A System for Object Query in Point Clouds Using Feature-Feature Matching

MICHAL STANIASZEK

SUPERVISOR: JOHN FOLKESSON
EXAMINER: DANICA KRAGIC

Abstract

In this report, we will describe our implementation of a system for retrieving instances of objects using feature based matching techniques. We use point cloud data gathered using RGB-D sensors in an office environment over a period of approximately a month. Some preprocessing steps are applied to the data to reduce the size of the clouds and use basic RANSAC-based plane segmentation to remove parts which we believe will not contain any objects. Using locations determined by one of several possible interest point selection methods, one of a number of possible types of features is extracted. Using a nearest neighbour approach, these features, are compared to a corresponding set extracted from a cloud representing a query object. Results from this comparison are used to vote for the position of the object in a 3D grid overlaid on the room cloud. We then cluster these votes and rank the clusters. The centroid of each of the clusters is used to extract a region from the room cloud which, in the ideal case, corresponds to the object that was passed to the system.

We perform an experimental evaluation of the system using various parameter settings in order to investigate computation time and other factors affecting the usability of the system.

Contents

Chapter 1

Introduction

Having a large amount of data is in most cases a good thing. Data, in an abstract sense, is the driving force behind the actions of every living thing, and as such holds great power. However, in order to make use of data, it is necessary to have some way of interacting with it in a useful way, and further processing it. While technologies for storing and interacting with data have been around for millennia, for the most part they were inconvenient and cumbersome. Writing allowed practically lossless transfer of information between generations, and is no doubt one of the most important inventions in the history of humanity. That being said, books are quite limited, especially when one wants to investigate a specific topic. Libraries are partial solutions to the problem, but most libraries don't possess all books in existence, and while indexes exist for a reason, it is still not easy to find what one is looking for.

With the internet and the immense amount of data available to its users, this problem of finding what one is looking for has been compounded, and good ways of getting around the problem have launched one of the most successful companies in history. At first, listing all of the early internet to create a database was a realistic proposition, and for some time this was a satisfactory solution. However, as the number of accessible data on the internet grew, the system became gradually more impractical. It could take minutes or even hours to get a result for a query, and the trawling of content caused network slowdowns [68, 73, 12]. Subsequent work in the area led to the development of search engines which were able to search for words in pages, and various innovations led this to become the very effective way of searching that we know today [13, 53].

While images have been on the internet since the early days, in recent years the advent of affordable digital cameras and the ubiquity of mobile phone cameras has led to hundreds of thousands of photographs being uploaded to the internet every minute [22, 32]. At its most basic, image search utilises the same techniques as text search, with information being extracted from metadata like tags, descriptions and keywords [34]. More recently, reverse image search has become more popular, allowing users to find similar images to an example by extracting information from

textures and trying to find other images which contain similar information [39]. There is still much information present in images that cannot currently be extracted and represented using image processing techniques, and this is an active research area.

An emerging method of data storage that will need to be searchable in the near future is 3D models and point clouds. 3D models have been used for a long time in computer games, medical imaging, and animation. More recently, developments in 3D printing have led to a growing number of websites which distribute models to use for printing [80]. For many years these sorts of models have been created using CAD programs, or in the case of object scanning, expensive time-of-flight cameras. The release of the Microsoft Kinect in late 2010 marked a turning point in the realm of 3D image processing, creating an affordable and effective method of gathering 3D data. Many research groups quickly purchased the hardware, and much work has been done in the area since. A 3D equivalent of the popular 2D image processing library OpenCV quickly came into existence for use with point clouds, as the data which comes from such 3D sensors is known [48, 55].

In this report, we will describe our approach the problem of retrieving from a data set objects that are similar to some object that we have provided, which we will call object query. In essence, we need to extract information from the data set and the objects that we are interested in which describes their properties in such a way that we can compare the descriptions to see if there are any similarities that imply the presence of an object in the data set. While the data set could be anything, in our particular case we have data from a project which studies long-term robot operation in office environments. The data set contains point clouds of a single office taken from the same position over a period of approximately a month. Some objects in the data set have also been labelled, so there is information about the positions of objects in the clouds.

While this project is not aiming to perform a specific task on an actual robotic system, within this context there are applications to which an object query system could be applied. Given a data set over long periods of time with clouds taken at various locations, the system could be used to track the motion of objects over time, and to provide information about where an object is likely to be at a certain time. One potential application is to help people find objects that have been misplaced.

The project will focus in particular on the implementation of a system which can perform object query. It will evaluate a number of standard methods for describing objects, and finding parts of objects that are particularly discriminative. While it is possible to describe objects as a whole, we will investigate the efficacy of using descriptions of small parts of the object to retrieve matches from the data set. Matches are found by comparing these descriptions, which are generally vectors of scalar values, to each other, and finding those which have the most similar values. This approach is called feature matching.

While the use of descriptors is the basis for the majority of systems for object retrieval, in most cases there are several additional layers applied on top of the basic feature-feature matching in the lower levels. This often includes costly pre-

segmentation of the input data, where it is necessary to determine what parts of the clouds are actually objects in order to create a description of them to use later. If the data is labelled in some way, this can be relatively easy to achieve, but we are interested in querying data which has no labels at all. As a result, we would like to investigate the effectiveness of using a very simple approach to the problem which does not required complex reasoning about the nature of objects.

In chapter 2, we explain some concepts that are important to understand the work, provide background information on relevant parts of the image processing literature, and attempt to introduce the reader to previous work in similar areas. The preprocessing steps that we apply to clouds are described in chapter 3. Brief descriptions of the interest points and descriptors that we use are given in chapters 4 and 5 along with some explanation as to why we wish to use these methods. Chapter 6 is the final chapter describing our system, wherein we discuss our approach to using descriptors to retrieve objects. Our experimental setup and the results of the experiments are described in chapter 7. We compare the quality of retrieval when different methods are used, and also investigate the time taken by the system under varying parameter settings. Finally, we summarise the system and our results in chapter 8.

Chapter 2

Background

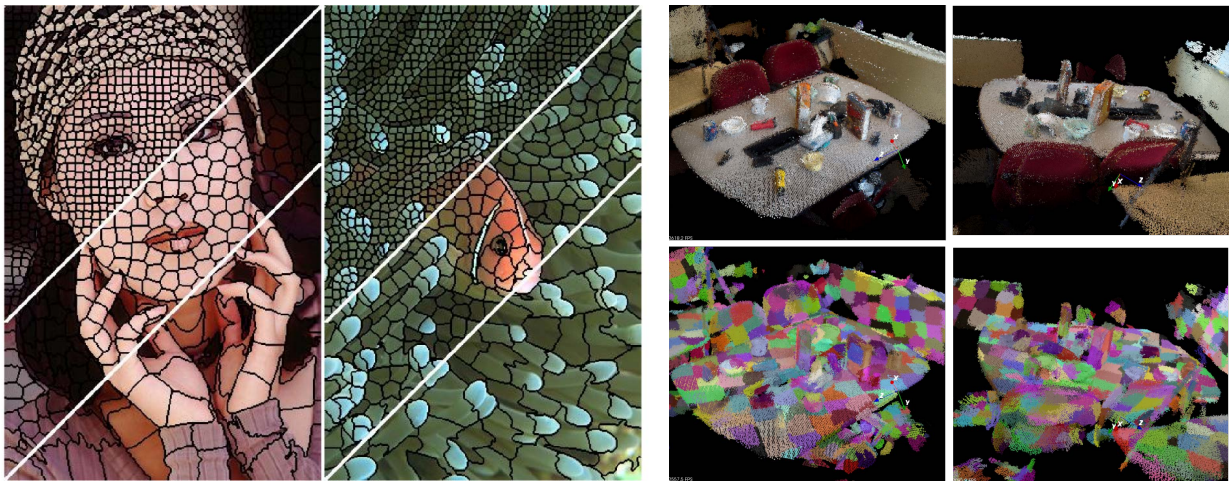
In this chapter we will introduce some key ideas relating to the project, and papers which are related to what we are interested in doing. While some of the techniques mentioned here are not directly used in the implementation of our system, they can be useful for context, or to give examples of different ways of approaching problems in this area. We discuss methods of finding interesting regions in image and point cloud data, and how these regions can be represented using descriptors, along with some methods for storing descriptors in ways that make it easy and efficient to find similarities.

2.1 Segmentation

Segmentation encompasses techniques for splitting an image or a point cloud into different parts, or grouping similar parts — this is essentially two sides of the same coin. In terms of images, segmentation might be used to try to find background and foreground pixels, or for point clouds, to separate objects from the surfaces on which they are resting. There are many different types of methods in the area, which approach the problem from different starting points.

Superpixel clustering is the most common technique used for segmenting images. The intent is to create regions in which all pixels have some sort of meaningful relationship. Graph based algorithms treat pixels as nodes in a graph, where the weights on edges between nodes are related to the similarity between the connected pixels — intensity, proximity and so on [1]. The most simple method is to use a threshold on the edge weights to create superpixels. Fulkerson et al. use superpixel methods to identify object classes in images [27]. An algorithm which applies the idea of superpixels to point clouds to create supervoxels (3D pixels) has also been developed [51].

Gradient ascent based algorithms iteratively improve clusters until some criterion for convergence is reached [1]. Popularised by Comaniciu [20], mean shift was first introduced by Fukunaga [26] in 1975, and rediscovered by Cheng [15] in 1995. The technique finds stationary points in a density estimate of the feature space, for



(a) Superpixels size 64, 256 and 1024 computed using SLIC [1] (b) Supervoxel oversegmentation [51]

Figure 2.1: Examples of 2D and 3D superpixel segmentations

example pixel RGB values, and uses those points to define regions in the space by allocating pixels to them. One common way of computing a density estimate is to place Gaussians at the location of each pixel, and then to sum the values of all the Gaussians over the entire space. Pixels which follow the gradient of the density to the same stationary point are part of the same segment. An example can be seen in Figure 2.2.

Random Sample Consensus (RANSAC) is a technique which uses shape models to find ideal models in noisy data. Points in the data set are randomly sampled, and used to construct a shape. For example, in the case of a line, two points are sampled, and define the line. Distances from points in the data set to the model defined by the randomly sampled points are then computed to find points which are inliers to the model. This number is stored, and the process repeated a number of times. At the end of the process, the model with the largest number of inliers is returned [23]. RANSAC can be applied to segmentation tasks by using it to find planes, cylinders, spheres and so on in point clouds. In the case of planes this is particularly useful, as they are usually not part of objects of interest, mostly making up walls, floors or surfaces on which interesting objects rest. By removing the points corresponding to these uninteresting surfaces, it should be possible to work only with parts of clouds that contain objects of interest.

Several extensions to RANSAC have been proposed. Maximum Likelihood Estimation Sample Consensus (MLESAC) chooses a solution that maximises the likelihood of the model instead of just the number of inliers [78]. M-estimator Sample Consensus (MSAC) uses a different cost function to the original implementation, additionally scoring the inliers depending on how well they fit the data [78]. The Progressive Sample Consensus (PROSAC) uses prior information about the like-

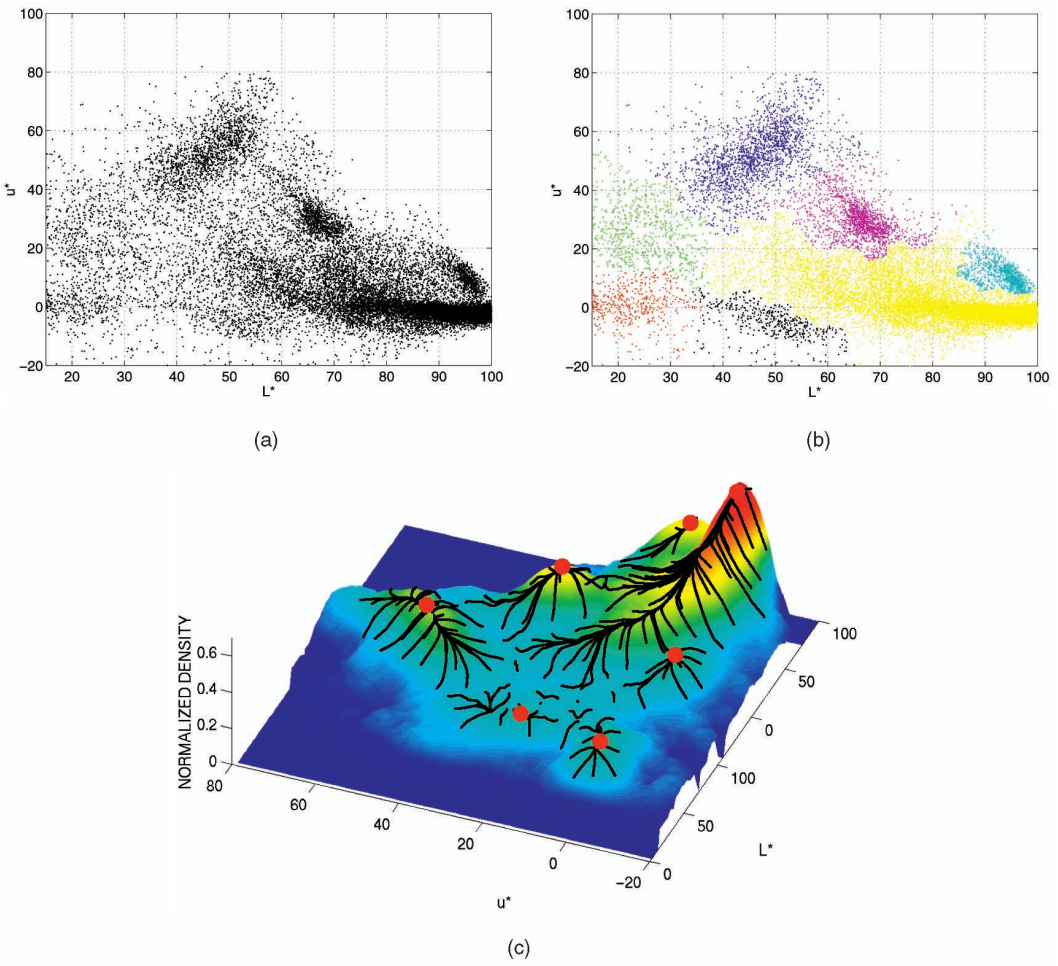


Figure 2.2: Visualisation of mean shift [20]. a) First two components of image pixels in LUV space. b) Decomposition found by running mean shift. c) Trajectories of mean shift over the density estimate.

lihood of input data being an inlier or an outlier to limit the sampling pool and greatly reduce computation cost [18].

