manifold edges which would need to be removed if required by an application, this is however not necessary for the chosen implementation in this work. Table 1 shows the statistics of the data

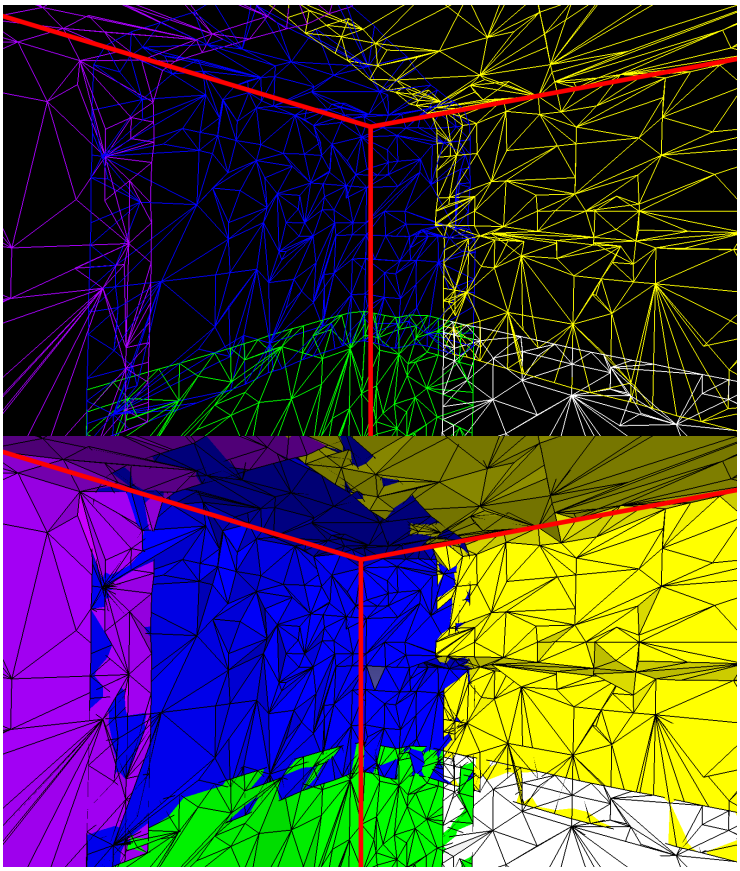

Figure 2: Example of the spatial map patches illustrating the overlap and lack of connectivity between them as well as an observed feature across multiple patches (red), each other colour corresponds to a separate patch.

sets used in this work.

2 Project goals

This work aims to ascertain whether the Hololens produced data sets in the previous section are suitable for reliably detecting salient structural features present in indoor geometry and which methods would be suitable to achieve this. Salient structural features present in the geometry surrounding a user can be useful in a number of scenarios. A couple of examples include:

Quickly vizualizing changes in the geometry. By moving, subdividing or joining features, a change to the geometry (such as a renovation of a room of

	Bounds(m)	No. Vertices	No. Faces	Avg. Face area(m^2)	Vertex density (v/m^3)
D_1	7.80 x 3.26 x 6.58	30654	51754	0.0033	183.16
D_2	5.05 x 3.83 x 8.43	42706	75887	0.0022	280.37
D_3	5.92 x 2.82 x 7.66	42134	74052	0.0021	350.02
D_4	8.35 x 3.34 x 9.03	62556	107689	0.0021	271.32
D_5	9.05 x 3.23 x 7.27	45423	76110	0.002	235.2530

Table 1: Data set statistics.

a house for instance) can be immediately visualized and interactively modified by the user without any considerable delay. Coupled with an AR device and graphics rendering, the potential for almost completely realizing a completed potential redesign in the virtual space is possible before any actual step has been taken towards it.

Validating a building during and after construction. With an AR device and a static anchor a theoretical model or blueprint of a building can be placed on top of the area where it is supposed to be or is being built. Such an application could allow for continuous validation of all aspects of the building as it is being built, reducing the risk of misinterpretation of the blueprint and thus incorrect construction.

In order to restrict the scope of this work, only line-type features are considered. As for the definition of 'interesting', this could be quite a subjective definition but in this work the line-type features present along creases and edges of the mesh are considered as such. The natural starting point for finding suitable methods is related literature on the subject. Another consideration for this work is validation. Having defined what constitutes an 'interesting' feature, a mathematical measure of quality of any produced features is required.

2.1 Feature detection methods

Since the data sets are represented by 3D meshes, approaches in related works that produced features with the same kind of data is a good point to start when deciding on the design of the feature identification algorithm. Naturally in a 3D mesh, information about vertex positions is available which also makes point cloud methods available for use, hence literature on feature detection in point cloud data is also relevant to this work. Another area that is dependent on detecting features or important areas of a mesh is mesh simplification, some of the works in this are are also considered. Section 3.3 recounts a number of relevant examples of works within these areas. Note that a lot of the related works deal with identifying line-type features as well as more complex features such as splines, curves and circles.

A seemingly promising approach is using planar feature identification to identify naturally flat surfaces in the data. The related works [28] [10] showed how finding these planes could be useful for finding different kinds of features in both point clouds and vertex meshes. An attempt will be made in this work

to use this method to achieve feature detection.

2.2 Feature validation

Due to the subjective nature of an 'interesting' feature in 3D space, a measure of quality has to be decided upon in order to rate the 'correctness' of any observed or identified features. To solve this problem hand-made validation features can be used. By placing features along edges and creases or any other area of a mesh where a 'line' is intuitively interpreted, a measure of similarity between these validation features and the features produced by the algorithm can now be decided on, which in turn reflects the quality of the produced features as the validation features should represent the 'correct' result.

3 Theoretical background

A vast amount of work exists on feature detection in 3D meshes as well as many works that rely on some sort of feature identification or weighting of areas of a mesh. Mesh simplification is very relevant to the problem of finding features in a mesh as it requires some sort of weight or value attached to the points or edges in the mesh in order to determine if they can be removed when reducing the mesh.

3.1 Saliency and features in 3D meshes

A salient feature in the context of 3D mesh data is defined as a pattern of points or edges that are considered 'interesting' [23]. Two examples relevant to this work: A corner where three relatively flat surfaces meet or a theoretical line that follows the intersection of two similar surfaces. More complex salient features include outstanding curvatures, pits & bumps for example. Several of the following related works deal with the task of finding saliency in arbitrary meshes and point cloud data.

3.2 Feature detection in 3D meshes

With a concept of 'salient' features defined as: 'Straight lines along edges and creases of the mesh' in place, which will be referred to as "line-type features" in the rest of this work, the next part of solving the feature extraction problem is to apply methods and algorithms suitable to identify these types of features in a 3D mesh which in this case is a Hololens-generated spatial map. The problem of identifying interesting or salient features in a 3D mesh can now be reduced to 'finding line-type features in a 3D mesh'. The next question is which method to apply to solve this problem? The following section details some interesting works dealing with feature detection directly or applications of it like in mesh decimation for example.

3.3 Works related to feature detection in 3D meshes

The work by Hussain, Okada & Niijima [19] approaches mesh simplification by determining a 'visual significance' value to the vertices used in a cost function in order to decide which vertices to remove using half edge collapse. These 'visually important' vertices relate to the more abstract features sought after in this work, with a more low level approach however as the global relationship of the mesh's vertices and edges was not examined in their application.

In the same spirit, Franc & Skala [11] applied the edge collapse method for simplification together with re-triangulation of a new vertex in order to preserve the model appearance. Their approach is also driven by a calculated vertex 'importance' value, which drives the simplification algorithm.

Work on feature extraction from point cloud data is also closely related to this work as the only input available is the local data points with some potential metadata attached to them. Any potential global features of the point cloud are not known at input.

The work by Gumhold, Wang & MacLeod [13] which presents a mehtod of detecting feature lines in triangulated point clouds using a point classification method. The classification used correlation ellipsoids of a points neighbours, and by weighting these points decided which might be part of a feature. The most prominent features are extracted and the creases and corners of the mesh recovered using a number of recovery methods, including least-squares plane fitting.