2.2 Interest Points and Saliency

Interest points are an important concept in many image processing applications, and often form part of two stage process for extracting descriptor information from images or scenes. As the name suggests, techniques which use this approach try to find points in the image which are interesting, by some measure. This can be any of a number of things, depending on the types of images or objects that is being described. The general idea is that regions which have extreme values for something

are more likely to be picked up later when the same object is observed in another image. This is very important, as when computing descriptors one would like to extract them at the same points on the same objects every time in order to ensure that two instances of the object can be matched.

Sipiran and Bustos extend the popular Harris detector [31] to 3D [66]. The original detector for 2D finds edges and corners in images by computing a matrix of the sum of squared distances between points in one patch of an image, and points in a shifted copy of this patch. Interest points are then selected using the eigenvalues of these matrices. The extended 3D version uses normals to do the computation.

The SUSAN detector [69] uses sub-regions of circular masks placed on an image to define a value for the intensity variation in a region. This method is extended to 3D by combining normal direction variation with intensity variation and using a spherical mask.

A multi-scale signature defined by the heat diffusion properties of objects called the Heat Kernel Signature (HKS) [74] is used in [50] to retrieve shapes. The method is applied to meshes and is robust to deformations of the shape, which is particularly important for model matching.

Shilane and Funkhouser introduce a distinctiveness measure over classes of meshed objects which uses a database of existing objects to compute the distinctiveness of descriptors computed all over the object, based on how similar descriptors are to those inside the class compared to those from other classes. This distinctiveness measure is then used to discriminate between different classes of objects [64].

Zhong introduces an interest point selection method specifically for 3D, which uses the covariance matrix to define the region around a point, and extracts information about the region using the eigenvalues [84].

2.3 Descriptors

The problem of describing regions of an image in a compact and useful manner has been studied for a long time in the computer vision community. For any given point in an image, we would like to create a description which can be used to represent the region around the point in some way. This descriptor, or feature, can then be compared to other descriptors to see if there is some similarity. If the similarity is within a given threshold, then we can assume that the points represented by the two descriptors come from the same object, or represent the same thing in both images. Thus, it is important to create features which are distinct for different regions. In addition, since objects move around and can be seen from different sides, or in different lights, an attractive property of descriptors is to give similar results for the same region which has been transformed in some way. In practice, this is quite difficult to achieve.

2.3.1 2D

While 2D descriptors are not directly usable on point clouds, the ideas that they use to give effective results can be transferred over to use for 3D description.

The Laplacian of Gaussians was introduced by Lindeberg, and uses derivatives combined with some other techniques to select interest points. [40]. This paper also introduces the concept of automatic scale selection for feature detection, which has played an important part in the field since then. The scale of features can be investigated by blurring an image using a Gaussian kernel — higher standard deviation blurs the image more, resulting in the removal of small scale features.

Even today the Scale Invariant Feature Transform (SIFT) is among the most popular descriptors for 2D images. It is invariant to scale and rotation, and is robust to some variation in affine distortion, viewpoint and illumination, and is distinctive, allowing for correct matching of single features in large databases. There are several stages of computation. Extrema are found in different scales to find points invariant to scale and orientation. Keypoints are selected at the extrema based on their stability. Image gradients at the keypoint are used to define its orientation for future computations. The image gradients are then transformed into a local descriptor vector with length 128 [41].

Mikolajczyk and Schmid [43] introduce the Harris-Laplace detector which is an improvement on SIFT [41] and the Laplacian of Gaussians [40] in the sense that it is able to deal with affine transformations. They do not, however, introduce a new type of descriptor to go with the point selection.

Speeded-Up Robust Features (SURF) is a more recent descriptor which can be computed and compared much faster than most other descriptors. It makes use of integral images, which replace pixels in an image or image patch with a cumulative sum of the pixel intensities over the rows and columns. This allows for fast computation of pixel intensities in an area of the image. SURF takes some ideas from SIFT, using the spatial distribution of gradients as a descriptor, but integrates over the gradients instead of using individual values, which makes it more robust to noise. The resulting descriptor is a 64 element vector, which means that it is also faster to compare than SIFT [4].

2.3.2 3D

One early descriptor which remains popular is the spin image. The descriptor is generated from a mesh model at oriented points with a surface normal. A plane intersecting the normal with a certain width and height is rotated around the normal, forming a cylinder. The plane is separated into bins. The bins accumulate the number of points which pass through a certain bin during the rotation. The resulting 2D image is the descriptor. By varying the width of the plane the region which defines the descriptor can be modified. A small width will give a local descriptor, while a large width will give a descriptor for the whole image [36, 35]. Figure 2.3 shows a visualisation of how the image is generated.

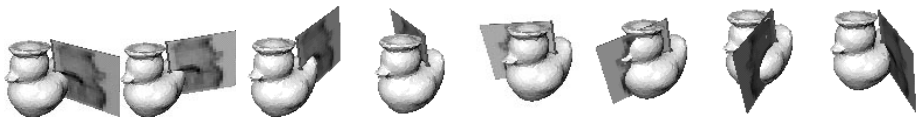


Figure 2.3: Frames from construction of a spin image [36]. The image plane spins around the oriented point normal and accumulates points.

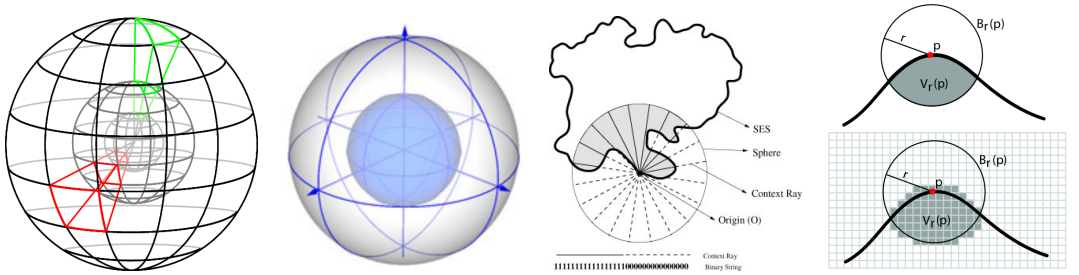
The Ensemble of Shape Functions (ESF) descriptor introduced in [81] by Wohlkinger and Vincze combines the Shape Distribution approach introduced by [49] along with some extensions proposed in [33]. It also makes use of their voxel-based distance measure from [82]. Pairs or triples of points are sampled from segmented partial clouds of objects, and histograms are created by extracting information such as distance, angle, ratios, and whether points are inside or outside (or a mix) of the model. See Figure 2.5.

The Point Feature Histogram (PFH) was introduced by Rusu et al. in [62]. It creates descriptors based on the angles between a point on a surface and k points close to it. The Fast Point Feature Histogram (FPFH) improved the speed of computation, and allowed the use of the descriptor in real time [58]. The Viewpoint Feature Histogram (VFH) extended the FPFH by adding viewpoint information to the histogram by computing statistics of surface normals relative to the viewpoint [60]. It also improved the speed of the FPFH. The clustered version (CVFH) further improved the viewpoint technique by mitigating the effect of missing parts and extending it to facilitate estimation of the rotation of objects [2].

Bo et al. develop the kernel descriptor initially created for RGB images for use on depth images and point clouds. The kernels are used to describe size, shape and edge features. Local features are combined to object-level features . Kernel descriptors avoid the need to quantise attributes. Similarity is instead defined by a match kernel [10], which improves recognition accuracy [9].

The point pair feature describes the relation between two oriented points on a model. This means that it does not depend so much on the quality and resolution of the model data. The model is described by grouping the point pair features of the model, providing a global distribution of all the features on the model surface [21].

3D Shape Context (3DSC) is an extension of the original Shape Context descriptor for 2D images [6]. A sphere is placed at a point, and its "top" is oriented to match the direction of the normal at the point. Bins are created within the sphere by equally spaced boundaries in the azimuth and elevation, and logarithmically spaced boundaries in the radial dimension (Figure 2.4a). The logarithmic spacing means that shape distortions far from the basis point have less effect on the descriptor. Each bin accumulates a weighted count based on the volume of the bin and local point density [25]. 3DSC does not compute a local reference frame — the vector of the azimuth is chosen randomly, and subdivisions computed from that.



(a) 3DSC [25] (b) SHOT [77] (c) Context Shape [63] (d) Integral Volume [29]

Figure 2.4: Visualisation of spherical descriptors.

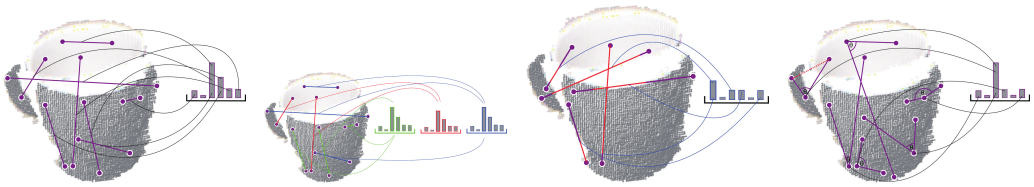
This means that a number of descriptors equal to the number of azimuth divisions need to be computed and stored in order to compensate, and the matching process is complicated as a result. The Unique Shape Context (USC) solves this problem by defining a local reference frame and using the directions of that reference frame to subdivide the sphere [76].

The Signature of Histograms of Orientations (SHOT) descriptor improves on 3DSC by taking inspiration from SIFT and making extensive use of histograms. The sphere is split into 32 volumes: 8 azimuth regions, 2 elevation and 2 radial (Figure 2.4b). A local histogram is computed in each of the regions, using the angle between the normal of points and the feature point. The local histograms are then combined to form the final descriptor [77]. The authors also extend the descriptor to include colour (COLORSHOT) [75].

The Rotation Invariant Feature Transform (RIFT) is a generalisation of SIFT. Using intensity values computed at each point from the RGB values, a gradient is computed. Concentric rings are placed around the initial point, and a histogram of the gradient orientations is created for points within each ring. The orientation of the gradient is computed relative to the line from the central point so that the descriptor is rotation invariant. The descriptor is 2D — one dimension is the distance, the other the gradient angle. The distance between two descriptors is measured using the earth mover's distance (EMD), which is a measure of the distance between two probability distributions [38].

Multi-scale descriptors are useful as they can be used to characterise regions of varying size. Cipriano et al. introduce such a descriptor for use on meshes [19]. It captures the statistics of the shape of the neighbourhood of a vertex by fitting a quadratic surface to it. Vertices in the region are weighted based on distance from an initial vertex, and a plane is constructed using a weighted average of the face normals. The parameters of the quadratic are then used to find its principle curvatures, which make up the descriptor.

Work in protein-protein docking also uses 3D descriptors to help with simulations of an otherwise lengthy and complex process. The Surface Histogram is introduced by Gu et al. [30], and uses the local geometry around two points with



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

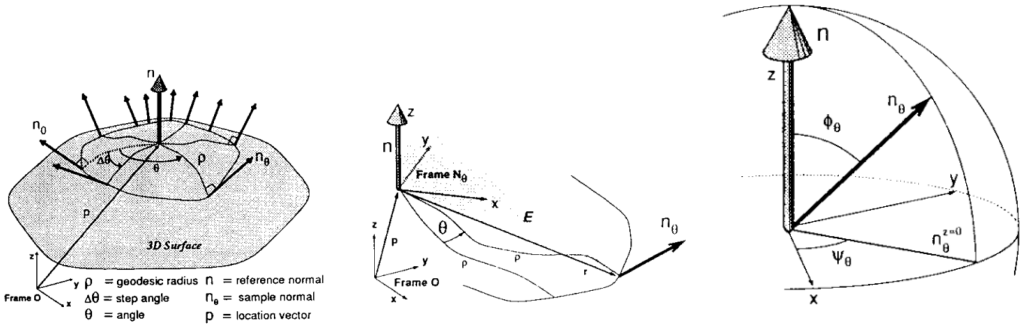
Figure 2.5: Examples of the measures used to construct the Ensemble of Shape Functions histograms of [81]. a) Distance between points. b) Whether the points are on or off the model, or mixed. c) Ratio of line segments on and off the surface of the model. d) Angle between pairs of lines.

specific normals on the surface of a protein. A coordinate system is defined by the two points and the line between them, and a rectangular voxel grid is defined around the points. The grid is then marked in locations where the surface crosses the grid, and a 2D image is constructed by squashing the data onto one of the axes. The descriptor is designed to immediately give a potential pose for the docking.

Another example of a shape descriptor from biology is the Context Shape [63]. A sphere is centred on a point, and rays are projected from this point to points evenly distributed on the surface of the sphere (Figure 2.4c). Each of the rays is divided into segments, with a binary value associated with each segment depending on whether the segment is inside or outside the protein. To compare the descriptor, a rotation is applied to match the rays, and a volume of overlap is computed based on matching bits in the rays.