An example of a semi-automatic feature detection method in point cloud data is the work by Pauly, Keiser & Gross [26] where the likelihood of a point lying on a feature is calculated by analysing the surface curvature of the point and its neighbourhood. What stands out about their approach is that the point classification is done at multiple scales of point neighbourhood in order to identify an 'optimal' neighbourhood size for a particular model, which in turn yields better feature detection as a result.

Commonly the metric for the simplification accuracy or 'performance' of such an algorithm is the distance measure between the vertices of the original 'correct' dense model and the produced simplified model. The Hausdorff distance is a very common metric for the similarity or 'closeness' of two geometric models, making it very suitable for several mesh simplification works [21] [22] [11].

3.4 Directly related works and mehtods

Since the focus of this work is on the straight line features of indoor geometry, finding intersections between fitting planes on the vertices of the mesh is a popular and efficient approach of detecting these in a triangulated mesh as well as in point cloud data, which would make them suitable for feature detection in this project. This would require a way to identify these planes in a 3D mesh.

The work by Gilani, Awrangjeb & Lu [12] proposed a method for detecting roof-features in areal point cloud scans of buildings. Using a combination of Principal Component Analysis and a normal estimation method (LRSCPK),

sharp edges were identified. Theoretical planes were then fitted to the data between the sharp features whose interesection results in the final features.

Salinas, Lafarge & Alliez [28] applied surface fitting to the mesh decimation problem. Information regarding the mesh structure was identified by fitting planar 'proxies' to the data set, i.e identifying planar surface components in it. This information was then used to guide the edge collapse operator in order to preserve this structure.

The work by Fleishman, Cohen-Or & Silva [10] was an example of an implementation of a moving least squares planar fitting method that was noise resistant as well as preserving sharp edges between piecewise smooth surfaces in a point cloud model. Their method proved useful to reconstruct surfaces in noisy point clouds.

In order to evaluate the difference between two linear features in the mesh, a distance measure is needed. In this work the midpoint distance was used, evaluated in the work by Wirtz & Paulus [30]. Section 4.5.1 describes how this measure was used.

4 Algorithm design

4.1 Overview

The objective of the feature detection algorithm was finding the intersection between fitted planes corresponding to the perceived surfaces in the mesh data. The approach could be divided into three discrete steps:

- Planar surfaces detection.
- Extraction of linear features and corner points.
- Feature adjustment.
- Validation.

The implementation depended on three variables: Plane distance threshold: d_{th}, Maximum angle difference: d_a and the Closeness threshold d_c. The exact function of these variables will be explained in the descriptions below.

4.2 Planar surface detection

The first step of the implementation was to find naturally occurring planes in the mesh. This was achieved by performing a repeated randomly seeded LSQ (Least Squares) fitted until all vertices in the mesh had been fit to a surface. The Least Squares method solves the problem of fitting a model function, in this case a function defining a plane in Euclidean 3-space to a set of data points (vertices). The measurement of how well the model is fit to these data points is the sum of the squared residuals, i.e the error between the predicted model and

the actual data points. In this work r_i is the distance between the proposed plane and a vertex fit to that plane:

$$S = \sum_{i=1}^{n} r_i \tag{1}$$

By minimizing the value of S the model is considered closer or better fitted to the data points. In the context of this work the model corresponds to a plane defined by an origin point and a normal vector.

The Fitting algorithm in pseudo-code:

1. Select a random triplet of adjacent vertices $v_{1-3} = v$ from the remainder of the vertices in the mesh that have not been fitted to a plane. **If no such triplets exist, the algorithm terminates.**
2. Perform a LSQ fit on v to obtain the plane p_v.
3. Select all neighbouring vertices v_f of v whose distance from the plane p_v is less than the distance threshold: d_{th} **AND** have not yet been fitted to another plane. **If no such points are found, proceed to step 6.**
4. Perform a LSQ fit on the union of v and v_f to obtain the plane $p2_v$
5. Set v as the union of v and v_f: $v = v \cup v_f$ **Return to step 3**.
6. Save the plane $p2_v$ as the plane representing the geometry spanning $v \cup v_f$.
7. **Return to step 1.**

This algorithm performs a search from an origin point (a randomly selected adjacent vertex triplet), gradually updating the LSQ fit iteratively until no new neighbours can be found due to a sudden change in distance from the previous plane or if the new neighbours have already been fit to another plane. A plane and its vertices are discarded if it is too small, i.e the plane has less than a certain number of vertices associated with it. In the case of Figure 3, planes spanning 25 vertices or less were removed to avoid small non-planar clusters of vertices being identified as such. This specific value was found by running a number of tests where the aim was to represent the most prominent planar features of the mesh without losing too much of the finer detail. Deciding exactly at how many vertices such a cutoff should be optimal, relies once again on the judgment of the user as well as on the vertex density of the mesh itself. A cutoff of 25 vertices proved suitable for this particular set of meshes. Figure 3:Middle shows the result of this algorithm on one of the spatial map data sets.

Due to the disconnect between patches observed in Figure 2 and the inherent randomness of selecting any triplet from the mesh to use as a starting point for the LSQ fit, the neighbour driven approach to the LSQ-fit produces planes that are 'bound' by their patch, i.e. the plane cannot extend beyond the patch it was seeded into. In order to obtain a reasonable approximation of the surfaces in the mesh, a second merging step is needed.

In this work, two planes are considered part of the same surface if the angle between them (the difference in angle between their normals) is small enough and the minimum distance of the groups of vertices they individually span is not too great. In pseudo-code:

The merge operation pseudo-code:

1. Select all planes generated by the plane fitting algorithm as the set P.
2. Set the repeat flag r to 'true'.
3. **IF** r equals 'false', the operation terminates.
4. Set the repeat flag r to 'false'.
5. For every unique and 'un-merged' pair p_1, p_2 of planes in P:
    1 Calculate the minimum distance min_d between the two closest vertices in p_1 and p_2.
    2 Calculate the difference in angle p_a between the normals of p_1 and p_2.
    3 **IF** $min_d < d_c$ **AND** $p_a < d_a$:
        - Perform a new LSQ plane fit on the sum of vertices in both p_1 and p_2
        - Store the resulting plane in p_1 and mark p_2 as 'merged'.
        - Set the repeat flag r to 'true'.
6. Return to step 3.

The merging step iteratively checks whether each pair of planes are suitable to be merged. If a pair of planes fulfill the requirements mentioned above, they are merged by performing a new LSQ fit on the union of vertices in both of the planes. The new plane is stored and the previous two are discarded. In order to properly merge all planes including any newly merged ones, the algorithm repeats the iterative check of all planes if at least one merge was performed in the previous iteration. When the algorithm does not find any more suitable pairs to merge, the algorithm terminates. Figure 3, bottom illustrates the result of applying the merging operation on the example of the surface fitting algorithm.