The splash descriptor was introduced by Stein et al. [72]. A point on the surface with a given surface normal (the reference normal) is chosen, and a slice around that with some geodesic radius (distance along the surface) is computed. Points on the circle are selected using some angle step, and the normal at that point is determined. A super splash is when this process is repeated for several different radii. For each normal on the circle, additional angles between it and a coordinate system centred on the reference normal are computed. These angles and the angle around the circle are then mapped into a 3D space, where polygonal approximation is made, connecting each point with a straight line. Some additional computation is done to allow the encoded polygons to act as a hash. Figure 2.6 shows part of the formulation.

Point Signatures are similar to the splash descriptor in the sense that they both sample points on a circle [16]. This descriptor again selects a reference normal, and has a specific radius. This time, the radius defines a sphere around the point. The intersection of the surface with the sphere is a 3D space curve. The orientation of the curve is defined by fitting a plane to it. The distances between the space curve and the fitted plane at sampled points define the signature of the reference point. These signatures can be compared by lining them up and checking whether



(a) (b) (c)

Figure 2.6: Splash descriptor [72]. a) shows the splash and normals around it. b) and c) show how the additional angles are defined.

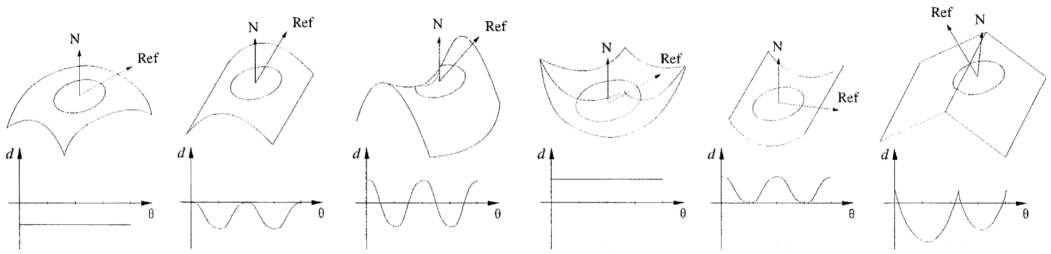


Figure 2.7: Examples of the point signature responses to different surfaces [16]. d is the distance from the reference vector to the space curve defined by the intersection of the surface with a sphere centred at N. Ref rotates about N.

the query falls within the tolerance band of previous signatures. Figure 2.7 shows signatures from various surfaces.

2.3.2.1 Descriptors With Interest Point Extraction

While many descriptors designed for 2D applications also select interest points during an initial step in the process, the 3D descriptors that we have mentioned above do not automatically find locations in the cloud which are good points at which to compute descriptors.

The Normal Aligned Radial Feature (NARF) is an interest point extraction method with a feature descriptor. A score for the image points is determined based on the surface changes at the point, and information about borders. An interest value is computed from this based on the score of the surrounding points. Smoothing is applied, and non-maximum suppression is applied to find the final interest points. To compute the descriptor, rays are projected over the range image from the centre at certain intervals. The intensities of cells lying under the ray are weighted based

on their distance from the centre, and a normalised weighted average of the pairwise difference of cells is used to define each element of the descriptor vector, which has a length equal to the number of rays [70]. The method is an improvement on a previous paper by the authors [71]. A problem with this method is that it uses range images directly. Point clouds can be used to generate range images by looking at them from different viewpoints, but this adds complexity to the method.

The integral volume descriptor is interesting as it combines interest point selection and description into one. The descriptor is defined as the volume of the intersection of a sphere centred at a point on the surface of an object with the inside of the object (Figure 2.4d). Interest points are selected by histogramming the descriptor values, identifying bins with a number of points less than a specified values, and selecting points from these bins. To ensure features are properly spaced, points in a certain radius of already selected points cannot be used. By modifying the radius of the sphere used to generate the descriptor, interest points at different scales can be selected [29].

2.4 Storing and Querying Descriptors

There are several techniques for storing and querying descriptors, mostly based on some form of tree. Recently, the k-d tree[8, 24] has been used for efficient approximate matching with either an error bound [3], where there is a bound placed on the error between the true nearest neighbour and the one found, or a time bound [5], where the search is stopped after examining a certain number of leaf nodes. Further improvements on the k-d tree are introduced in [65], where multiple randomised trees are used to optimise the search. A priority search tree algorithm is introduced in [45], which appears to be very effective. This may be the same one as in [44]. The algorithm in the last two papers has been integrated into PCL, which is useful.

A different approach to nearest neighbour search is the balltree, which uses hyperspheres in a hierarchy to enclose points in the space [47]. Unlike the k-d tree, regions on the same level of the tree are allowed to intersect, and do not need to partition the whole space, which gives the balltree its representative power.

The vocabulary tree [46] makes use of techniques from document search to index images. Using k-means clustering, construction stage creates a hierarchical quantisation of the image patch descriptors. In the query phase, descriptors are compared to the cluster centres, and go down the tree until a leaf is reached. The path through the tree is used as a scoring measure to present retrieval results.

Philbin et al. [52] show that flat (single-level) k-means clustering can be scaled to large vocabulary sizes if approximate nearest neighbour methods are used. Early systems for image retrieval used a flat clustering scheme, which could not scale to large vocabularies [67]. The paper also introduces a re-ranking method which uses spatial correspondences, which improves the retrieval quality.

Boiman et al. [11] introduce the Naive Bayes Nearest Neighbour (NBNN) classifier. It uses nearest neighbour distances in the space of descriptors instead of images, computing "image-to-class" distances without quantising the descriptors. In general, quantisation allows for dimensionality reduction, at the expense of the discriminative power of descriptors. NBNN "can exploit the discriminative power of both (few) high and (many) low informative descriptors". The problem here is that the classes must be known beforehand, and in our case we do not have that information. The local NBNN [42] does not do the search based on classes. Instead, all the descriptors are merged into a k-d tree on which approximate k-NN is run to find descriptors in the local region of a query descriptor. A distance to classes not present in the k-NN region is approximated by the distance to the $k+1$th neighbour.

Funkhouser and Shilane present a method for querying a database of 3D objects represented by local shape features [28]. Partial matches (correspondences) are stored in a priority queue sorted by geometric deformation and the feature similarity. This means that only objects in the database with a high probability of being a match need to be processed.

Some work has been done on optimising the retrieval of relevant images by learning from user input [56]. When retrieved images are presented, the user ranks them in terms of relevance, and this rank is then used to improve the relevance of future searches.

Chapter 3

Preprocessing

The first step in the object query system is to perform some preprocessing on the clouds in the data set — while not strictly necessary, there are some benefits to doing so, chief of which is a reduction in computation time. The data set that we have consists of around 80 clouds of a single room, taken at different times during different days of approximately a month of time. The clouds are made up of a number of intermediate frames, which are registered into a complete cloud. The robot used to collect the clouds takes several sweeps of the room, changing the angle of the camera after each sweep. The clouds are constructed using frames taken from a sweep where the camera is pointing slightly below the horizontal. Examples of the raw clouds can be seen in Figure 3.1.

Figure 3.1: Sample raw cloud viewed from several angles

3.1 Downsampling

In their merged forms, the clouds on average contain approximately 4,300,000 points for a room which is around 4m wide, 5.5m deep and 3m high. This number of points does not actually provide us with much additional information, since the intermediate frames all have the same resolution. As such, we can safely downsample the cloud to get a more reasonable number of points.

To downsample, we make use of a voxel grid, which splits the 3D space in which the cloud sits into smaller subspaces of equal size called voxels. The width, height and depth of voxels in the space can be specified, but we are interested in keeping all dimensions the same resolution, and so we specify the parameters so that each voxel is a cube. At this stage, we would like to perform a simple downsampling to reduce the number of points, but we wish to keep small details in the cloud — something in the realm of a 1cm resolution is ideal in this case.

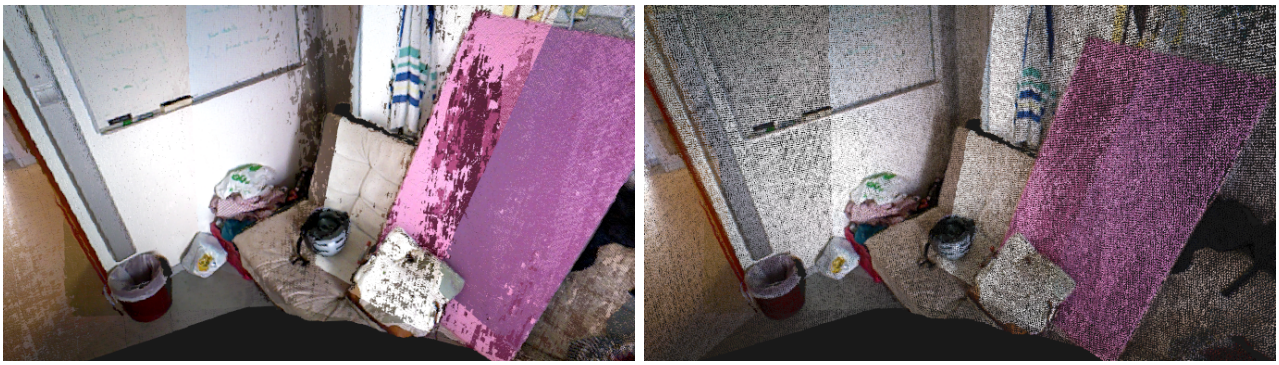
Downsampling with a 1cm resolution gives a reduction in size of the clouds of on average 78%, to approximately 950,000 points. Figure 3.2 shows the effect of the downsampling. While there is slight degradation of the textures, this is to some extent a visual effect which is viewpoint dependent. Most of the structure in the cloud is retained, which is key. This step is important, as it greatly affects the speed of computation of subsequent steps in the system, but it is a trade off. If the downsampling resolution is too low, then we lose a lot of information about the surface structure of parts of the cloud, and this is likely to lead to worse performance when trying to find matches. How tolerant we are to low resolution also depends on the kinds of objects that we are interested in finding. If we do not care about smaller objects, then even with a lower resolution the results should still be satisfactory. However, a lower resolution likely means that it will be necessary to look at larger regions of space in order to describe points. We will investigate the effects of this in chapter 7.

Show some of the objects which are made up of slices due to viewing angle - downsampling can reduce this slicing effect.

3.2 Transformation and Trimming

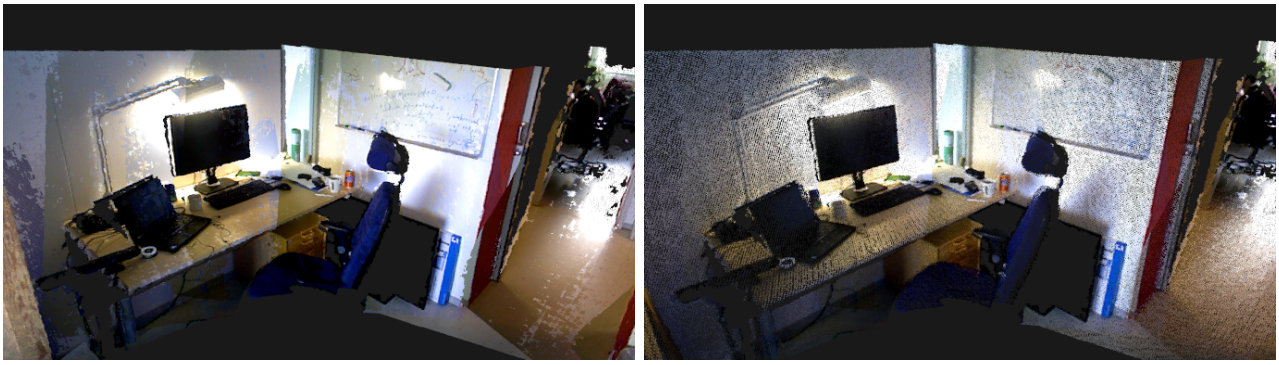
Once the cloud has been downsampled, there is a little more that needs to be done in order to get the cloud into a convenient form. The raw data that we have has clouds which have their origin at the position of the camera while the room was being scanned. Our data is a subset of a larger dataset which contains clouds of more than one room — if we were to use the data without applying any additional transformations, all the clouds would sit on top of each other at the origin, whereas we would ideally like to have them in their true position relative to the origin. The robot collecting data knows its position, so this information is stored.

As mentioned before, each cloud is a combination of a number of intermediate frames, each of which has corresponding information about the pose of the camera



(a) Sofa

(b) Desk 1



(c) Desk 2 side

(d) Desk 2 front

Figure 3.2: The effect of downsampling. The left column shows the original clouds, the right column clouds downsampled with voxel size of 1cm^3.