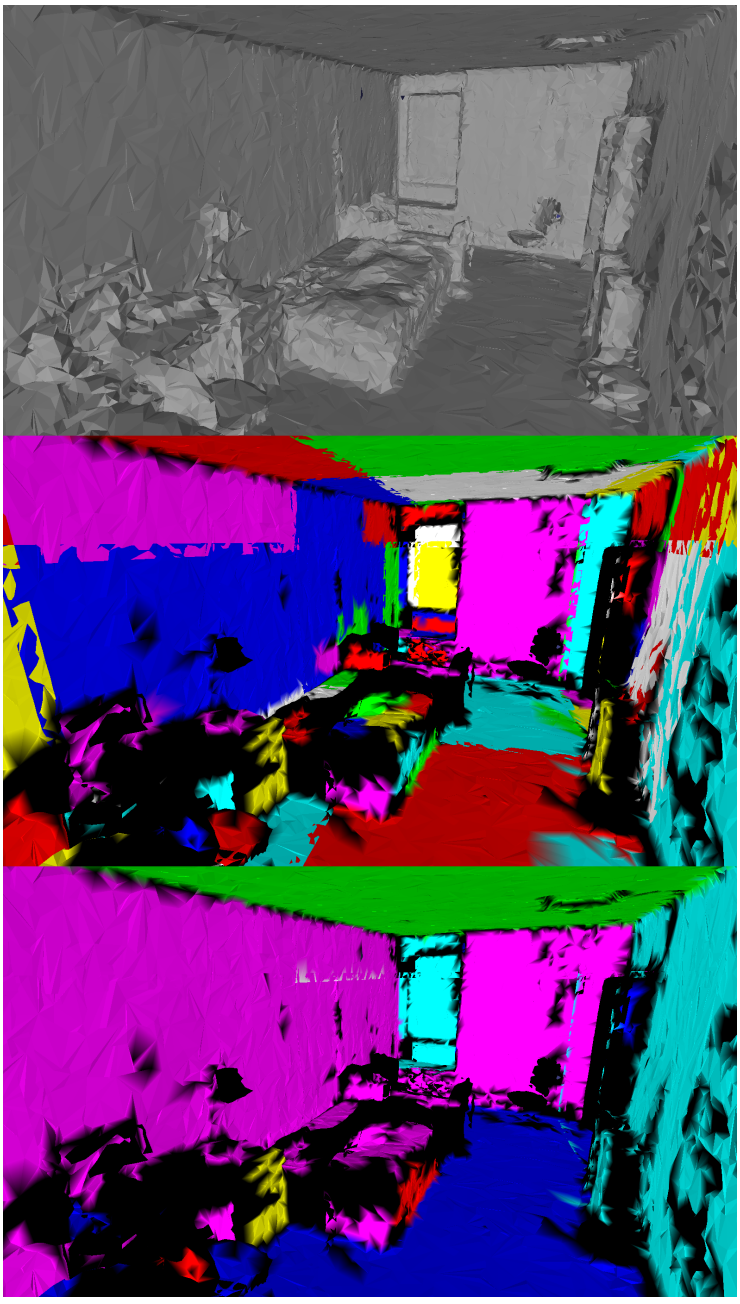

Figure 3: **Top**: Mesh reference **Middle**: Result of the LSQ-fit algorithm, each color corresponds to an individual plane. *Black vertices were not fit to any plane.* Note that colors are repeated. **Bottom**: The result of the merging operation on the planes shown in the middle image.

4.3 Extraction of linear features and corner points

With the planar features in the mesh identified, it is now possible to find the implied straight-line features defined by the intersection of these planes. The line present at the intersection between two planes in 3-space is defined as (assuming the planes are not parallel):

$$P_1 : n_1 = a \tag{2}$$

$$P_2 : n_2 = b \tag{3}$$

$$(1) v_l = n_1 \times n_2 \tag{4}$$

$$(2) n_1 * p_l = a \tag{5}$$

$$(3) n_2 * p_l = b \tag{6}$$

$$(4) L = p_l + v_l * t \tag{7}$$

The line along the intersection of two planes P_1 and P_2 will be parallel to the result of the cross product of their normals $v_l(1)$. To define the line a point on the line is needed, this point p_l must be present in both planes, i.e on the sougth after line(2,3). Solving for the point p_l, the line tracing the intersection between the planes P_1 and P_2 can now be defined(4). The intersections of each pair of planes found in the previous step are calculated this way.

Some filtering of the intersections is required since the cartesian product of intersections between all unique pairs of planes in the set generated by the merge operation produces lines outside the bounding box of the mesh. Therefore only the intersections that lie within the bounding box of the mesh are considered 'valid' features. Another condition on the intersections is that similarly to the merge step, the vertex clusters spanning each plane must be within a certain distance of one another to not produce features in distant geometry. The reasoning behind this is that if two distant planar features that are not connected or particularly close, their intersection would not represent a feature of the mesh but rather a theoretical line of two infinite planes. Since this kind of feature does not fall under the definition of interesting defined in section 3.2, they must be discared.

The line obtained by intersecting two planes is an infinitely long line in 3-space, naturally the features sought after are not. To find an approximation of how long the line is i.e find the two edge points of the line, the support of the vertices fitted to the planes involved in the intersection are used. By projecting all vertices involved onto the line, the two points that are farthest apart approximately represent the edges where the planes end, which in turn is where the line between them should also end. Formally:

$$Proj_{pts} = [(v_1 * L), (v_2 * L)...(v_n * L)] \tag{8}$$

From these projected points the two candidates who are the furthest apart(p_1 and p_2), are set as the endpoints of the line-type feature. The result is a 3D

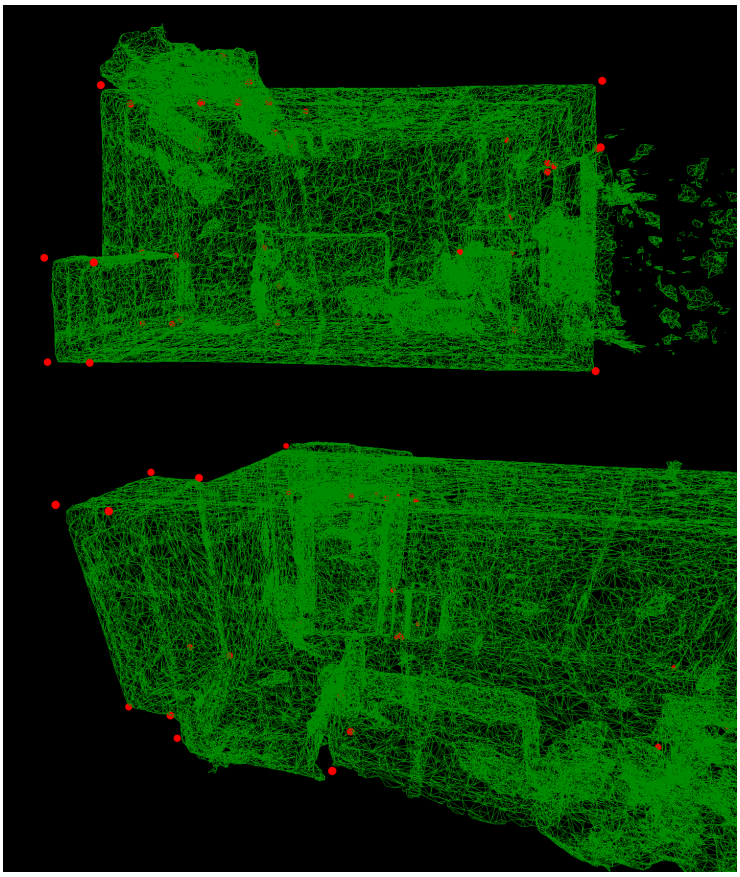

Figure 4: An example of the corners (red) generated using the data set D_2. **Top:** Top-down view **Bottom:** Angled view.

line between points p_1 and p_2 supported by the vertices in both planes used to find it which define a line-type feature. Figure 5: Middle shows an example of the features generated this way.

4.4 Feature adjustment

Similarly to how line-type features can be acquired from the intersection of two planes, an approximate corner in the mesh can be found by intersecting three planes. The formal definition assuming no one of the three pairs of planes are parallel, the point of intersection p can be found:

$$p = (|n_1 n_2 n_3|)^{-1}[(p_1 * n_1)(n_2 \times n_3) + (p_2 * n_2)(n_3 \times n_1) + (p_3 * n_3)(n_2 \times n_1)] \quad (9)$$

Note: In the above equation $*$ and $\times$ represent the dot and cross products

respectively and the expression ($|n_1 n_2 n_3|$) is the determinant of the matrix where n_{1-3} are written in sequence and each plane is defined as a point p_{1-3} and a normal vector n_{1-3}. Solving the equation gives the point p which represents the corner defined by the intersection. The above equation is applied to every unique triplet of planes found in the mesh.

It can be assumed that features defined by a triplet of planes would coincide at the corner defined by the same triplet of planes. Once again the cartesian product of corners produced by intersecting every triplet of planes requires some filtering. Only the corners within the bounding box of the mesh are considered. Out of these, only the ones that were produced by adjacent or significantly close planes are kept, while the rest are discarded. Figure 4 illustrates an example of the filtered corners produced by this part of the algorithm.