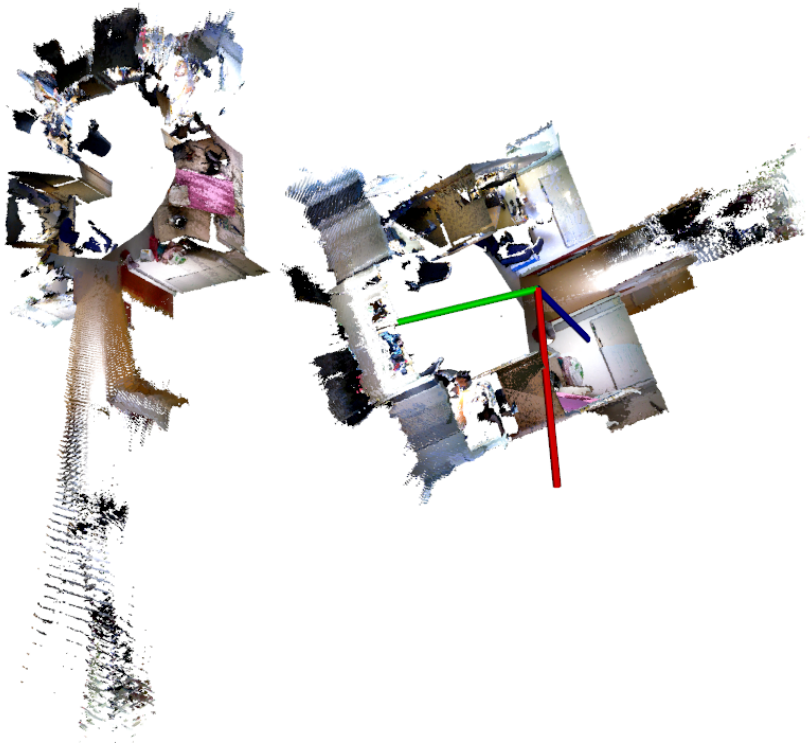


Figure 3.3: Original cloud and the transformed cloud. The original cloud is on the right. The coordinate axis shows the global reference frame — none of the axes are aligned for the original cloud, but the transformed cloud is well aligned with the x-y plane.

when the frame was taken, which we can use to transform the complete cloud into its actual position in space.

An added benefit of this transformation is that it allows us to remove the floor and ceiling by using a simple thresholding filter on the z axis, as the floor of the cloud is now aligned with the x-y plane of the global reference frame, as opposed to being aligned with the cloud's rotated reference frame (Figure 3.3) The threshold for the ceiling can be determined by measuring the ceiling height, and the floor is assumed to be a $z = 0$. We add a small offset to each of the values to ensure that the parts are correctly removed even if there is some noise.

Although we would like the system to be as generic as possible, the particular subset of clouds that we are using have a large number of points outside the room which do not give any useful information. To this end, we also include additional filters on the x axis to remove these points. Figure 3.4 shows the end result of this step.

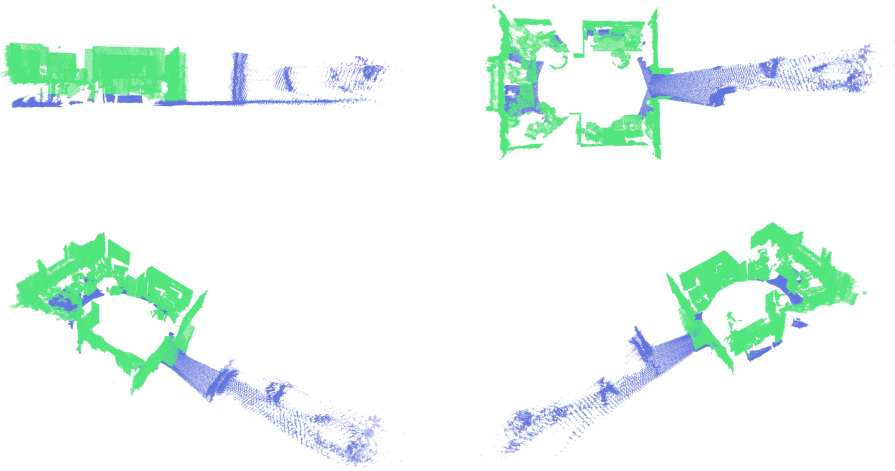


Figure 3.4: Result of trimming step. Transformed cloud is blue, trimmed is green.

Figure 3.5: Example of the result of plane extraction.

3.3 Plane Extraction

Having extracted normals from the cloud, we come to what is the most costly preprocessing step. Due to the structured nature of our dataset, the number of planes present in the clouds is quite high. While the presence of planes can be used to define surfaces and the like, in our system we are not interested in using the planes for anything in particular, and as such removing them from the cloud is good, because we remove a large portion of points in the clouds which are not be parts of any object, speeding up computation time of subsequent steps. An example of the result of this step can be seen in Figure 3.5.

Plane extraction is done by running RANSAC multiple times with a plane model. A plane can be described by its general form equation

$$ax + by + cz + d = 0 \tag{3.1}$$

where the normal vector $\mathbf{n}$ is defined by the coefficients a, b and c. To get the model coefficients, RANSAC samples three points (p_1, p_2 and p_3) from the input cloud. From these three points, the normal is computed using the cross product [79]

$$\mathbf{n} = (p_2 - p_1) \times (p_3 - p_1) \tag{3.2}$$

Once the plane coefficients have been computed, we must find the inlier points of this plane model, based on their distance to the plane. The perpendicular distance

Figure 3.6: RANSAC with basic plane model and with plane-normal model.

of a point p to a plane is

$$D = \frac{\mathbf{n} \cdot p + d}{\mid \mathbf{n} \mid} \tag{3.3}$$

A point is considered to be an inlier if $D < D_t$, where D_t is some threshold on the distance. The RANSAC algorithm repeats the point sampling n times, storing the plane coefficients and number of inliers. At the end of the process, the best plane the one with the largest number of inliers.

While this simple formulation can work well, there can be issues where the planes that are extracted are not actually planes, due to there being regions in the cloud where there can be a large number of inliers, but no actual plane, as seen in Figure 3.6. This effect can be mitigated by including a single additional step to the inlier check, which also looks at the angle between the plane normal and the normal at the point, computed by

$$\theta = \cos^{-1}(\mathbf{n} \cdot p) \tag{3.4}$$

A point is then considered an inlier only if it passes the distance threshold check and $\theta < \theta_t$, where θ_t is the threshold on the angle. This simple addition gives much more consistent results.

The RANSAC implementation that we are working with uses only a single distance computation

$$D_a = w\theta + (1 - w)D \tag{3.5}$$

$$w = (1 - p_c)w_n \tag{3.6}$$

where p_c is the curvature at the point p, and w_n is a predefined weight on the distance between the point and plane normals. $p_c \to 0$ on flat surfaces, so in these regions the normal will have a higher influence on the aggregate distance D_a, whereas in regions of high curvature the euclidean distance will be more important. Inliers are points where $D_a < T$.

When extracting planes, we use several parameters in addition to the aggregate distance threshold T to tweak the behaviour. The main aim of the additional parameters is to prevent planes which are too small from being extracted. We set a hard limit on the total number of planes which can be extracted, and also define a threshold on the minimum number of points N_{min} in a plane.

$$N_{\text{min}} = \max(\eta N_{\text{trim}},\, N_{\text{fixed}}) \tag{3.7}$$

where η is a small positive positive value. Since we are dealing with large clouds, a suitable range of values is $[0.02, 0.05]$. N_{trim} is the number of points in the trimmed cloud. N_{fixed} is a fixed value. We choose the maximum of the two values to ensure that fluctuations in the cloud size are compensated for.

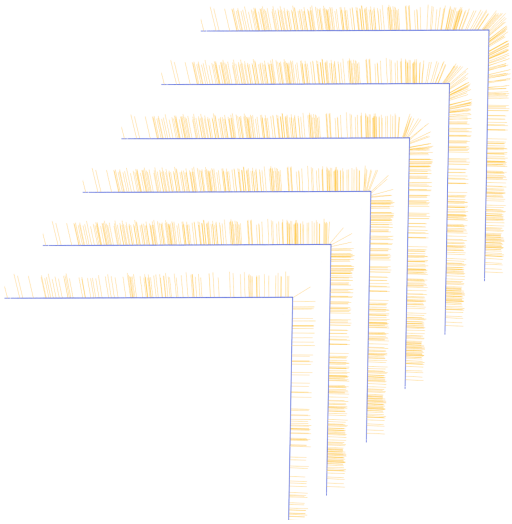


Figure 3.7: Example of the smoothing effect of normal estimation radius. From bottom to top, 0.01, 0.025, 0.5, 0.2, 0.25, 0.5cm radius. Normals are indicated by orange lines. Note the tendency of normals with higher radius to tilt as they approach the corner. Normals on the top section are slightly skewed due to perspective.

3.4 Normal Estimation

In this step, normals are estimated for each point in the cloud. The normal at a point is the vector which is perpendicular to the curvature of the surface at that point. By estimating normals for clouds, we can get some more information about the surface structure of the cloud. Normals are used in several parts of the system, including by feature selection methods and features. As mentioned above, they are also used in the plane extraction step to increase accuracy.

There are many ways of estimating normals, but the method we use is formulated as a least squares plane fitting problem, which is used to estimate the normal of the plane tangent to the surface at the point at which the normal is to be computed [57]. The computation gives an ambiguous result in terms of the sign of the normal. To correct for this, a viewpoint is needed, which serves to define what sign is used. Perhaps the most important thing to note is that the normal must be computed using points in a neighbourhood; either within a certain radius, or the nearest k points. The neighbourhood determines the scale factor that results. A small neighbourhood gives a small scale factor, and a large neighbourhood a large scale factor. A large scale factor can be bad if the objects that one is trying to examine have regions where the rate of change of surface curvature is high, such as at the corners of tables. It results in the smearing of edges and the suppression of fine detail [57]. Figure 3.7 shows an example of the effect of different neighbourhood sizes on the results.

Include figure with annotation objects with different normal radius

Figure 3.8: Planes extracted with different settings for the normal radius

During preprocessing we compute two different sets of normals using different settings for the radius. One set is for use with plane extraction, which has a higher value for the radius, somewhat mitigating the effect of noise on the normals, and resulting in less patchy extraction of planes (Figure 3.8).

Chapter 4

Interest Point Selection

Once the preprocessing step has been completed, we can move on to computing features from the processed clouds. First, however, we need to choose the points at which the descriptors will be computed. The idea of interest point selection is to choose points in the cloud which are better in some way than other points for feature extraction — which points these are depends on the method used. This is important, as if we can compute descriptors at locations which are unique to an object, it makes any correspondences that are found when comparing the descriptors likely to be actual occurrences of the object in the clouds in which we are searching.

In this section we will describe in some detail the methods that we use. All of the methods used are part of the PCL keypoints library [54].

4.1 Uniform

The first and most obvious method of selecting points for feature extraction is not to try to select interesting points at all, but to simply spread points uniformly over the space. With this method, one would expect to extract a larger number of points than with targeted methods, and since the entire space is covered, it is unlikely that there will be any omissions of points that are interesting.

The problem with having a large number of points is that this results in more features having to be computed and compared in later stages.

To compute the uniform points, we simply downsample the cloud once more. The size of the voxels used determines the spread of the points over the space — the behaviour of this method is determined entirely by a single parameter.

4.2 ISS

Zhong [84] introduces the intrinsic shape signature interest point selection method as one of a series of steps in the computation of the ISS descriptor introduced in the same paper.

The main component of this method is the scatter matrix, which is the covariance matrix of points within a spherical region around a sample point. For a point p_i, the 3×3 scatter matrix is

$$S(p_i) = \sum_{|p_j - p_i| < r_s} (p_j - p_i)(p_j - p_i)^T \tag{4.1}$$

where p_j is another point in the cloud. r_s defines the saliency radius, which limits the points which we consider to be in the neighbourhood of p_i. Interest points are only extracted in regions where there are at least n_{min} points in the neighbourhood of p_i.

Once S is computed, its eigenvalues λ_i^1, λ_i^2 and λ_i^3 (with decreasing magnitude) are extracted. The smallest eigenvalue λ_i^3 can be used to measure the 3D point variations in the neighbourhood of the point [84]. If it happens that two of the eigenvalues computed are equal, the reference frame of the point can become ambiguous, so limits are applied to the ratio of the eigenvalues such that

$$\frac{\lambda_i^2}{\lambda_i^1} < \gamma_{21}, \; \frac{\lambda_i^3}{\lambda_i^2} < \gamma_{32} \tag{4.2}$$

With this formulation, it is likely that more points are considered interest points than are not. To thin the interest points further, non-maximum suppression is used. Essentially, this removes from the interest points any point where the value of λ_i^3 at the point is not the maximum in the neighbourhood of the point. This neighbourhood is defined by the radius r_n, whose value is usually distinct from the value of r_s.

4.3 SUSAN

The SUSAN (Smallest Univalue Segment Assimilating Nucleus) detector is based on an algorithm introduced for 2D feature detection by Smith [69]. We use an extension of this detector to 3D. The basis for the SUSAN principle comes from the concept of each image point having an associated local area which has intensity and normal direction values that are similar to it.

A spherical region called the mask is defined, with some radius r_m which has at its centre a point referred to as the nucleus. Looking at the points within the spherical region, we compare their values of the normal direction and intensity to the nucleus values. From this comparison, a region of space which has similar values to the nucleus can be defined. This region is known as the univalue segment assimilating nucleus or USAN. Figure 4.1 shows this principle in the 2D case. The USAN contains information about the structure of the cloud in small region. Depending on the position of the nucleus in the cloud, the volume of the USAN will vary. In regions where all points are similar, the USAN is large, and it is small when the region has a large variation in point intensity and normal direction. Based on this

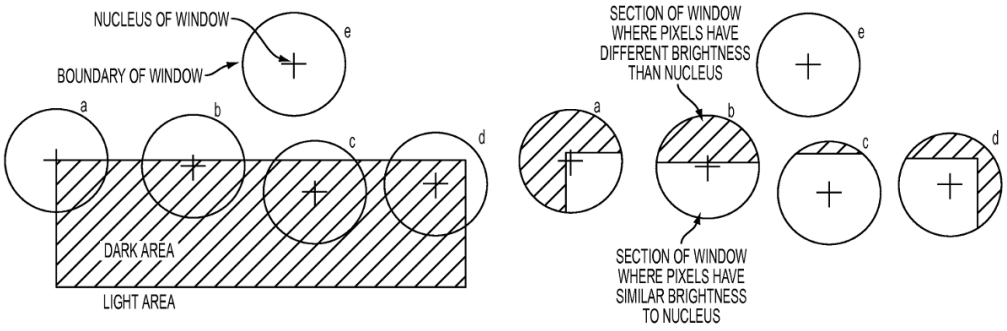


Figure 4.1: Concept of nucleus and mask in 2D SUSAN detector. USAN is the white region in the right image [83].

observation, using the inverted USAN volume as a feature detector should result in the selection of descriptive points — hence the name *Smallest* USAN.