With a reasonable approximation of the corners available the feature endpoints can be adjusted in order to have the features coincide at these corner points to increase the consistency of where the line-type features meet. This may not always produce a 'better' feature. However, once again the problem of far away corners affecting features with no vertex support near them presents itself. Thus the features are only adjusted when the feature endpoints are within a certain distance to the corner, which is defined by the closeness threshold d_c used in the LSQ-fitting part of the algorithm. Iteratively the distance between each line-type features endpoints and their relevant corners is measured and if it proves less than d_c, the endpoint is moved to coincide with the corners position. Figure 5: Bottom illustrates the adjustment of the features shown in the Middle image. Once the features have been adjusted, the feature extraction process is complete.

4.5 Validation

In order to measure the accuracy of the features that the implementation produces, a validation step is needed. The validation step compares the implementation-produced features and predetermined features representing a humans interpretation of what these features might be. The higher the similarity between the two sets, the better the accuracy of the algorithm following the definition in section 2.2.

4.5.1 Distance measure

In the context of this project, a measure of two separate line type features is required. Wirtz & Paulus [30] evaluated some of the most common distance measures between line segments. Among these is the midpoint distance measure which was chosen to determine similarity between two features in this work. Two identical lines at the exact same position have the midpoint distance 0. As their relative angle, distance and shape changes, the value increases. Figure 6 illustrates a few generic examples of midpoint distances.

The mathematical definition of the midpint distance:
$d_{midpoint}(l_1, l_2) = ||p_1, l_1 - p_1, l_2|| + ||p_2, l_1 - p_2, l_2|| + 3||m_1 - m_2||$ where p_x, l_y

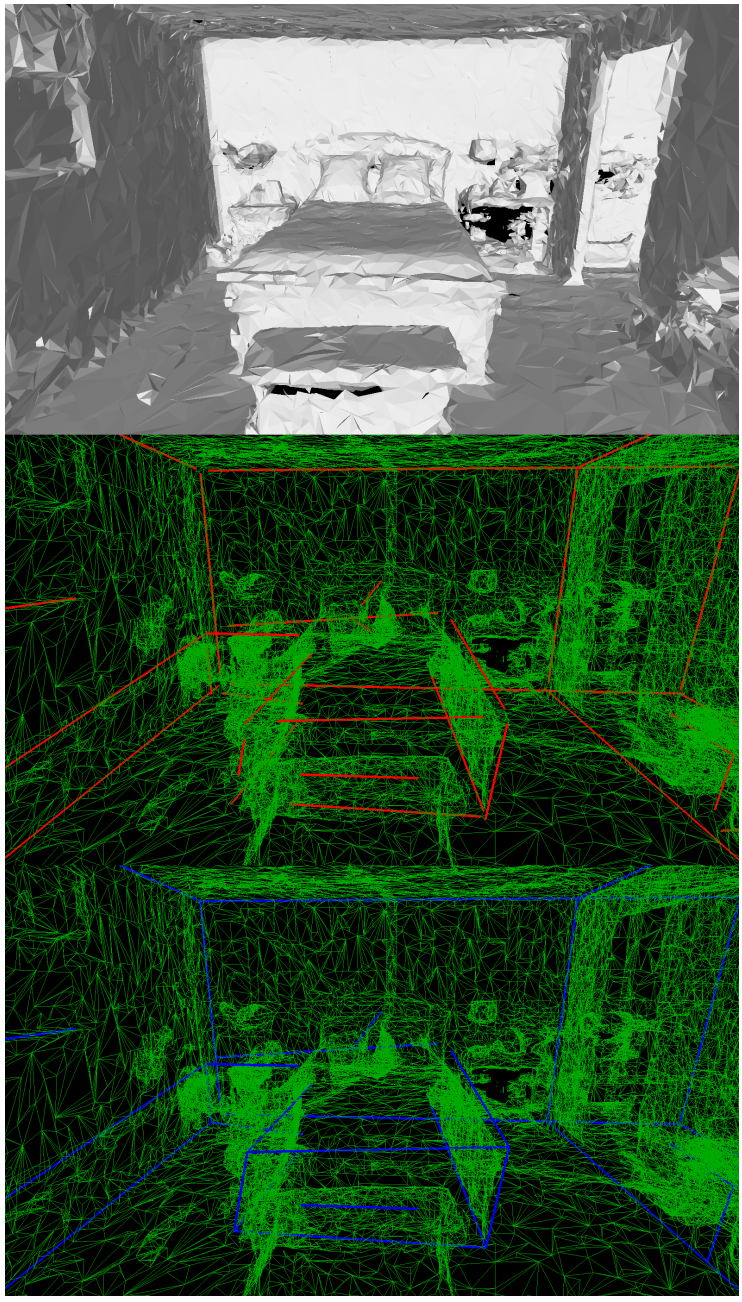

Figure 5: Examples of features found in the data set D3. **Top:** Mesh reference. **Middle:** Features (red) before corner adjustment. **Bottom:** Features (blue) after corner adjustment.

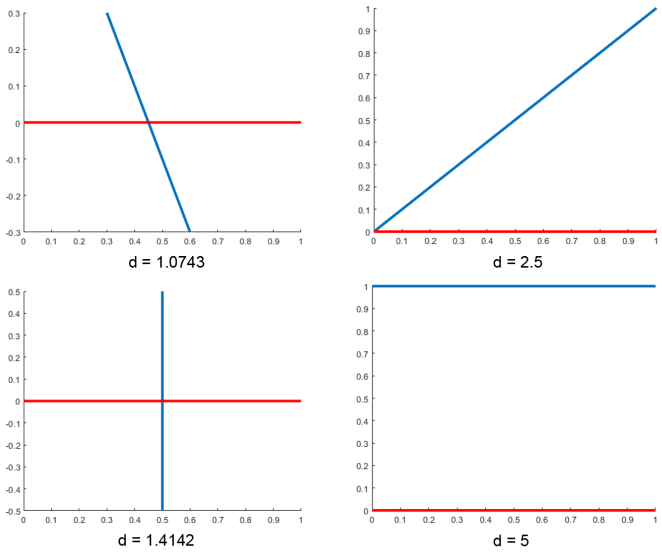

Figure 6: Midpoint distance reference examples.

and p_x, l_z represent the two pairs of closest endpoints of both lines l_1 and l_2 respectively. m_1 and m_2 represent each lines midpoint. The midpoint distance does not correspond to a physical distance but is rather a similarity measure between two lines.

The reason for choosing the midpoint distance measure is twofold, firstly the algorithm is well suited to N-Dimensional lines which trivializes its conversion from 2D to 3D. Secondly the distance value increases gradually with regards to difference in length of the two lines, their relative angle and the distance between their mid and endpoints (Figure 7). Since the distance increases 'smoothly' without any sudden jumps the risk of identifying two intuitively dissimilar lines as close is reduced by using this distance measure. For a more detailed explanation of how these distance measures were evaluated as well as how they compare to one another, see the above cited work by Wirtz & Paulus.

4.5.2 Validation features

With a distance measure in place, measuring the similarity between two line type features is now possible. The validation-features used to generate the results in section 5 were chosen by placing a number of linear features by hand where they intuitively appear.

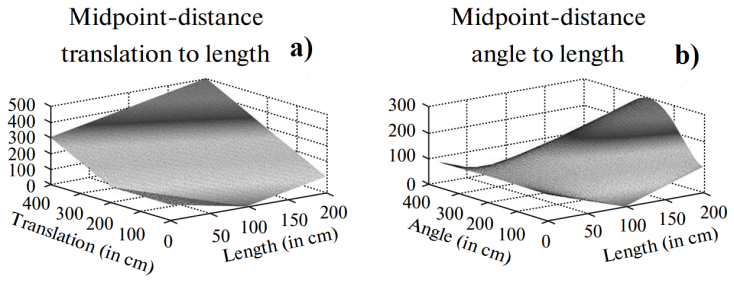

Figure 7: **a**) A 3D graph of the distance between two lines as a function of the translation and length of the second line. **b**) The distance between two lines as a function of the length and angle of the second line. Note that in both cases the progression of the distance resembles a smooth curve without plateaus or sudden peaks or valleys.

5 Results

The feature detection algorithm presented in this work was tested on five different spatial maps of indoor environments $D_1 - D_5$. The validation features selected for testing consisted of different sized features in order to challenge the versatility of the detection method.

Due to the random nature of the LSQ-fitting step, the algorithm produced some varying results depending on where the resulting plane's entry points were randomly selected. Because of this, two tables are presented below, one details a single individual run and validation of the feature detection and the other the average of five runs on the same data set. This allows for comparison between the average case when using these methods and an example of what results a one-time user could expect. Figure 14 illustrates the difference between different iterations of the feature detection on the same data set.

The scores in three rightmost columns of both result tables represent the midpoint distance (Section 4.5.1). Figure 6 presents some reference values of midpoint distances. The individual score of a validation feature is defined as the midpoint distance between it and the 'best' detected feature (lowest midpoint distance) compared to it. The total feature detection score of the mesh is presented as the average score of all validation features.

Figure 8 shows the difference between the validation features and a result of the algorithm.

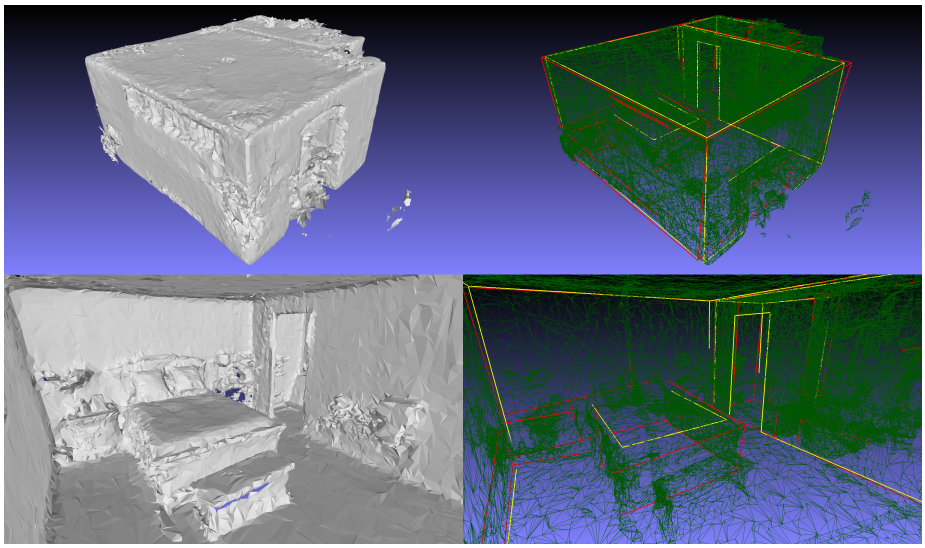

Figure 8: **Right:** Example of the disparity between the validation features (yellow) and the detected features.(red) **Left:** Mesh reference.

5.1 Table 1: Individual case

	N.o Val Feats	Avg. Feat score	Best Feat score	Worst Feat score
D_1	23	3.9296	2.2974	7.2705
D_2	20	0.7929	0.1764	3.1433
D_3	25	2.5439	0.2008	6.7950
D_4	35	2.0840	0.1591	6.7712
D_5	31	2.2148	0.1602	6.3120

5.2 Table 2: Five iterations

	N.o Val Feats	Avg. score	Best Avg. Feat score	Worst Avg. Feat score
D_1	23	4.2989	4.1626	4.4295
D_2	20	0.9526	0.8311	1.0539
D_3	25	2.2768	2.0362	2.4649
D_4	35	2.2702	2.0236	2.5184
D_5	31	1.9346	1.6303	2.3184

Figures 9,10,11,12 and 13, show a flat face render of the data sets used in this work along with a wire-frame example of the features detected (red) in the data sets $D_1 - D_5$ respectively.

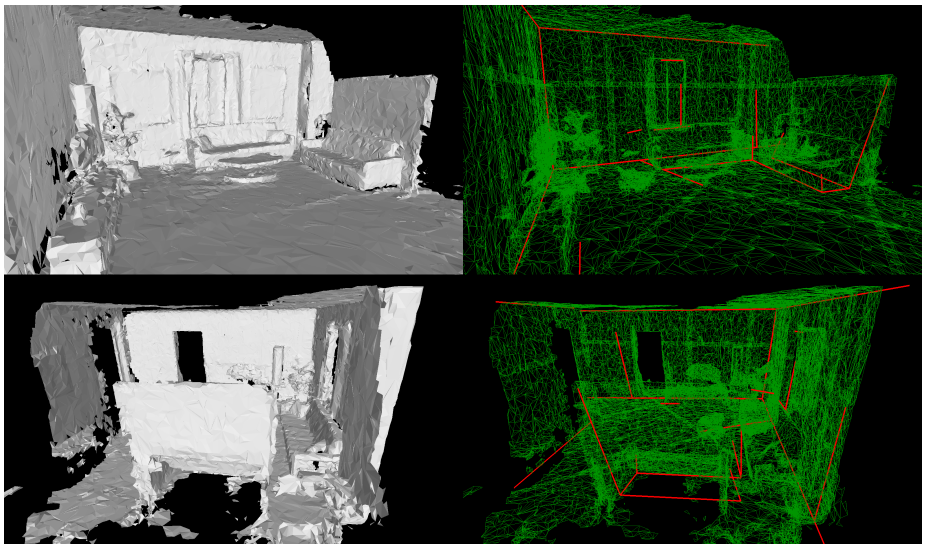

Figure 9: Data set 1 (D_1)

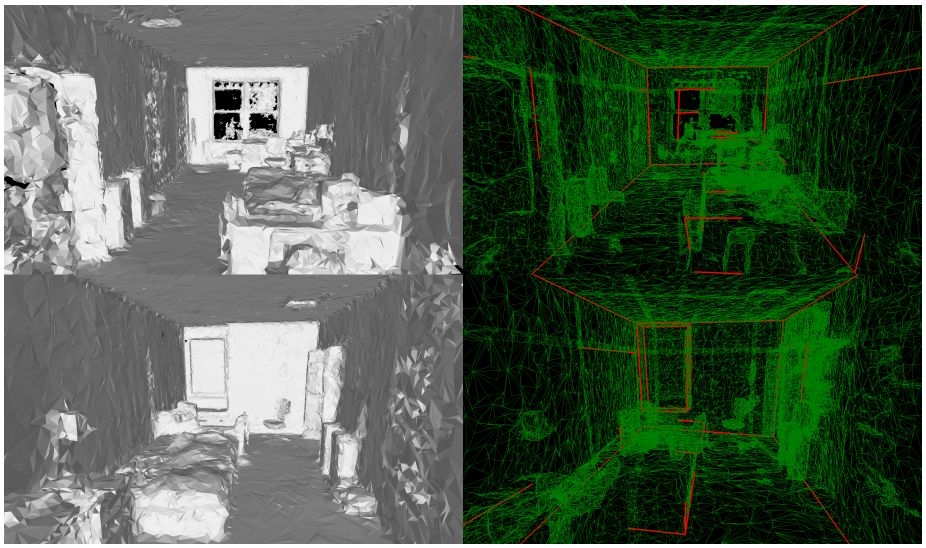

Figure 10: Data set 2 (D_2)