To compute SUSAN keypoints, the following process is applied to each point p_i in the cloud. First, all points p_j in the neighbourhood defined by r_m are found. We then define the USAN and the centroid of the mask. In order to be considered as part of the USAN, a point must fulfill the inequalities

$$|I_i - I_j| \leq I_t \tag{4.3}$$

$$1 - \mathbf{n}_i \cdot \mathbf{n}_j \leq \theta_t \tag{4.4}$$

where I is the intensity of a point, and $\mathbf{n}$ is the normal, and I_t and θ_t are user-defined thresholds on the intensity and angular difference. The intensity is computed from RGB values using

$$I = \frac{r + g + b}{3} \tag{4.5}$$

We assume that each channel of the RGB value of a point has the same weight.

The centroid C is computed using

$$C = \frac{1}{|\text{USAN}|} \sum_{p_j \in \text{USAN}} p_j \tag{4.6}$$

The last thing to do for each point is to ensure that the number of points in the USAN is within the bound

$$0 < |USAN| < 0.5(N - 1) \tag{4.7}$$

where N is the number of points in the neighbourhood of the nucleus. If this check is successful, the output intensity of the nucleus is set to $I_o = 0.5(N-1) - |USAN|$. This defines the response of the feature selection at this nucleus.

Once I_o has been computed for all valid points in the cloud, non-maximum suppression is applied. Only those points which have the minimal intensity in the neighbourhood defined by r_m are used as the final interest points.

4.4 SIFT

The scale invariant feature transform, introduced by Lowe [41] in 1999, it is still commonly used for many 2D image processing applications. The most important concept introduced in the paper is that of scale invariance — the SIFT feature extraction method automatically selects features at different scales. This effect is achieved by applying a Gaussian blur with different standard deviation in 2D, and by downsampling clouds in 3D. The leaf size used for downsampling is determined by the number of *octaves* N_o. Within each octave, there are several *scales* N_s which are applied. After all scales in an octave have been computed, the leaf size is doubled, and the process repeated, until it has been applied to all octaves.

For each octave, the procedure begins with downsampling the point cloud to the scale defined for that octave S_o, which initially is set to the minimum scale $S_{\min}$. The scales in the octave are then defined by

$$s_i = \frac{S_o \cdot 2^{i-1}}{N_s} \tag{4.8}$$

where $0 < i < N_s$. Each point p in the cloud has its nearest neighbours P_{nn} computed, within a radius three times the maximum scale s_{N_s} in that octave. The same neighbours are used for computation of the difference of Gaussians D for each scale in the octave. In each scale, a Gaussian response R is computed

$$R_i = \frac{\sum_{q \in P_{nn}} 0.299 q_r + 0.587 q_g + 0.114 q_b \exp\left\{-\frac{0.5}{\sigma} \mid\mid p - q \mid\mid\right\}}{\sum_{q \in P_{nn}} \exp\left\{-\frac{0.5}{\sigma} \mid\mid p - q \mid\mid\right\}} \tag{4.9}$$

q_r, q_g and q_b are the red, blue and green channels of the colour at the point q, $\sigma = s_i^2$. The difference of Gaussians is then

$$\mathrm{DoG}_i = R_i - R_{i-1} \tag{4.10}$$

where $1 < i < N_s$. These values are then used to find extrema in the scale space. The neighbourhood of each point is examined again, and the maximum and minimum values of the DoG at any point within the neighbourhood are found for each scale. For consideration as an interest point, the value of DoG_i must be greater than a minimum contrast threshold t_c. This limits the inclusion of points where the responses are not very different. If this threshold is exceeded, the value of DoG at the point is checked to see if it is a maximum or a minimum in its neighbourhood, and is either larger or smaller than the maximum and minimum values for the same point in the neighbouring scale spaces. If all these criteria are fulfilled, the point is added to the interest points.

Once this process is completed for a single octave, the scale S_o is doubled, and the process repeats N_o times. The selected interest points are the aggregated results from each individual octave.

4.5 Harris

Like SIFT, the Harris detector, introduced by the eponymous Harris [31], was originally used as a method for edge and corner detection in images. Much like ISS, it uses a covariance matrix applied to the neighbourhood of a point as the basis of its function. Rather than using the points themselves, however, the Harris detector finds the covariance matrix of the normals in the neighbourhood. The response at the point is then computed by combining the determinant and trace of the matrix. The resulting responses for each point are then thinned using non-maximum suppression, and the remaining points are the interest points.

Chapter 5

Descriptor Extraction

In this step, we compute descriptors at each of the locations that was selected by the interest point method that was used. The end goal of the combination of the interest point selection and descriptor extraction steps is to produce a set of descriptors that can be used to represent the scene without having to keep all of the original data. In addition, putting information about the scene into a compact representation also allows the comparison of two scenes much more quickly than would otherwise be possible. Instead of comparing complex structures in the scene, simple vectorial representations are compared instead. The intention is to compute these representations and have them stored, so that they can be accessed later to compare to the descriptors extracted from query objects. Correspondences between descriptors from the query object and parts of the room clouds should indicate the presence of the object in that cloud. We will try to use this to more reliably retrieve objects.

However, finding a compact representation that is also distinctive enough and produces similar results for similar regions of space is not easy. In the image processing literature, a lot of work has been done to develop novel descriptors which are faster to compute, are invariant to more effects that might reduce their effectiveness, and better represent the image data. As with interest point selection methods, current methods in 3D often make use of the lessons learned from the development of 2D methods.

5.1 SHOT

The signature of histograms of orientations descriptor is the product of a study on 2D descriptors, particularly SIFT, and makes extensive use of histograms, as the authors believe that this is part of the reason why the descriptor is so effective [77]. The paper also discusses the importance of the local reference frame, or RF. Defining a local RF is a way to ensure that the same descriptor will be computed if the same points are translated or rotated, or if there is noise or clutter in the region the descriptor algorithm uses to define the descriptor values. This is analogous to

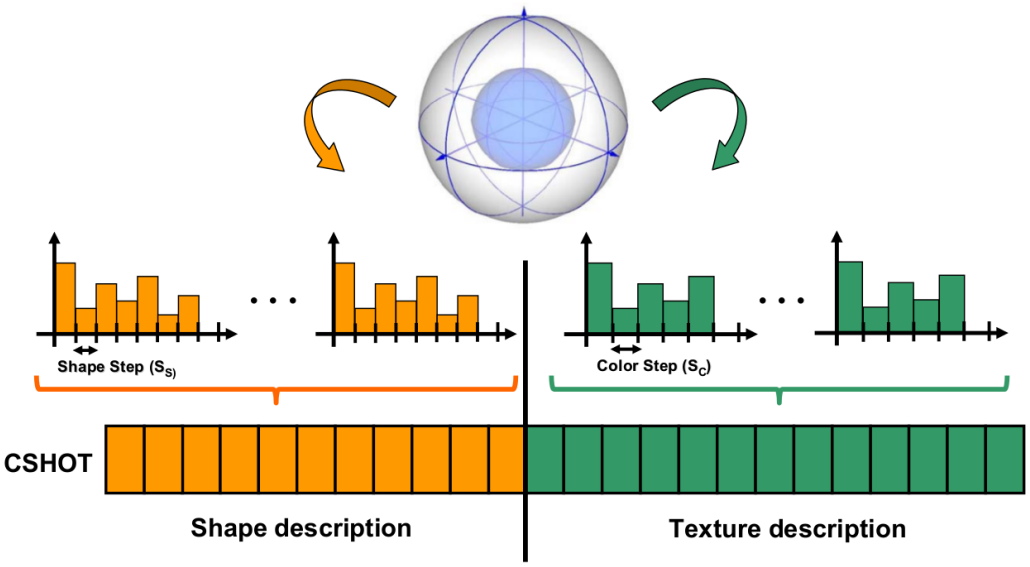


Figure 5.1: Representation of the construction of the SHOTCOLOR descriptor and the subdivisions used for local histogram computation.

the problem of rotation and scale invariance in 2D descriptors. The RF is computed based on the eigenvalue decomposition of a special scatter matrix M

$$M = \frac{1}{\sum_{i:d_i \leq R}(R - d_i)} \sum_{i:d_i \leq R} (R - d_i)(\mathbf{p}_i - \mathbf{p})(\mathbf{p}_i - \mathbf{p})^T \tag{5.1}$$

where R is the radius used to define the neighbourhood used to compute the descriptor, $\mathbf{p}$ is the point at which the feature is to be computed, $\mathbf{p}_i$ is another point in the cloud, and d_i is the euclidean distance between $\mathbf{p}_i$ and $\mathbf{p}$.

The addition made by the authors to previous work is to use a weighted linear combination, where the weight of a point is lower the further it is from the central point. This is in order to reduce the effect of clutter. The eigenvalue decomposition alone is not sufficient to define an unambiguous RF. To do so, the authors use a technique introduced in [14], which orients the signs of the eigenvectors to make them coherent with those of the points that they are representing.

Having defined the local RF, information about the location of points within R is accumulated to create the descriptor. This is done by computing local histograms within subdivisions of the spherical region, and then grouping them together to form the final descriptor. The spherical region is split into 32 regions by splitting the sphere along the radial, azimuth and elevation axes, as seen in Figure 5.1. Points in each subdivision are grouped into bins of the local histogram according to the cosine of the angle between their normals and the normal of the feature point $\mathbf{p}$. This formulation reduces computation time, and does not require complicated binning. Using the actual angle to allocate points to bins has the disadvantage of needing

different bin resolutions depending on whether directions are close to or orthogonal to the normal direction [77].

5.2 SHOTCOLOR

In a subsequent paper, the authors of SHOT introduce an extension to the descriptor which also includes colour information [75]. To include colour information, the same histogramming method is applied, but instead of using the dot product of the RGB vectors at the feature point and a neighbourhood point, the authors use the L_1 norm of the RGB value converted to the CIELab colour space. The value a pair of points is therefore

$$l(C_p, C_q) = \sum_{i=1}^{3} \mid C_p(i) - C_q(i) \mid \tag{5.2}$$

where C is the CIELab colour triple of a point.

5.3 USC

Another descriptor introduced by Tombari is the universal shape context [76]. This descriptor is an improvement of the 3D shape context, in the sense that it has a unique reference frame, constructed in the same way as for SHOT. As with SHOT, a spherical region is defined by a radius R_s and a central point, and separated into a number of bins. For each point, the number of points N in a local region defined by radius R_d is computed. A weighting w for this point is then computed according to

$$w = \frac{1}{N} V_b \tag{5.3}$$

where V_b is the volume of the bin that the point falls within. For each point, w is accumulated in the descriptor vector according to the bin into which it falls. The final descriptor is therefore a representation of the combined weights of all of the points in the sphere.

5.4 PFH

The point feature histogram is the first in a family of descriptors described by Rusu et al. [59]. The descriptor was developed in order to improve performance of cloud matching with iterative closest point (ICP). Most of the descriptors used for that purpose previously had only a single dimension, and as a result could not be properly descriptive over a whole scene. The descriptor is likely to have a similar value at different points in the scene, which causes problem as one cannot have a high confidence that a correspondence between two points in two different scenes is the correct one.

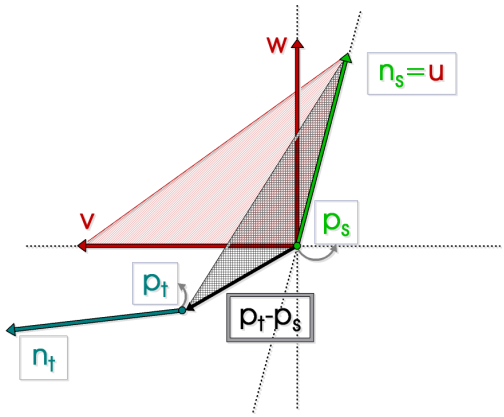


Figure 5.2: Darboux frame defined by the vectors u, v and w [61].

To solve this problem, the PFH descriptor uses 16 dimensions, computed based on histograms, which allow for more distinctive features. As with all other descriptors, operations are done on points within a radius r of the central point p. Each pair of points in the neighbourhood is examined, as in Figure 5.3. In each pair, a source point p_s and a target point p_t are determined by looking at the normals of the points, and comparing the angles relative to the line connecting the two points. The point whose normal has the smaller angle is the source, the other the target. A reference frame with the origin as the source point is defined using the normal of the source n_s as

$$u = n_s \tag{5.4}$$
$$v = (p_t - p_s) \times u \tag{5.5}$$
$$w = u \times v \tag{5.6}$$

as shown in Figure 5.2.

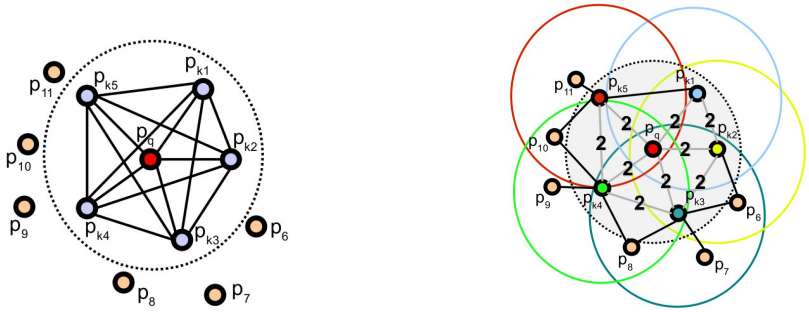
Using this frame, four features are computed

$$f_0 = v \cdot n_t \tag{5.7}$$
$$f_1 = || \, p_t - p_s \, || \tag{5.8}$$
$$f_2 = \frac{u \cdot (p_t - p_s)}{f_1} \tag{5.9}$$
$$f_3 = atan(w \cdot n_t, u \cdot n_t) \tag{5.10}$$

where n_t is the normal of the target point. These values are then used to compute an index

$$idx = \sum_{i=0}^{3} \left\lfloor \frac{f_i \cdot d}{f_{i\,\max} - f_{i\,\min}} \right\rfloor \cdot d^i \tag{5.11}$$

that defines which of the histogram bins the pair of points falls. d is the number of divisions to apply to the possible feature value range $f_{i\,\max} - f_{i\,\min}$.