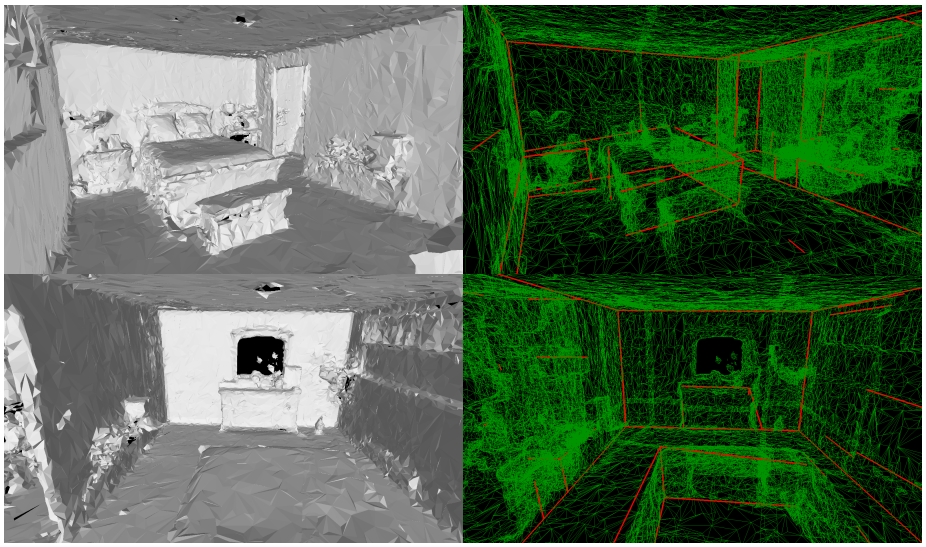

Figure 11: Data set 3 (D_3)
,

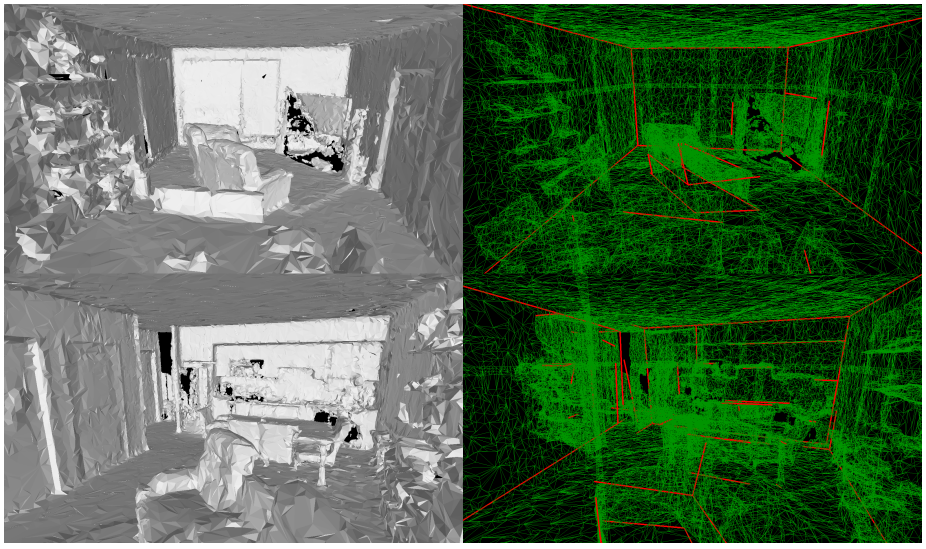

Figure 12: Data set 4 (D_4)

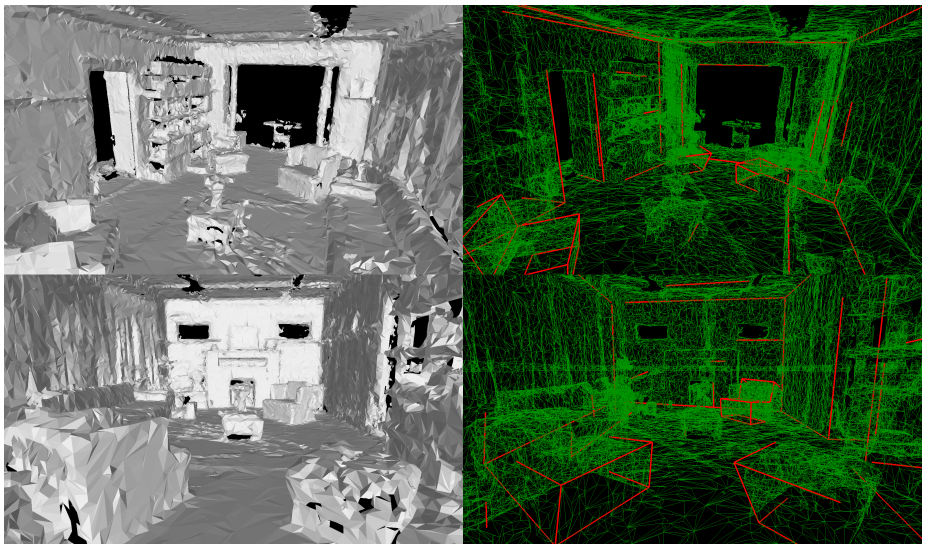

Figure 13: Data set 5 (D_5)

6 Discussion & Conclusions

The first impression of the results was that this feature detection method was not very well suited to the first data set D_1. An overview of the data sets found in Appendix 1 reveals a clue to a possible reason, the data set lacked a full map of the room's walls with consistent corners compared to the other sets. As corners were used to refine the length of the detected features, this might have resulted in relatively worse features more dissimilar to the validation features.

Comparing the remaining measured distances to the reference distances presented in Figure 6, it would seem the feature detection produced a decent set of approximations of the validation features. For applications where an inexact, incomplete approximation of such features would be enough, these methods would suffice to provide them.

However as an algorithm trying to identify 'exact' line-type features in an arbitrary 3D mesh, the results were arguably quite poor. A contributing factor was the quality of the mesh itself as described in section 1.4, but the main problem was the design of the algorithm itself. Firstly it is dependent on too many variables (Distance threshold: d_{th}, Maximum angle difference d_a and Closeness threshold d_c) which made it a very 'manual' and unstable algorithm, placing the responsibility of producing a qualitative result on the user. Secondly because of these variables the design is not self-adjusting when it comes to mesh size and vertex density. A semi-automatic or automatic algorithm design might have avoided some of these problems and made the implementation more versatile.

There is also an argument to be made about the definition of an 'exact' or

even a 'good' result in the context of the hand crafted validation-features. The definition of a perfect feature is when its score is exactly zero. There is however no measure as to how good or bad a feature score of 1, 2 or 100 is objectively. The quality of features is still subjective in essence and thus reliant on the whims of a potential user. A potential way to better understand the quality measure of features would then be to simply let users decide which features are good fits and which ones aren't showing them visual examples of feature-score pairs.

Another point on the subject of trying to find an exact or correct result when validating features is whether there is a greater need for accuracy or responsiveness and performance. Many graphical applications utilize a number of approximate calculations either in situations where details pass by a user too quickly to be noticed or are placed in the background and not paid full attention to. These kinds of approximations are made in places where the user experience is affected the least whilst trying to improve the overall performance or responsiveness of the application. The same principle can apply to holograms in an AR application where their exact placement or dimensions might not be paramount for the user's experience.

Originally the implementation was supposed to be a distributed program on the Hololens headset itself allowing the user to manipulate and visualize the features in the AR space. This was however decided to be outside the scope of this work and a theoretical implementation of the feature detection algorithm using Matlab was chosen instead. An implementation on the Headset itself would require great consideration of the performance aspect as well as storage space, if the user would for example move around generating new meshes as they went. This could prove difficult for the Hololens limited resources to handle without a fairly optimized algorithm and certainly outside what the proposed algorithm in this work would be capable of.

Another area that would need attention in order to improve the result of this algorithm is to eliminate or at least control the randomness factor as result of selecting random starting locations for the LSQ-fitting step. By eliminating this the algorithm would have more predictable and stable results which would make recreating exact results possible.