(a) PFH (b) FPFH

Figure 5.3: Influence diagrams of the PFH and FPFH descriptors [58].

Because this descriptor computation operates on *all* point pairs in the neighbourhood, depending on the size of the neighbourhood and the point density, computation can be slow.

5.5 FPFH

The fast point feature histogram was introduced to tailor the PFH descriptor to real time applications while maintaining its discriminative power [58]. Instead of computing the values in Equation (5.4) for every pair of points in the neighbourhood of r, they are only computed for each point and its k neighbours p_k within the same radius — this is called the simplified point feature histogram (SPFH). Because of the use of two neighbourhoods, the final descriptor can include information from points up to $2r$ away. Once the SPFH value for each point in the neighbourhood is computed, it is integrated into the final FPFH descriptor using

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPFH(p_k) \tag{5.12}$$

where k is the number of neighbours within r and ω_k is the distance between p and p_k.

Because only direct neighbours of each point are considered, some interconnections that would exist in the PFH computation do not in FPFH, which contributes to its increased speed. The downside of this is that some of the geometric structures which exist in the region will not be included. In addition, due to the nature of the computation, the values for some points will be factored into the descriptor twice.

A further improvement made by the authors is a modification of the way the histogram is populated. Instead of using Equation (5.11) to index into an array where a single index indicates certain value ranges for all the features, they use separate histograms for each feature and concatenate them. In the original case,

there might be a large number of indices which do not contain any information, because combined feature values do not fall into the ranges. In the simplified case, there are likely to be fewer histogram indices which do not contain any information.

5.6 PFHRGB

An extension of the original PFH descriptor, this version also takes into account RGB values of points when computing features. This adds three features to the four from Equation (5.12). The computation is simple, taking the ratio of colour values between the two points p_s and p_t and ensuring that they are in the interval $[-1, 1]$.

$$f_4 = \frac{p_{s_r}}{p_{t_r}} \tag{5.13}$$

$$f_5 = \frac{p_{s_g}}{p_{t_g}} \tag{5.14}$$

$$f_6 = \frac{p_{s_b}}{p_{t_b}} \tag{5.15}$$

If one of the values f_i is outside the interval, then it is pushed back into the interval using

$$f_i = -\frac{1}{f_i} \tag{5.16}$$

These values are then added to the histogram and combined to create the final descriptor as with the basic PFH method.

Chapter 6

Object Query

The final step in the system is to use the features that have been extracted from the original clouds, and from those clouds retrieve what we think is likely to be the object that we are looking for. It is necessary to mention here the assumption we make about objects received by the system. That is, we assume the cloud that makes up the object consists of points that come only from that object and nothing else. Or, to look at this from the opposite perspective, we assume that the entirety of the cloud received by the system is the object.

This assumption changes the problem somewhat. Since our basic assumption from the beginning is that we do not have any labelled in the system, it would be much more complex for the system to receive, for example, a single frame from a scan of a room. In order to perform object query in this scenario, it would be necessary to have a definition of "objectness", which could facilitate the segmentation of the frame into objects and non-objects. These segmented objects could then be used effectively for object query as the cloud would actually be representative of that object.

6.1 Preprocessing Objects

To ensure that the objects are in a usable form, we apply the downsampling, transformation and normal extraction steps from preprocessing. Without downsampling the feature extraction step is particularly expensive, because there are a large number of points concentrated in small regions, so even with a small radius, the number of neighbours that have to be examined is generally very large. In addition, the raw annotations have issues with a "slicing" effect caused by the positioning of the camera when the frames were taken, and its depth resolution (Figure 6.1). Using the raw clouds to compute features would most likely result in worse retrieval performance as the surface structure of the cloud is essentially a series of stacked planes, which is not actually what the surface looks like in the target clouds that we are passing to the system.

Figure 6.1: Slicing effect

6.2 Query Process

The part of the system receives two clouds; the query cloud, containing the features extracted from the cloud of the object that we are interested in finding, and the target cloud, which contains features extracted from the cloud which we wish to search for the object. For point q in the query cloud, we search the target cloud for the closest K points using a nearest neighbour search. As we do not care about the exact distance, but only the relative distances of points, we use a simplified L_2 norm which does not take the square root of distances, returning only the sum of the squares of the differences between each dimension, which speeds up computation. The nearest neighbour search uses optimised functions implemented in the FLANN library [44, 45].

In the next step, we use of the computed nearest neighbours of each point to place votes into a 3D grid overlaid onto the target cloud. We construct the grid with a width, depth and height based on the minimum and maximum values of the x, y and z dimensions of the points in the target cloud. This space is then subdivided using a step, which defines the dimensions of each of the individual voting spaces. For each neighbour descriptor, we add a vote in the grid in the cell that contains the point at which it was computed.

This voting scheme is more robust than simply taking the single nearest descriptor. In some cases there will be locations where the descriptor matches some other point in the space better than the actual corresponding point on the object we are interested in finding. By taking a number of nearest neighbours and using all of them to vote, we hope to populate the voting space in such a way that these false positive values are hidden by the true positives, leading to a space where the position of the actual object is the one where there is the highest concentration of votes. We would expect this to be the case, as while some of the descriptors will match other positions in the cloud, the majority are likely to be found in regions where the actual object is.

Once the voting grid is populated, we perform clustering in this space to extract regions which contain potential matches. The clustering performed is simple Euclidean clustering, shown in Algorithm 1. Essentially, at each point we grow a cluster by adding points to a queue until there are no more points that can be added that fall within a sphere of radius r of any point in the queue. Once no more points can be found, the queue is converted to a cluster and the process begins again with another point from the cloud. We do not directly take into account the number of votes that points have when clustering them. Instead, rather than using all of the points in the voting grid, we take only the N points which have the highest number of votes, and cluster those. If, as we would hope, most of the points with high numbers of votes correspond to the same object, then they are likely to be in

the same spatial region, and can therefore be extracted by the clustering.

Algorithm 1: Euclidean Cluster Extraction [57]

Data: k-d tree representation of cloud P, Empty cluster list C, empty queue Q, processed point list L

for $p_i \in P$, $p_i \notin L$ **do**
  $Q \longleftarrow p_i$
  for $q_i \in Q$ **do**
    Find set P_k^j of neighbours p_j of q_i where $q_i - p_j < r$ **for** $p_k^j \in P_k^j$ **do**
      if $p_k^j \notin L$ **then**
        $Q \longleftarrow p_k^j$
        $L \longleftarrow p_k^j$
  $C \longleftarrow Q$
  $Q \longleftarrow \emptyset$

Once points have been clustered, the total score of each cluster is computed by summing the votes of all the points it contains. Once this is done, we rank the clusters by their scores. In this action, we make the assumption that the larger the score, the higher chance that the cluster lies on the object. Another option is to also consider the point density by dividing the score by the total number of points in the cluster. For each cluster, we find its centroid and then remove a sphere with a radius of αr centred on this point from the target cloud. α is a small multiplier greater than 1, used to ensure that a large part of the object is captured even if the centroid is offset from the object. This extracted region should contain the object.

Chapter 7

Experimental Results

In this chapter, we will investigate the effects of different variables or descriptors on the output of different parts of the system. We will also look at the computation time of the various parts of each step. In addition, we will attempt to discuss the reasons for the variations that we observe. The goal of this section is to give the reader an insight into the behaviour of the system under different parameter settings, and more importantly, to present retrieval results from our data set.

The data that we have consists of 86 clouds of the same room, at different times over the period of approximately a month. Each cloud also has some annotated objects in the form of clouds which have the object segmented from the scene. Each object has a corresponding label so that it can be identified in any of the clouds.

All experiments were run on a Lenovo X230 with 16GB of RAM and an 2.90GHz Intel i7-3520M processor. The machine was in use throughout the time of the experiments, so there is likely to be some variation introduced into the computation time due to other processes.

7.1 Preprocessing

Although each stage in the system is important, the preprocessing step has a large impact on the performance of the system in later stages. In particular, the number of points removed from a cloud can affect the results both positively and negatively, the plane extraction in particular. If tuned correctly, it can remove most of the extraneous planes, leaving only parts of the scene which have some non-planar structure, which are most likely to be objects.

Several settings affect the result of preprocessing. The distance threshold d defines the threshold below which points are considered to be part of the plane model. The distance is computed according to Equation (3.5). Another important parameter that affects RANSAC directly is the number of iterations N. Because of the random sampling, it is possible to be unlucky with the point selections made, so retrying only a few times can result in the planes that are extracted having only a small number of points on them. However, the process of recomputing the distances

to all points takes a long time, so if N is too high, the time cost might outweigh the improved results. M is the maximum number of planes that are to be extracted from a cloud. We set a hard limit P_{min} on the minimum number of points in a plane. If an extracted plane has fewer than P_{min} points, it is discarded and RANSAC is rerun. The number of times that the system will skip a plane is also fixed. If too many planes are skipped, the extraction procedure will exit.

We also define parameters for the normal computations. The radius n_p is used when computing normals for planes, and n_f is used when extracting normals for features. During development we noticed that a higher radius on the plane extraction normals generally led to better results in terms of the number of points removed by the procedure. In contrast, for feature normals we would like to try to retain smaller details about the surfaces, so a smaller radius makes more sense in order to avoid smoothing effects.

The final parameter that is used is the leaf size l for downsampling, which determines the dimensions of the voxels.

7.1.1 Parameter Settings

We have used several different parameter settings for testing. We will refer to them using the shortened keys below.

DEF This indicates the base parameter setting from which we make changes. The settings here were determined through observation of performance during development.

- $d = 0.05$
- $N = 300$
- $M = 10$
- $P_{\text{min}} = 20000$
- $n_p = 0.08$
- $n_f = 0.03$
- $l = 0.01$

DT Reduced distance threshold for RANSAC, reduced value for the minimum number of points.

- $d = 0.02$
- $P_{\text{min}} = 8000$

NR Reduced plane normal computation radius

- $n_p = 0.04$

RI500 Increased RANSAC iterations

- $N = 500$

RI100 Decreased RANSAC iterations

- $N = 100$

RI50 Decreased RANSAC iterations

- $N = 50$

DS15 Increased downsample leafsize

- $l = 0.015$

DS15M Increased downsample leafsize, decreased minimum points

- $l = 0.015$
- $P_{\text{min}} = 9000$

DS2 Increased downsample leafsize

- $l = 0.02$

DS2M Increased downsample leafsize, decreased minimum points

- $l = 0.02$
- $P_{\text{min}} = 4500$

7.1.2 Analysis

Table 7.1 shows the time taken by various stages of preprocessing. As is clear from the table, and as expected, the majority of the time is spent on RANSAC, for plane extraction. The second most costly operation is the normal computation. From Tables 7.2 and 7.3, we can see that the largest reduction in points occurs in the downsampling step, as expected. The trimming and plane extraction also reduce the clouds by a large proportion, but the number in terms of raw points is much smaller due to the downsampling.

Looking at the timings for the decreased distance threshold (DT), we see that they are almost the same as the default setting, as we would expect. However, even though DT extracts more planes, likely as a result of the reduced minimum number of points, the size of the clouds at the end of the process is approximately 1.3 times the size of those produced by the default setting. This is because the number of inlier points on each plane is greatly reduced by the limit on the distance. In general the smaller distance threshold results in fewer points being removed from the clouds.

The NR set has a normal computation time that is much faster than those of other settings with the same downsampling size. This is because the number of points used to compute the normal at each point is smaller, and as a result the computation of the plane parallel to the normal is faster. Although NR performs

Setting	Downsample	Trim	Normals	Planes	PerPlane	NormalsF	Total
DEF	0.72±0.08	0.16±0.06	16.33±1.11	116.12±13.63	15.10±1.43	1.42±0.21	133.32±14.13
DT	0.54±0.06	0.23±0.02	12.29±1.30	114.30±18.81	13.69±1.65	1.65±0.17	127.35±19.44
NR	0.88±0.35	0.18±0.08	4.12±0.54	175.35±35.32	26.31±5.32	1.60±0.27	180.52±35.44
RI500	1.07±0.22	1.82±0.81	13.20±1.23	262.81±60.10	34.19±9.00	1.17±0.20	278.90±60.37
RI100	1.14±0.38	1.25±1.63	14.92±6.96	62.68±21.67	10.59±3.78	1.86±0.99	80.00±24.77
RI50	0.70±0.09	0.17±0.06	15.86±0.99	16.02±5.02	3.82±1.02	2.15±0.48	32.75±5.30
DS15	0.99±0.23	0.10±0.03	3.65±0.52	51.43±17.19	12.47±4.93	0.65±0.15	56.17±17.27
DS15M	0.87±0.31	0.34±0.50	4.77±2.27	100.47±116.74	13.05±16.78	0.68±0.29	106.45±116.80
DS2	0.58±0.07	0.03±0.00	1.31±0.14	13.32±2.06	7.71±1.56	0.41±0.06	15.24±2.08
DS2M	0.56±0.10	0.03±0.00	1.23±0.25	32.13±4.06	4.26±0.61	0.23±0.05	33.96±4.27

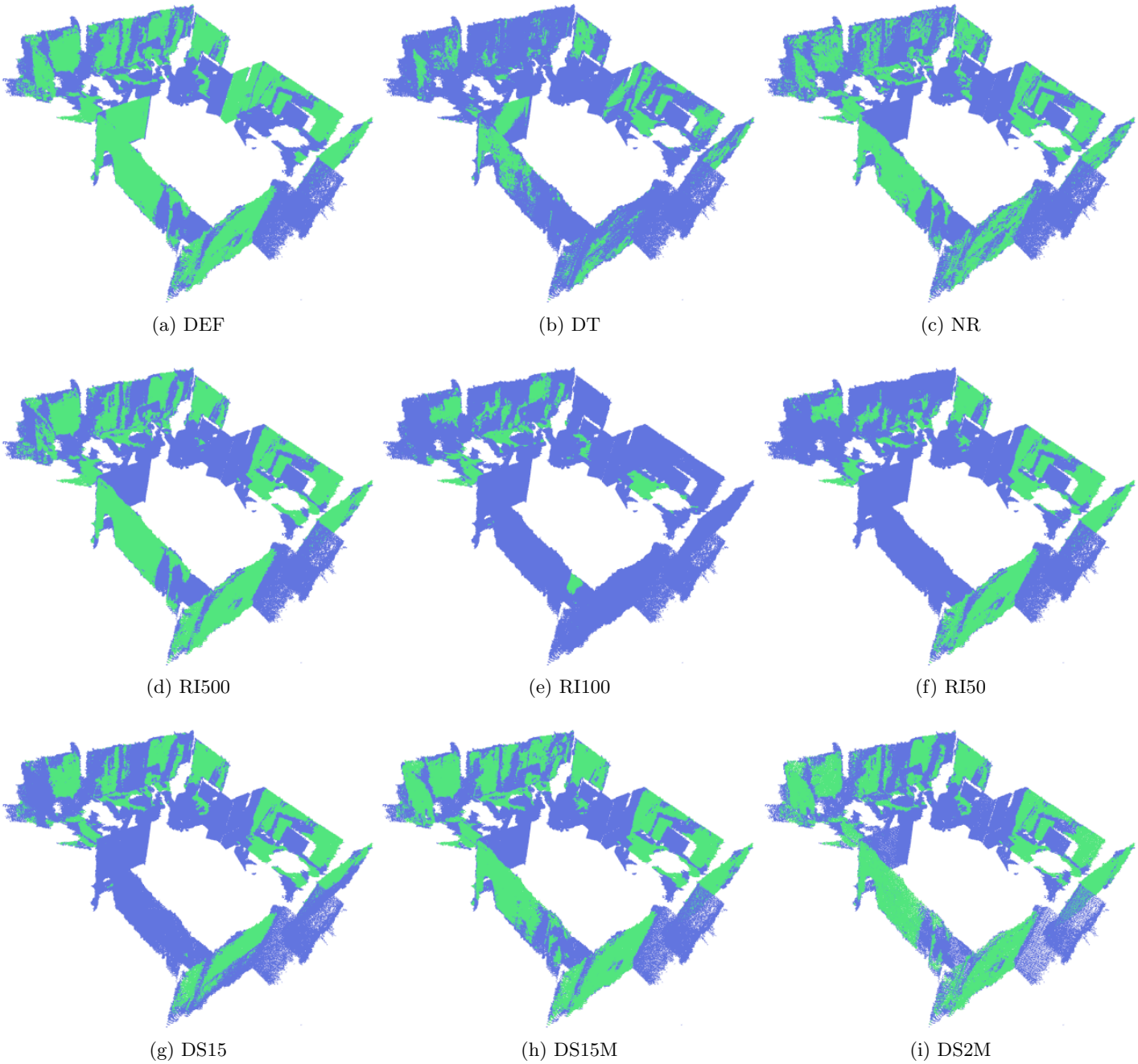
Table 7.1: Average time taken for various stages of preprocessing. The total time is a sum of downsample, trim, normals and planes, skipping the feature normal computation. Average over 86 clouds. ($\mu \pm \sigma$)

Setting	Original	Downsampled	Trimmed	NPlanes	PerPlane	Planes
DEF	4,347,665±125,147	988,703±40,359	783,173±34,610	7.78±1.30	50,542±6,301	394,992±46,514
DT	——— " ———	——— " ———	——— " ———	8.42±1.87	20,114±2,382	615,910±40,667
NR	——— " ———	——— " ———	——— " ———	6.75±1.03	44,261±4,269	508,862±48,995
RI500	——— " ———	——— " ———	——— " ———	7.74±1.10	52,250±6,407	386,967±39,903
RI100	——— " ———	——— " ———	——— " ———	6.22±1.85	45,224±9,510	519,703±102,722
RI50	——— " ———	——— " ———	——— " ———	4.60±2.32	38,566±15,670	583,620±115,296
DS15	——— " ———	485,732±20,386	368,765±16,645	4.42±1.53	26,500±4,772	248,010±45,101
DS15M	——— " ———	——— " ———	——— " ———	7.90±1.20	22,657±3,172	192,516±16,784
DS2	——— " ———	276,378±11,503	203,385±8,910	1.58±0.94	10,766±6,312	180,607±17,235
DS2M	——— " ———	——— " ———	——— " ———	7.84±1.29	12,001±1,505	110,543±12,245

Table 7.2: Number of points in various stages of preprocessing, taken as an average over 86 clouds. NPlanes indicates the number of planes extracted. PerPlane is the number of points per plane, computed from the number of planes and the difference in number of points between Trimmed and Planes. Average over 86 clouds. ($\mu \pm \sigma$)

Setting	Downsample	Trim	Trim Orig	Plane	Plane Orig
DEF	0.23±0.01	0.79±0.01	0.18±0.00	0.50±0.06	0.09±0.01
DT	—— " ——	—— " ——	—— " ——	0.79±0.04	0.14±0.01
NR	—— " ——	—— " ——	—— " ——	0.63±0.05	0.12±0.01
RI500	—— " ——	—— " ——	—— " ——	0.49±0.05	0.09±0.01
RI100	—— " ——	—— " ——	—— " ——	0.65±0.12	0.12±0.02
RI50	—— " ——	—— " ——	—— " ——	0.75±0.15	0.13±0.03
DS15	0.11±0.00	0.76±0.02	0.08±0.00	0.67±0.12	0.06±0.01
DS15M	—— " ——	—— " ——	—— " ——	0.52±0.04	0.04±0.00
DS2	0.06±0.00	0.74±0.02	0.05±0.00	0.89±0.07	0.04±0.00
DS2M	—— " ——	—— " ——	—— " ——	0.54±0.06	0.03±0.00

Table 7.3: Reduction of point numbers as proportion of cloud size at previous step (columns 1,2 and 4) and original cloud (columns 3 and 5), as an average over 86 clouds. ($\mu \pm \sigma$)



(a) DEF (b) DT (c) NR

(d) RI500 (e) RI100 (f) RI50

(g) DS15 (h) DS15M (i) DS2M

Figure 7.1: Comparison of the result of plane extraction on the same arbitrarily chosen cloud for different parameter settings. Green regions are extracted planes, blue regions are the remaining cloud which is output by the preprocessing step. DS2 result not shown as no planes were extracted.

better in terms of reduction than DT, it is still not as good as the default setting from that perspective. This is because the smaller normal radius results in less smoothing, which means that points which are actually on planes may have noise on the normals which result in them not being added. While the NR set should have timings for planes similar to DEF and DT, the plane computation time seems to have increased a lot. We do not have a satisfactory explanation for this — we ran a second shorter experiment to check the timings, and the results were similar. This is an unexpected result as the number of planes extracted on average is also very similar. Furthermore, the internal computations done by RANSAC should not be affected, since the same number of normals must be compared in each case. We are unable to determine the reason behind this effect.

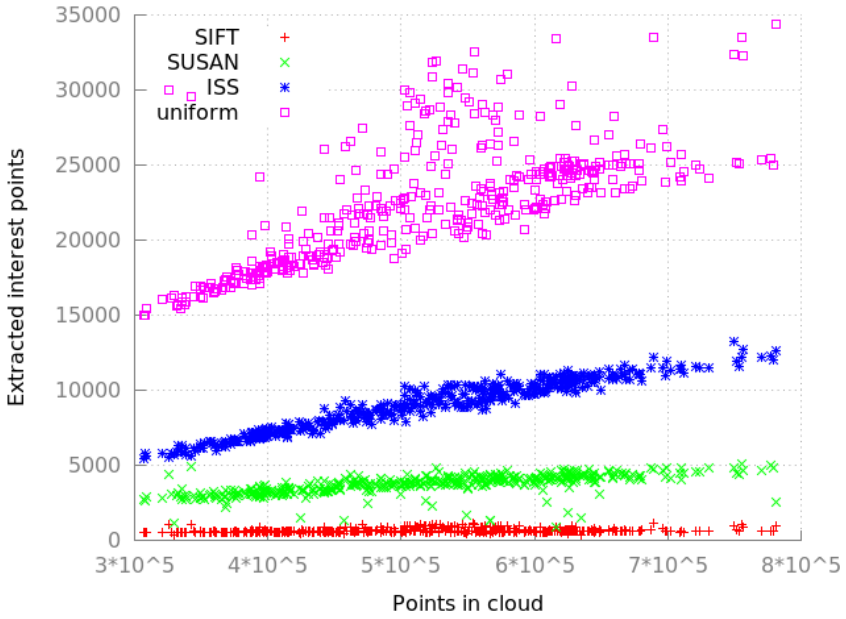
The RI sets look at the effect of the RANSAC iterations, and as expected it is very noticeable. RI500 has a much increased computation time due to the increased iterations, but the reduction that results is not much better than the result using only 300 iterations in the DEF set. We can see that increasing the number of iterations has diminishing returns in terms of the increased number of points removed versus the additional computation time. Decreasing the number of iterations has the opposite effect — reducing computation time, but also reducing the number of points removed.

In the DS sets, we look at the effect of the leaf size used for downsampling. Increasing the leaf size greatly reduces the number of points in the cloud, as is to be expected. This also affects subsequent steps, as both normal computation and RANSAC examine fewer points. In terms of the proportion of points removed, for DS15 and DS2 we did not reduce the minimum points per plane, and this effect shows in the results. Fewer planes were removed, and as a consequence the effectiveness of the plane extraction is reduced. In the DS2M and DS15M sets, we decrease the minimum points per plane, and the proportions are more in line with what would be expected, extracting proportionally almost as many points as that extracted by the default setting.

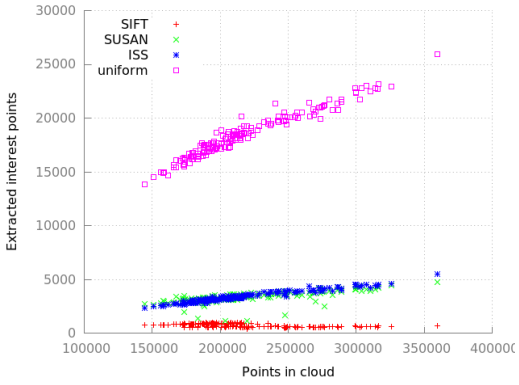
Figure 7.1 shows examples of planes extracted using the different parameter settings, and the clouds that go on to the next stage of processing.

7.2 Feature Extraction

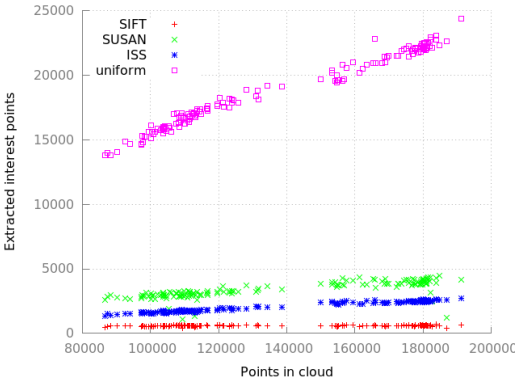
7.3 Object Query



(a) 0.01m

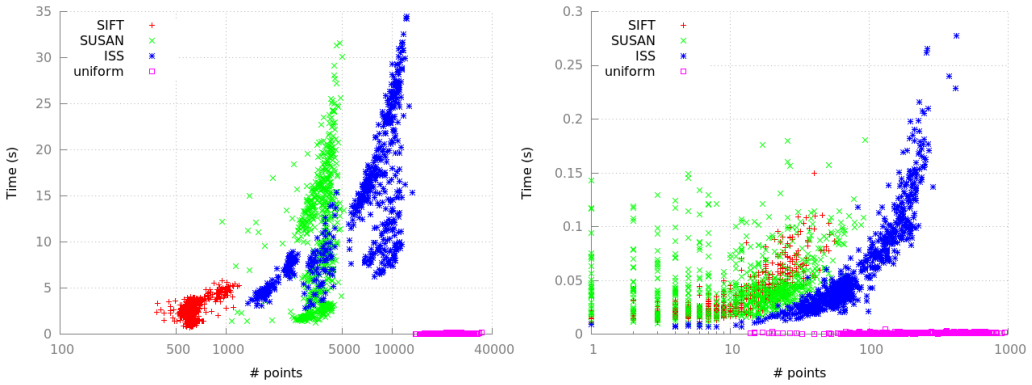


(b) 0.015m



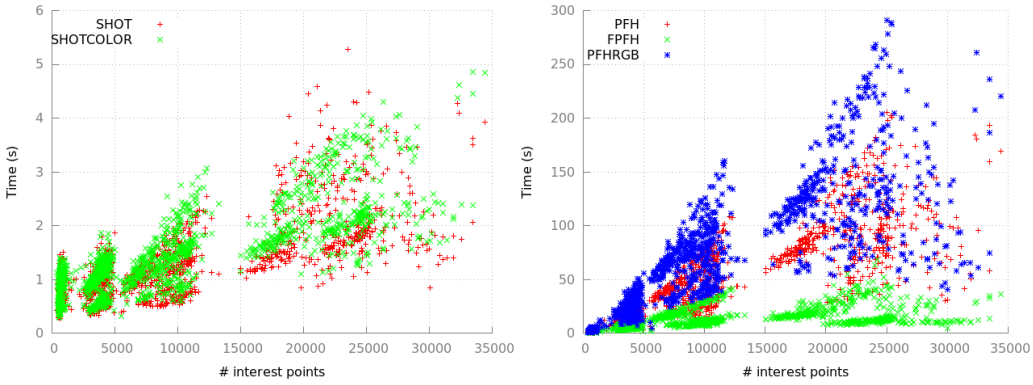
(c) 0.02m

Figure 7.2: Number of points extracted from clouds by different methods relative to the number of points in the cloud. The anomalies from the uniform extraction come from the fact that it does not exclude lone points, while other methods do. Subcaption indicates downsample leaf size.