A failed attempt was made to try and find line-type features using the SOD/ESOD operators described in the work by Hubeli, Meyer & Gross [18] [17]. The SOD and ESOD operators used a measure of the angle between two triangles that share an edge, assigning that edge a weight. The higher the value the more likely the edge was located on a crease or edge of a mesh. The SOD operator is defined as:

$$w(e) = cos^{-1}(\frac{n_1}{||n_1||} \cdot \frac{n_2}{||n_2||}) \tag{10}$$

The ESOD operator used the same equation but replaces the triangle-normals n_1 and n_2 with the normals of the vertices located opposite the edge. See the provided reference above for further details on SOD/ESOD. When these operators were applied to the Hololens meshes there was not enough consistency

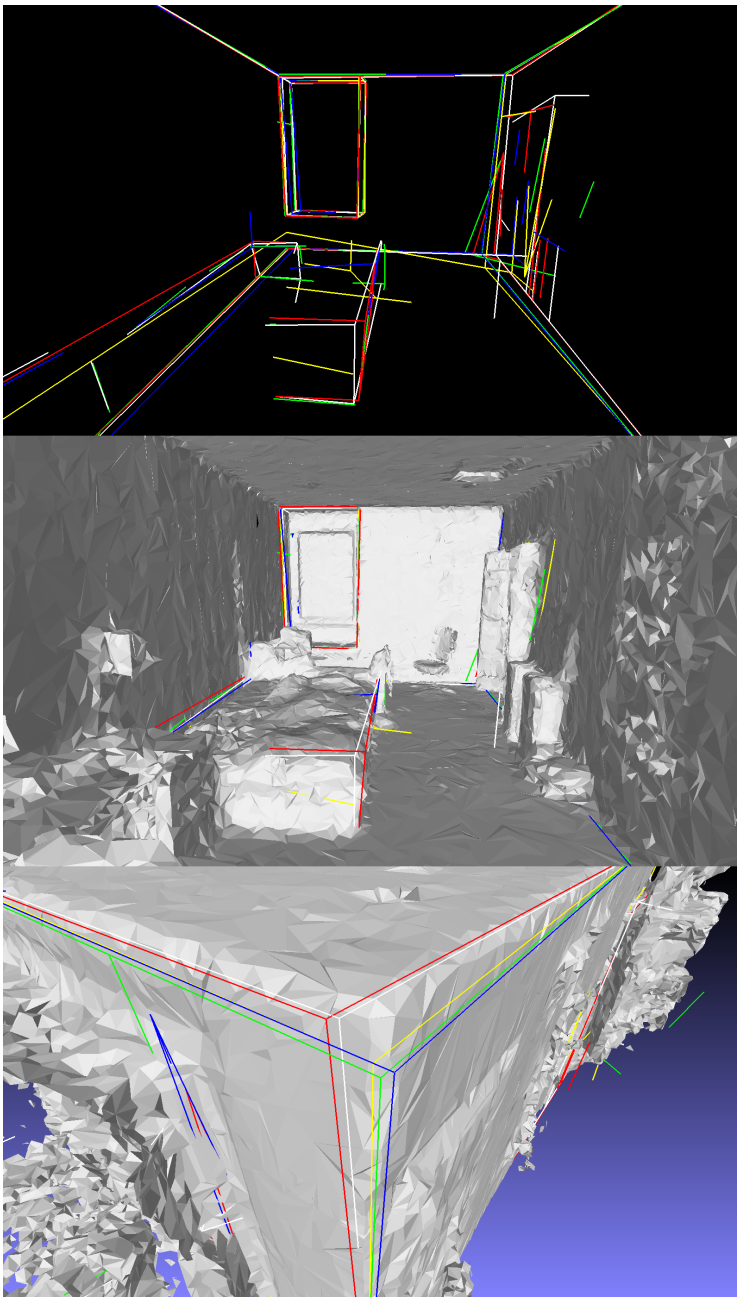

Figure 14: A visual representation of the difference in results between five separate runs of the feature detection algorithm. Each colour corresponds to a different run resulting features. **Top:** Features overlaid. **Middle:** Features overlaid With reference mesh. **Bottom:** Corner close up.

in the result to allow for further feature refinement.

An idea for such a refinement was to trace a path along the edges with a weight above a certain threshold, breaking the path at forks and greatly differing angles between the edges as to detect where one feature is connected to another. This proved to be an insufficient method given the Hololens data, as it was much too noisy and the discontinuity between mesh patches (Figure 2) which in turn produced noisy SOD/ESOD weights (Figure 15).

The usefulness of AR in a wide variety of applications was shown by the litterature in sections 1.1 and 1.2. Although quite a bit more elaborate than the methods used in this work, it would not be unreasonable to assume that similar applications could benefit from lower level algorithms like a feature detection algorithm described in this work. As the lower level framework for AR technology improves and expands the ease of developing higher level applications and algorithms should also improve.

The majority of the related works on feature detection or mesh optimization were not practically applied and of a more theoretical nature than a potential deployed implementation on an AR device. A difference between such an implementation and the theoretical tests conducted in this work is the fact that an AR device would continuously map a users surroundings (in the case of the Hololens) while the results shown in this work was achieved using a complete pre-scanned sample map. If the feature detection algorithm is not fully automatic, different variables controlling the scale or density of features for example might not be suitable for newly scanned portions of the spatial map. As the spatial map extends, features already detected could be in need of updating if an edge is a part of a larger feature whose underlying mesh was not completely scanned in at the time of detection. The same problem might arise if the user is allowed to interact with the features in real time without overriding a users changes with further detection or causing incorrect detections.

7 Future work

As mentioned in section 5, a side effect of this feature extraction method was that the result varies as a result of the randomly seeded starting points used in the plane-detection step. An implementation that does not rely on a random starting point would benefit the consistency of the features produced. Alternatively, choosing these random points more carefully could also stabilize the result by not allowing starting positions along creases or in pits in the mesh for example. A way to achieve this could be to consider the neighbourhood of the starting point before the actual algorithm starts.

The variables defined in section 4.1 were found to be the seemingly optimal values for these sample-mesh data sets by iteratively increasing each variable and finding the best average result produced with them. A semi-automatic approach would be useful here to detect the best variable values for each mesh individually. An option could be preprocessing of the target mesh, gathering more information about the individual parts of the mesh and perhaps classifying

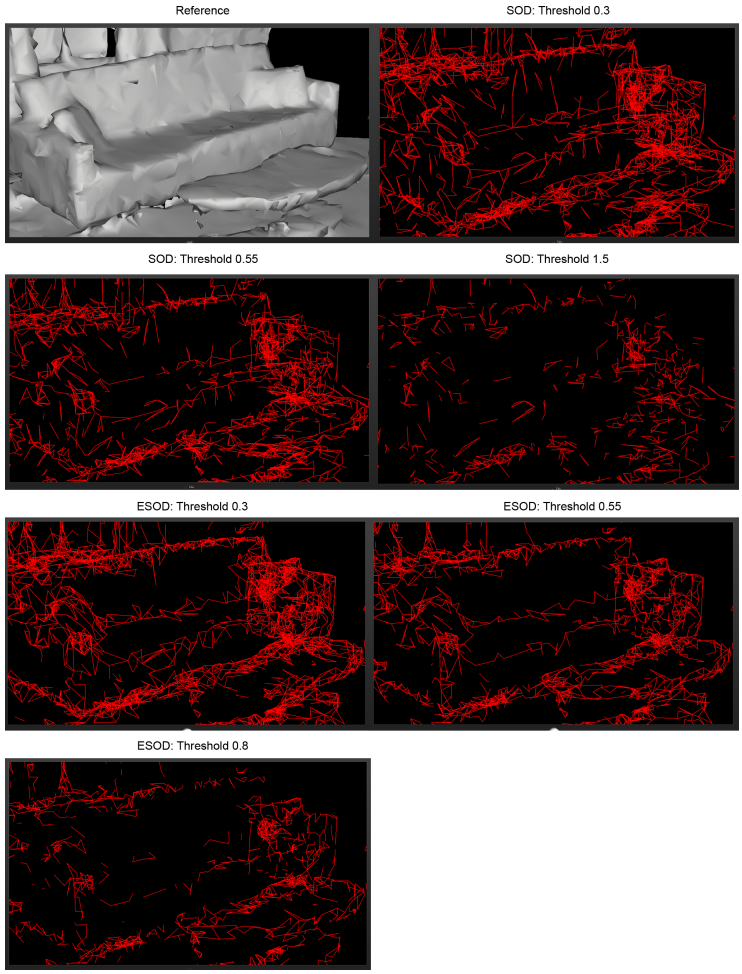

Figure 15: Results of the SOD/ESOD operators on a section of with different thresholds. Notice how as the threshold increases, noise is reduced but at the cost of relevant feature information is lost along the edges of the couch for example.

them or weighting the vertices to help the feature detection performance later on.