(a) Scene clouds (b) Annotation clouds

Figure 7.3: Time taken for interest point extraction for different numbers of points for the various interest point methods, shown on a logarithmic scale. All clouds downsampled with 0.01 leaf size. Each point indicates the time to compute interest points for a single cloud.



(a) SHOT family (b) PFH family

Figure 7.4: Computation time for each feature type in relation to number of interest points. All clouds downsampled with 0.01 leaf size.

Chapter 8

Conclusion and Further Work

Describe what the project was about, what was achieved, summarise the experiments, describe what can be improved.

Bibliography

[1] Radhakrishna Achanta et al. "SLIC superpixels compared to state-of-the-art superpixel methods". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.11 (2012), pp. 2274–2282.

[2] Aitor Aldoma et al. "CAD-model recognition and 6DOF pose estimation using 3D cues". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 585–592.

[3] Sunil Arya et al. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 891–923.

[4] Herbert Bay et al. "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[5] Jeffrey S Beis and David G Lowe. "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces". In: *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE. 1997, pp. 1000–1006.

[6] Serge Belongie, Jitendra Malik, and Jan Puzicha. "Shape matching and object recognition using shape contexts". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.4 (2002), pp. 509–522.

[7] Mirela Ben-Chen and Craig Gotsman. "Characterizing Shape Using Conformal Factors." In: *3DOR*. 2008, pp. 1–8.

[8] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[9] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. "Depth kernel descriptors for object recognition". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 821–826.

[10] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. "Kernel descriptors for visual recognition". In: *Advances in Neural Information Processing Systems*. 2010, pp. 244–252.

[11] Oren Boiman, Eli Shechtman, and Michal Irani. "In defense of nearest-neighbor based image classification". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[12] C Mic Bowman, Peter B Danzig, and Michael F Schwartz. *Research problems for scalable internet resource discovery*. Tech. rep. DTIC Document, 1993.

[13] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer networks and ISDN systems* 30.1 (1998), pp. 107–117.

[14] Rasmus Bro, Evrim Acar, and Tamara G Kolda. "Resolving the sign ambiguity in the singular value decomposition". In: *Journal of Chemometrics* 22.2 (2008), pp. 135–140.

[15] Yizong Cheng. "Mean shift, mode seeking, and clustering". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17.8 (1995), pp. 790–799.

[16] Chin Seng Chua and Ray Jarvis. "Point signatures: A new representation for 3d object recognition". In: *International Journal of Computer Vision* 25.1 (1997), pp. 63–85.

[17] Haili Chui and Anand Rangarajan. "A new point matching algorithm for non-rigid registration". In: *Computer Vision and Image Understanding* 89.2 (2003), pp. 114–141.

[18] Ondrej Chum and Jiri Matas. "Matching with PROSAC-progressive sample consensus". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 220–226.

[19] Gregory Cipriano, George N Phillips, and Michael Gleicher. "Multi-scale surface descriptors". In: *Visualization and Computer Graphics, IEEE Transactions on* 15.6 (2009), pp. 1201–1208.

[20] Dorin Comaniciu and Peter Meer. "Mean shift: A robust approach toward feature space analysis". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.5 (2002), pp. 603–619.

[21] Bertram Drost et al. "Model globally, match locally: Efficient and robust 3D object recognition". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 998–1005.

[22] Facebook, Inc. *Form S-1 Registration Statement Under The Securities Act of 1933: Facebook, Inc.* 2012. URL: `http://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm` (visited on 05/12/2015).

[23] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[24] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. "An algorithm for finding best matches in logarithmic expected time". In: *ACM Transactions on Mathematical Software (TOMS)* 3.3 (1977), pp. 209–226.

[25] Andrea Frome et al. "Recognizing objects in range data using regional point descriptors". In: *Computer Vision-ECCV 2004*. Springer, 2004, pp. 224–237.

[26] Keinosuke Fukunaga and Larry Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *Information Theory, IEEE Transactions on* 21.1 (1975), pp. 32–40.

[27] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. "Class segmentation and object localization with superpixel neighborhoods". In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 670–677.

[28] Thomas Funkhouser and Philip Shilane. "Partial matching of 3 D shapes with priority-driven search". In: *ACM International Conference Proceeding Series*. Vol. 256. Citeseer. 2006, pp. 131–142.

[29] Natasha Gelfand et al. "Robust global registration". In: *Symposium on geometry processing*. Vol. 2. 3. 2005, p. 5.

[30] Shengyin Gu et al. "Surface-histogram: A new shape descriptor for protein-protein docking". In: *Proteins: Structure, Function, and Bioinformatics* 80.1 (2012), pp. 221–238.

[31] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. Manchester, UK. 1988, p. 50.

[32] Stan Horaczek. *How Many Photos Are Uploaded to The Internet Every Minute?* 2013. URL: `http://www.popphoto.com/news/2013/05/how-many-photos-are-uploaded-to-internet-every-minute` (visited on 05/12/2015).

[33] Cheuk Yiu Ip et al. "Using shape distributions to compare solid models". In: *Proceedings of the seventh ACM symposium on Solid modeling and applications*. ACM. 2002, pp. 273–280.

[34] Yushi Jing and Shumeet Baluja. "Pagerank for product image search". In: *Proceedings of the 17th international conference on World Wide Web*. ACM. 2008, pp. 307–316.

[35] Andrew E. Johnson and Martial Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.5 (1999), pp. 433–449.

[36] Andrew Edie Johnson. "Spin-images: a representation for 3-D surface matching". PhD thesis. Citeseer, 1997.

[37] Jan Knopp et al. "Hough transform and 3D SURF for robust three dimensional classification". In: *Computer Vision–ECCV 2010*. Springer, 2010, pp. 589–602.

[38] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. "A sparse texture representation using local affine regions". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 27.8 (2005), pp. 1265–1278.

[39] Michael S Lew et al. "Content-based multimedia information retrieval: State of the art and challenges". In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 2.1 (2006), pp. 1–19.

[40] Tony Lindeberg. "Feature detection with automatic scale selection". In: *International journal of computer vision* 30.2 (1998), pp. 79–116.

[41] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[42] Sancho McCann and David G Lowe. "Local naive bayes nearest neighbor for image classification". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 3650–3656.

[43] Krystian Mikolajczyk and Cordelia Schmid. "Scale & affine invariant interest point detectors". In: *International journal of computer vision* 60.1 (2004), pp. 63–86.

[44] Marius Muja and David G Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." In: *VISAPP (1)* 2 (2009).

[45] Marius Muja and David G. Lowe. "Scalable Nearest Neighbor Algorithms for High Dimensional Data". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (2014), pp. 2227–40.

[46] David Nister and Henrik Stewenius. "Scalable recognition with a vocabulary tree". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2006, pp. 2161–2168.

[47] Stephen Malvern Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[48] opencv.org. *OpenCV*. URL: `http://www.opencv.org` (visited on 05/13/2015).

[49] Robert Osada et al. "Shape distributions". In: *ACM Transactions on Graphics (TOG)* 21.4 (2002), pp. 807–832.

[50] Maks Ovsjanikov et al. "Shape Google: a computer vision approach to invariant shape retrieval". In: *Proc. NORDIA* 1.2 (2009).

[51] Jeremie Papon et al. "Voxel cloud connectivity segmentation-supervoxels for point clouds". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 2027–2034.

[52] James Philbin et al. "Object retrieval with large vocabularies and fast spatial matching". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[53] Brian Pinkerton. "Finding what people want: Experiences with the WebCrawler". In: *Proceedings of the Second International World Wide Web Conference*. Vol. 94. Chicago. 1994, pp. 17–20.

[54] pointclouds.org. *PCL keypoints library*. URL: `http://docs.pointclouds.org/trunk/group__keypoints.html` (visited on 05/19/2015).

[55] pointclouds.org. *Point Cloud Library*. URL: `http://www.pointclouds.org` (visited on 05/13/2015).

[56] Yong Rui and Thomas Huang. "Optimizing learning in image retrieval". In: *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*. Vol. 1. IEEE. 2000, pp. 236–243.

[57] Radu Bogdan Rusu. "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments". PhD thesis. Computer Science department, Technische Universitaet Muenchen, Germany, 2009.

[58] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast point feature histograms (FPFH) for 3D registration". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 3212–3217.

[59] Radu Bogdan Rusu et al. "Aligning point cloud views using persistent feature histograms". In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 3384–3391.

[60] Radu Bogdan Rusu et al. "Fast 3d recognition and pose using the viewpoint feature histogram". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 2155–2162.

[61] Radu Bogdan Rusu et al. "Learning informative point classes for the acquisition of object model maps". In: *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*. IEEE. 2008, pp. 643–650.

[62] Radu Bogdan Rusu et al. "Persistent point feature histograms for 3D point clouds". In: *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*. 2008, pp. 119–128.

[63] Zujun Shentu et al. "Context shapes: Efficient complementary shape matching for protein–protein docking". In: *Proteins: Structure, Function, and Bioinformatics* 70.3 (2008), pp. 1056–1073.

[64] Philip Shilane and Thomas Funkhouser. "Distinctive regions of 3D surfaces". In: *ACM Transactions on Graphics (TOG)* 26.2 (2007), p. 7.

[65] Chanop Silpa-Anan and Richard Hartley. "Optimised KD-trees for fast image descriptor matching". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[66] Ivan Sipiran and Benjamin Bustos. "Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes". In: *The Visual Computer* 27.11 (2011), pp. 963–976.

[67] Josef Sivic and Andrew Zisserman. "Video Google: A text retrieval approach to object matching in videos". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE. 2003, pp. 1470–1477.

[68] Bill Slawski. *Just What Was The First Search Engine?* May 2006. URL: `http://www.seobythesea.com/2006/02/just-what-was-the-first-search-engine/` (visited on 05/11/2015).

[69] Stephen M Smith and J Michael Brady. "SUSANa new approach to low level image processing". In: *International journal of computer vision* 23.1 (1997), pp. 45–78.

[70] Bastian Steder et al. "Point feature extraction on 3D range scans taking into account object boundaries". In: *Robotics and automation (icra), 2011 ieee international conference on*. IEEE. 2011, pp. 2601–2608.

[71] Bastian Steder et al. "Robust on-line model-based object detection from range images". In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE. 2009, pp. 4739–4744.

[72] Fridtjof Stein and Gérard Medioni. "Structural indexing: Efficient 3-D object recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 125–145.

[73] Danny Stieben. *The Archie Search Engine — The World's First Search!* May 2013. URL: `http://www.makeuseof.com/tag/the-archie-search-engine-the-worlds-first-search/` (visited on 05/11/2015).

[74] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. "A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion". In: *Computer graphics forum*. Vol. 28. 5. Wiley Online Library. 2009, pp. 1383–1392.

[75] Federico Tombari, Samuele Salti, and Luigi Di Stefano. "A combined texture-shape descriptor for enhanced 3D feature matching". In: *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE. 2011, pp. 809–812.

[76] Federico Tombari, Samuele Salti, and Luigi Di Stefano. "Unique shape context for 3D data description". In: *Proceedings of the ACM workshop on 3D object retrieval*. ACM. 2010, pp. 57–62.

[77] Federico Tombari, Samuele Salti, and Luigi Di Stefano. "Unique signatures of histograms for local surface description". In: *Computer Vision–ECCV 2010*. Springer, 2010, pp. 356–369.

[78] Philip HS Torr and Andrew Zisserman. "MLESAC: A new robust estimator with application to estimating image geometry". In: *Computer Vision and Image Understanding* 78.1 (2000), pp. 138–156.

[79] Eric W. Weisstein. *Plane*. URL: `http://mathworld.wolfram.com/Plane.html` (visited on 05/18/2015).

[80] RepRap Wiki. *Printable Part Sources*. URL: `http://reprap.org/wiki/Printable_part_sources` (visited on 05/13/2015).

[81] Walter Wohlkinger and Markus Vincze. "Ensemble of shape functions for 3d object classification". In: *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. 2011, pp. 2987–2992.

[82] Walter Wohlkinger and Markus Vincze. "Shape distributions on voxel surfaces for 3D object classification from depth images". In: *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 115–120.

[83] Y. Zhang and R.P. Loce. *SUSAN-based corner sharpening*. US Patent 8,456,711. 2013. URL: `http://www.google.com/patents/US8456711`.

[84] Yu Zhong. "Intrinsic shape signatures: A shape descriptor for 3D object recognition". In: *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 689–696.