Perhaps the most obvious improvement to these methods when it comes to feature detection would be the detection of additional feature types. Circles and curves are two examples of other feature types.

Another way to improve the result of feature detection methods on the Hololens spatial map data would be to 'improve' the data itself. Simplification and smoothing algorithms could be used to remove unwanted artefacts or reduce some of the noise present in the data.

It was shown in this work that feature detection is possible with the data provided by the Hololens headset. A natural follow up project would be to implement this to run on an actual Hololens hardware. A direct implementation of this algorithm would not be advised however since they are not written efficiently and considering the limited performance available on the Hololens headset, performance would have to be of a higher priority.

If an application would require real-time feature detection an option could be to move the majority of the workload to the GPU instead of the current CPU implementation. DeCoro & Tatarchuk [8] implemented real time mesh simplification by utilizing the parallel computing capabilities of the GPU. For example: With proper synchronization, the same parallelism could be applied to the plane-fitting step where multiple surfaces are fitted on parts of the mesh to later be merged together to form planes to be used for the feature detection step. In relation to the previous point, the Hololens headset is equipped with a relatively powerful GPU component which makes this approach a possible candidate for a distributed implementation.

Another avenue for increasing performance would be to compromise the exactness of the result, utilizing approximate calculations of planar features, line-type features and corner points as an example. As discussed in section 6, some AR applications would be more time and by extension performance sensitive without the need of an exact result.

8 Acknowledgements

None of the data sets used in this work were generated using an actual Hololens headset. The data sets were all provided along with the Hololens emulator. The specific settings used to generate these meshes were not known at the time of writing. The algorithm design and implementation was done in MathWork's Matlab.

References

[1] Kiljae Ahn, Dae-Sik Ko, and Sang-Hoon Gim. A study on the architecture of mixed reality application for architectural design collaboration. In *International Conference on Applied Computing and Information Technology*, pages 48–61. Springer, 2018.

[2] Misty Antonioli, Corinne Blake, and Kelly Sparks. Augmented reality applications in education. *The Journal of Technology Studies*, pages 96–107, 2014.

[3] Lisa Avila and Mike Bailey. Augment your reality. *IEEE computer graphics and applications*, 36(1):6–7, 2016.

[4] EZ Barsom, M Graafland, and MP Schijven. Systematic review on the effectiveness of augmented reality applications in medical training. *Surgical endoscopy*, 30(10):4174–4183, 2016.

[5] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia tools and applications*, 51(1):341–377, 2011.

[6] Henry Chen, Austin S Lee, Mark Swift, and John C Tang. 3d collaboration method over hololens™ and skype™ end points. In *Proceedings of the 3rd International Workshop on Immersive Media Experiences*, pages 27–30. ACM, 2015.

[7] Hung-Lin Chi, Shih-Chung Kang, and Xiangyu Wang. Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. *Automation in construction*, 33:116–122, 2013.

[8] Christopher DeCoro and Natalya Tatarchuk. Real-time mesh simplification using the gpu. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 161–166. ACM, 2007.

[9] Gabriel Evans, Jack Miller, Mariangely Iglesias Pena, Anastacia MacAllister, and Eliot Winer. Evaluating the microsoft hololens through an augmented reality assembly application. In *Degraded Environments: Sensing, Processing, and Display 2017*, volume 10197, page 101970V. International Society for Optics and Photonics, 2017.

[10] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T Silva. Robust moving least-squares fitting with sharp features. In *ACM transactions on graphics (TOG)*, volume 24, pages 544–552. ACM, 2005.

[11] Martin Franc and Václav Skala. Mesh simplification with respect to a model appearance. 2006.

[12] Syed Ali Naqi Gilani, Mohammad Awrangjeb, and Guojun Lu. Robust building roof segmentation using airborne point cloud data. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 859–863. IEEE, 2016.

[13] Stefan Gumhold, Xinlong Wang, and Rob S MacLeod. Feature extraction from point clouds. In *IMR*. Citeseer, 2001.

[14] Matthew G Hanna, Ishtiaque Ahmed, Jeffrey Nine, Shyam Prajapati, and Liron Pantanowitz. Augmented reality technology using microsoft hololens in anatomic pathology. *Archives of pathology & laboratory medicine*, 142(5):638–644, 2018.

[15] Paul Hockett and Tim Ingleby. Augmented reality with hololens: Experiential architectures embedded in the real world. *arXiv preprint arXiv:1610.04281*, 2016.

[16] MA Hoffman and JB Provance. Visualization of molecular structures using hololens-based augmented reality. *AMIA Summits on Translational Science Proceedings*, 2017:68, 2017.

[17] Andreas Hubeli and Markus Gross. Multiresolution feature extraction for unstructured meshes. In *Proceedings of the Conference on Visualization'01*, pages 287–294. IEEE Computer Society, 2001.

[18] Andreas Hubeli, Kuno Meyer, and Markus H Gross. Mesh edge detection. *CS technical report*, 351, 2000.

[19] Mohammad Hussain, Yoshihiro Okada, and Koichi Niijima. Efficient and feature-preserving triangular mesh decimation. 2004.

[20] Hannes Kaufmann and Dieter Schmalstieg. Mathematics and geometry education with collaborative augmented reality. In *ACM SIGGRAPH 2002 conference abstracts and applications*, pages 37–41. ACM, 2002.

[21] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In *Proceedings of Seventh Annual IEEE Visualization'96*, pages 311–318. IEEE, 1996.

[22] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics interface*, volume 98, pages 43–50, 1998.

[23] Chang Ha Lee, Amitabh Varshney, and David W Jacobs. Mesh saliency. In *ACM transactions on graphics (TOG)*, volume 24, pages 659–666. ACM, 2005.

[24] Mehdi Mekni and Andre Lemieux. Augmented reality: Applications, challenges and future trends. *Applied Computational Science*, pages 205–214, 2014.

[25] Andrew YC Nee, SK Ong, George Chryssolouris, and Dimitris Mourtzis. Augmented reality applications in design and manufacturing. *CIRP annals*, 61(2):657–679, 2012.

[26] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale feature extraction on point-sampled surfaces. In *Computer graphics forum*, volume 22, pages 281–289. Wiley Online Library, 2003.

[27] Philip Pratt, Matthew Ives, Graham Lawton, Jonathan Simmons, Nasko Radev, Liana Spyropoulou, and Dimitri Amiras. Through the hololens™ looking glass: augmented reality for extremity reconstruction surgery using 3d vascular models with perforating vessels. *European radiology experimental*, 2(1):2, 2018.

[28] David Salinas, Florent Lafarge, and Pierre Alliez. Structure-aware mesh decimation. In *Computer Graphics Forum*, volume 34, pages 211–227. Wiley Online Library, 2015.

[29] Bruce Thomas, Benjamin Close, John Donoghue, John Squires, Phillip De Bondi, Michael Morris, and Wayne Piekarski. Arquake: An outdoor/indoor augmented reality first person application. In *Digest of Papers. Fourth International Symposium on Wearable Computers*, pages 139–146. IEEE, 2000.

[30] Stefan Wirtz and Dietrich Paulus. Evaluation of established line segment distance functions. *Pattern Recognition and Image Analysis*, 26(2):354–359, 2016.

[31] Rafal Wojciechowski, Krzysztof Walczak, Martin White, and Wojciech Cellary. Building virtual and augmented reality museum exhibitions. In *Proceedings of the ninth international conference on 3D Web technology*, pages 135–144. ACM, 2004.

[32] Longyu Zhang, Sifeng Chen, Haiwei Dong, and Abdulmotaleb El Saddik. Visualizing toronto city data with hololens: Using augmented reality for a city model. *IEEE Consumer Electronics Magazine*, 7(3):73–80, 2018.